Partial Boolean Functions for QBF Semantics

Allen Van Gelder

Computer Science Dept., SOE–3, Univ. of California,
Santa Cruz, CA 95064,
avg@cs.ucsc.edu

Abstract

Long-distance resolution for quantified boolean formula (QBF) solving was introduced in 2002 by Zhang and Malik, but has been controversial for the following decade, because it derives and uses tautologous clauses. Balabanov and Jiang (2012) gave a set of proof rules (called LDQ-resolution) that include long-distance resolution with conditions, but did not attach any "meaning" (i.e., semantic interpretation) to the derived tautologous clauses. Egly, Lonsing and Widl (2013) showed that the QBF certificate extraction algorithm of Goutltiaeva, Van Gelder and Bacchus (2011) could be applied with correct results to LDQ refutations. These results and others have brought LDQ-resolution back into the mainstream. This paper introduces partial boolean functions (pbfs) and develops a semantic interpretation for universal literals in QBF clauses. It develops LDP-resolutioni, which is refutationally complete for QBF formulas in prenex conjunction normal form (PCNF). LDP-resolution is shown to derive only logical consequences.

1 Introduction

Quantified boolean formulas (QBFs) provide a natural expression of many AI problems on finite domains, such as planning, hypothetical reasoning, counterfactual reasoning, game analysis, quantified constraint satisfaction, and various forms of verification. Practical tools for QBF reasoning are of interest and value in a variety of constraint-satisfaction problems.

Long-distance resolution for quantified boolean formula (QBF) solving (see Section 2 for definitions) was introduced more than a decade ago [23], but has been controversial for the following decade, because it derives and uses tautologous clauses. Balabanov and Jiang gave a set of proof rules (called *LDQ-resolution*) that include long-distance resolution with conditions [2]. They claimed that any LDQ-resolution refutation can be transformed into a Q-resolution refutation. Further work showed that LDQ-resolution can be implemented efficiently in a search-based QBF solver and may generate exponentially shorter refutations than Q-resolution on a certain family of formulas [8]. Peitl *et al.* [15] showed that it may be combined with the *standard dependency scheme* [17, 13]. Beyersdorff and Blinkhorn used the idea of fully exhibiting model trees to show that it may be combined with the *RRS dependency scheme* [3]. Blinkhorn and Beyersdorff showed that combining formula restriction with the *RRS dependency scheme* can exponentially shorten refutations in a variant of a well-studied QBF family, even without long-distance resolution [5].

None of this work addressed the question of whether clauses or similar formulas derived by these proof systems were ***logical consequences*** of the original QBF; they only considered whether derived formulas preserved the *truth value* of a closed QBF. A derived formula is a logical consequence of an original formula if conjoining the derived formula with the original formula does not reduce the set of model trees.

Definition 1 We say a system is *strategy sound* if it only derives logical consequences of the original formula. The qualifier "strategy" is included because many papers consider a system to be "sound" if it does not return an incorrect truth value. That is, only one bit of the output is considered important. □

It is known in the literature that the Q-resolution proof system, dating from 1995, derives only logical consequences. Recent years have seen the introduction of several QBF proof systems that might have exponentially shorter refutations than are possible with Q-resolution on certain formula families. This paper introduces ***partial boolean functions*** (***pbfs***) and develops a semantic interpretation for universal literals in QBF clauses based on pbfs. A special case of pbfs is described, called ***partial Herbrand functions*** (***phfs***). It introduces a long-distance resolution system using phfs called LDP-resolution that is "strategy sound," i.e., it derives only logical consequences of the original QBF formula. LDP applies to QBF formulas in prenex conjunction normal form (PCNF).

It is then shown that some well known systems may derive formulas that are **not** logical consequences, although they do not change the truth value of the entire original formula.

The importance of this distinction becomes apparent if the original formula is a subformula in the overall logic for a large project: A logical consequence may be added to the original subformula without changing its set of model trees, and may be useful in its own right in other parts of the project. It further demonstrates that universal reductions based on several more aggressive dependency schemes found in the literature [22, 20], even without any form of long-distance resolution, lead to derived clauses that are *not* necessarily logical consequences.

QBF can be formulated as a decidable fragment of full first-order logic. Future work may include study of other decidable fragments to see if partial Herbrand functions or partial Skolem functions have a useful role as replacements for total functions. For example, Samer has considered quantified constraint-satisfaction problems in which variables take values in finite domains [16].

2 Basic Definitions and Notation

This paper uses standard notation as much as possible, but contains many specific definitions and notations that are used throughout the paper. Page limits force us to assume that the reader is familiar with the common definitions related to QBFs in the literature.

This paper uses capitalized Greek letters Ψ, Φ, and Θ to denote QBFs. We use 0 and 1 for truth values of literals and use *true* and *false* for semantic values of formulas.

We follow certain notational conventions for boolean variables and literals (signed variables) to make reading easier: Lowercase letters near the ***beginning*** of the alphabet (e.g., b, c, d, e) denote existential literals, while lowercase letters near the ***end*** of the alphabet (e.g., u, v, w, x) denote universal literals, while ***middle*** letters (e.g., p, q, r) are of ambiguous quantifier type. Quantifier types are implied frequently throughout the paper without restating this convention.

In contexts where a literal is expected, p might denote a positive or negative literal, while $\overline{p}$ denotes the negation of p. To emphasize that p stands for a ***variable***, rather than a *literal*, the notation $|p|$ is used. Clauses may be written as $[p, q, \overline{r}]$); $[]$ denotes the empty clause; $[1]$ denotes an existentially tautologous clause.

For this paper we require the ***original QBF*** (to be solved) to be ***closed*** (i.e., with no free variables), in ***prenex conjunctive normal form*** (**PCNF**), and have a ***matrix*** of ***non-tautologous original clauses*** that are ***completely reduced***, which means that every *universal* literal in the clause that is inner to every *existential* literal in the same clause is removed. Derived formulas may be in a slightly more general prenex form.

Perhaps the most constructive way to think of QBF semantics is as a two-player game between the E-player, who sets existential variables, and the A-player, who sets universal variables. Repeatedly, the outermost unset variable in the prenex is set to 0 or 1 by the appropriate player. The E-player tries to make the matrix eventually evaluate to *true*, which requires *every* clause to be satisfied, whereas the A-player tries to make the matrix eventually evaluate to $false$, which is accomplished by *some* clause being falsified.

Practical proof systems focus on proving that the A-player has a winning strategy for the original QBF Φ, which is synonymous with Φ being *false*. When the A-player needs to choose a value for an outermost unset universal variable u, the values of all *existential* variables outer to u have been set. Therefore the choice made by the A-player can be thought of as a boolean function of these outer existential variables, say $u(e_1, e_2, \ldots)$. Such functions are called ***Herbrand functions*** for u.

A winning A-strategy is a complete set of Herbrand functions, one for each universal variable, such that setting the universal variables according to these functions during a play of the game ensures that Φ evaluates to *false* no matter how the E-player chooses. Now suppose each occurrence of a universal variable u in the matrix of Φ is replaced by the corresponding Herbrand function $u(e_1, e_2, \ldots)$. The resulting Φ has entirely existential variables and is *false* if and only if it is unsatisfiable.

Historically, Herbrand functions are total boolean functions but in practice partial functions are often sufficient to specify a winning A-strategy. This paper introduces a theory based on partial boolean functions and partial Herbrand functions that avoids certain logical problems that arise when total Herbrand functions are used. Several technical definitions are needed.

Definition 2 For this paper the prenex is *totally ordered*. The ***qdepth***, or ***scope***, of a variable is its alternation depth, with 1 outermost. Notations: $p \prec q$ means $qdepth(p) < qdepth(q)$; $p \preceq q$ means $qdepth(p) \leq qdepth(q)$; $p \prec\!\!\prec q$ means p precedes q in the totally ordered prenex.

A few special operations involving a set of literals S are defined, assuming the prenex $\overrightarrow{Q}$ is known by the context.

$$\begin{aligned}
\mathsf{exist}(S) &= \{\text{the existential literals in } S\} && (1)\\
\mathsf{univ}(S) &= \{\text{the universal literals in } S\} && (2)\\
(S \prec q) &= \{\text{the literals in } S \text{ outer to } q\} && (3)\\
(q \prec S) &= \{\text{the literals in } S \text{ inner to } q\} && (4)
\end{aligned}$$

Depending on context, S might be a clause, a prenex, a partial assignment, or other logical expression. $\square$

Definition 3 A ***prenex-ordered assignment*** is a *total* assignment that is represented by a sequence of literals that are assigned 1 and are in prenex order. $\square$

A strategy for the E-player can be represented as an E-assignment tree, which we now describe. (See also [18, 19]). We say that a ***branch*** is a path from the root node to some leaf node, represented as a sequence of tree edges, which are labeled with a prenex-ordered assignment.

Although it seems artificial at first, it is very useful to represent a tree as the nonempty set of its branches. A ***branch prefix*** is a path (sequence of edges) from the root node that might terminate before reaching a leaf. The same branch prefix can occur in many branches, and represents a tree node; the empty branch prefix is the tree root.

Definition 4 Let a QBF $\Phi = \overrightarrow{Q}.\mathcal{F}$ be given, with $k = |\mathsf{univ}(\Phi)|$. An ***E-assignment tree*** T for Φ is a set of exactly 2^k prenex-ordered assignments for Φ that defines a tree and satisfies these constraints on branching:

1. Let (σ, e) denote a branch prefix in T. Then *no* branch of T has the prefix $(\sigma, \overline{e})$.
2. Let (σ, u) denote a branch prefix in T. Then *some* branch of T has the prefix $(\sigma, \overline{u})$.

A ***model tree*** M for Φ is an E-assignment tree in which each branch τ makes $\mathcal{F}$ *true*, i.e., $\tau \models \mathcal{F}$ in the usual propositional sense.

Although the wording is different, if M is a model tree by this definition, it is also a model tree by definitions found in other papers [19]. Every model tree M represents a winning E-strategy in the obvious way.

Definition 5 Departing from the CNF formalism, we denote the ***if-then-else*** boolean operator on three parameters (or gate with three inputs) by $\mathit{ite}(C,\, T,\, F)$. Here C, T, and F are themselves boolean formulas. For any partial assignment τ:

if $C\lceil_\tau = 1$, then $\mathit{ite}(C,\, T,\, F)\lceil_\tau = T\lceil_\tau$;
if $C\lceil_\tau = 0$, then $\mathit{ite}(C,\, T,\, F)\lceil_\tau = F\lceil_\tau$.

See Definition 12 for abbreviated forms of *ite* denoted by $\mathit{if}(C,\, T)$ and $\mathit{cp}(T)$. □

The *ite* operator is basic to many programming languages and is popular for circuit design, due to nice properties. For example, it can simulate unary and binary boolean formulas, e.g., $(A \wedge B) \equiv \mathit{ite}(A,\, B,\, 0)$, $\mathit{xor}(A,\, B) \equiv \mathit{ite}(A,\, \neg(B),\, B)$, *et al.* It is famous as the foundation for binary decision diagrams (BDDs) developed in the pioneering work of Bryant [6]. Recently it has played a prominent role in works on QBF [7, 11]. For this paper, *ite* and its formulation in terms of restriction are important for the development of partial Herbrand functions in derived clauses.

We assume the reader is familiar with the *Q-resolution* proof system, with two operations, *resolution* and *universal reduction*. Q-resolution is *refutationally* complete [KBKF95], but not *inferentially* complete. That is, $\Phi \models C$ might hold for a clause C, yet neither C nor a clause that subsumes C is derivable by Q-resolution. Also, Q-resolution prohibits derivation of a tautologous clause.

Definition 6 ***Resolution*** is defined as usual for propositional clauses, but the notation used is more specific than usual, to facilitate analysis of proofs. Let clauses $C_1 = [\overline{p},\, \alpha]$ and $C_2 = [p,\, \beta]$, where α and β are (possibly empty) literal sequences containing neither p nor $\overline{p}$. Then the ***resolvent*** is: $\mathsf{res}_p(C_1,\, C_2) = \alpha \cup \beta$.

Variable $|p|$ and its literals are called the ***clashing variable*** and ***clashing literals***. Note that the second operand is always the clause that contains p. This paper considers only existential clashing literals, but considers proof systems that permit tautologous resolvents under certain conditions. □

Definition 7 A universal literal u is said to be ***tailing*** in a clause C if its *qdepth* is greater than that of any existential literal in C, i.e., $(u \prec \mathsf{exist}(C)) = \emptyset$.

Universal reduction removes tailing universal literals from clauses. If u is *tailing* in $C_3 = [\alpha,\, u]$, then the ***reduced clause*** is: $\mathsf{unrd}_u(C_3) = \alpha$.

In addition $\mathsf{unrd}_*(C_3)$ denotes the *completely reduced* clause that results from applying unrd successively on all tailing universal literals in C_3. □

Lemma 8 Resolution and universal reduction are strategy-sound operations. ■

Proof: Straightforward application of the definitions. ■

Definition 9 We now define an extension of resolution that makes it a total function so that operations in a resolution proof remain well defined after a restriction is applied to the leaf clauses [21, 9].. As with *Q-resolution*, the clashing literal is required to be existential. To distinguish from standard resolution we call the new operator ***total resolution*** and denote it as "trex".

As a technicality, the clause [?] means disallowed operands, and should never occur. Also [1] represents a clause with the literal 1 in it, due to a restriction; it is treated as the widest possible clause and always evaluates to *true*. If normal resolution is defined, trex is the same. The other cases are summarized next.

If the clashing literal is present in one operand and not the other, and neither operand is [1], the resolvent is the operand *without* a clashing literal. If one operand is [1] and the other contains its clashing literal, or both operands are [1], the resolvent is [1]. If one operand is [1] and the other *lacks* its clashing literal, the other clause is the resolvent. Finally, if both operands are regular clauses that lack their clashing literal, the clause with fewer literals becomes the resolvent; ties are broken by some deterministic procedure. These cases explain the importance of specifying the clashing literal in the notation. □

Lemma 10 If $D = \mathsf{trex}_p(C_1,\, C_2)$, then

$$\mathsf{res}_p\,(C_1\lceil_\tau,\, C_2\lceil_\tau) \;\subseteq\; D\lceil_\tau. \tag{5}$$

Proof: Straightforward application of the definitions. ■

2.1 Boolean If-Then-Else

Definition 11 Departing from the CNF formalism, we denote the ***if-then-else*** boolean operator on three parameters (or gate with three inputs) by $\mathit{ite}(C,\, T,\, F)$. Here C, T, and F are themselves boolean formulas. For any partial assignment τ:

if $C\lceil_\tau = 1$, then $\mathit{ite}(C,\, T,\, F)\lceil_\tau = T\lceil_\tau$;
if $C\lceil_\tau = 0$, then $\mathit{ite}(C,\, T,\, F)\lceil_\tau = F\lceil_\tau$. □

This operator is basic to many programming languages and is popular for circuit design, due to nice properties. For example, it can simulate unary and binary boolean formulas, e.g., $(A \wedge B) \equiv \mathit{ite}(A,\, B,\, 0)$, $\mathit{xor}(A,\, B) \equiv \mathit{ite}(A,\, \neg(B),\, B)$, *et al.* It is famous as the foundation for binary decision diagrams (BDDs) developed in the pioneering work of Bryant [6]. Recently it has played a prominent role in works on QBF [7, 11].

The *ite* operator has several implementations using two-parameter boolean operators, but it is often most natural as a three-parameter operator. Well known representations in DNF and CNF are:

$$\textbf{DNF:}\ \mathit{ite}(C,\, T,\, F) \;\equiv\; (C \wedge T) \vee (\neg(C) \wedge F) \tag{6}$$
$$\textbf{CNF:}\ \mathit{ite}(C,\, T,\, F) \;\equiv\; (\neg(C) \vee T) \wedge (C \vee F). \tag{7}$$

3 Partial Boolean Functions

This section introduces *partial boolean functions*, which play a central role in the interpretation of QBF derivations. For this discussion, let $\mathbf{V}$ be a fixed nonempty set of k propositional variables: $\mathbf{V} = \{q_i \mid 1 \le i \le k\}$.

Recall that the mathematical definition of a boolean function F on $\mathbf{V}$ is a $(k+1)$-ary relation (i.e., set of rows) in which each *total* assignment τ to the variables of $\mathbf{V}$ appears in the first k columns of exactly one row of F, and

q_1	q_2	q_3	F_1
1	0	0	1
1	0	1	1

q_1	q_2	q_3	G_1
0	0	0	0
1	0	0	0

q_1	q_2	q_3	G_2
0	0	0	0
1	0	1	1

Figure 1: Pbfs discussed in Example 14.

$F(\tau)$ appears in column $(k+1)$, called the ***output column***. This relation has 2^k rows and is often called a *truth table*. Although this seems like a cumbersome way to represent a function it is the natural lead-in to partial functions. For example, set operations like union and intersection are not useful on total functions because the result is not a function except in trivial cases. However, these operations prove to be very useful on partial functions.

Definition 12 A ***partial boolean function*** (**pbf**) on $\mathbf{V}$ is a (not necessarily proper) subset of the rows of some total boolean function on $\mathbf{V}$. The ***empty pbf*** has no rows and is denoted by $\diamondsuit$. With some abuse of notation we also write $\diamondsuit$ for the "value" of a pbf at total assignments that do not correspond to any row in the pbf.

We use the *ite* abbreviations $\mathit{if}(C, T) = \mathit{ite}(C, T, \diamondsuit)$ and $\mathit{cp}(T) = \mathit{if}(1, T)$.

Two pbfs F and G are said to be ***consistent*** if the union of their rows is a pbf; that is, there is no total assignment τ such that $F(\tau) = 0$ and $G(\tau) = 1$, or *vice versa*. Consistency extends to sets of pbfs in the natural way. □

Definition 13 ***Restriction*** on a pbf (or total boolean function) f defined over a set of variables $\mathbf{V}$ is denoted as $f\lceil_\sigma$ and can be defined as an operation on the relation that defines f, as follows: In general σ is a partial assignment to $\mathbf{V}$, represented as an ordered set of *literals*, $p_1, \ldots, p_m$. It suffices to describe the operation for $m = 1$, as the complete restriction is the same as applying it one variable at a time. So, let $\sigma = (p)$, a single literal with $|p| \in \mathbf{V}$ and let column j in the relational table for f represent $|p|$. **(1)** If p is the positive literal of $|p|$, let f_0 be the selection of all rows of f in which column j is 1; if p is the negative literal select the rows of f that have 0 in column j. **(2)** Let f_1 be a copy of f_0 except that all the entries in column j are inverted. **(3)** Then $f\lceil_p = (f_0 \cup f_1)$. Actually, the union is disjoint and this relation is called the *cylindrification* of f_0.

Thus, in our terminology, $f\lceil_\sigma$ is defined on the same set of variables as f, although its value is independent of the values of variables assigned by σ. □

Example 14 Let $\mathbf{V} = \{q_1, q_2, q_3\}$; Let $F = (q_1 \wedge \neg(q_2))$ and $G = (q_2 \vee q_3)$ be total boolean functions. The relational tables for three pbfs are shown in Figure 1, where F_1 is a pbf of F and G_1 and G_2 are pbfs of G. Note that F_1 and G_1 ***are not consistent***, whereas F_1 and G_2 are consistent. Of course, G_1 and G_2 must be consistent, because they are subsets of the same total function. □

Example 15 Let $\mathbf{V} = \{q_1, q_2, q_3\}$. Let $F = (q_1 \wedge \neg(q_2))$ and let $G = (q_2 \vee q_3)$, and let pbfs F_1, G_1, and G_2 be as discussed in Example 14 and shown in Figure 1. Then $(F_1)\lceil_{\overline{q_1}} = \diamondsuit$. Restrictions on q_1 are shown in Figure 2. The standard definition of restriction (Definition 13) gives $F\lceil_{q_1} = \neg(q_2)$ and $G\lceil_{q_1} = G$. □

Lemma 16 Let F be a pbf defined on $\mathbf{V}$ and let G be an extension of F to a total boolean function. Let σ be a partial assignment on $\mathbf{V}$. Then $G\lceil_\sigma$ is a total boolean function, and it is an extension of $F\lceil_\sigma$. ■

Definition 17 Boolean operators can be extended to accept partial boolean functions (pbfs) as parameters. The input pbfs should all be defined on the same set of propositional variables $\mathbf{V}$. The idea is that for every extension of the input pbfs to total boolean functions the output should be an extension of the output pbf on the same $\mathbf{V}$.

The main idea is that for a boolean operator whose parameters are pbfs instead of total functions, the output is defined for an assignment σ if and only if the input pbfs have sufficient known information at σ so that only one output value is possible no matter how the input pbfs are extended to total functions.

Most identities for unary and binary boolean operators are obvious, such as $\neg(\diamondsuit) \equiv \diamondsuit$, $(1 \wedge \diamondsuit) \equiv \diamondsuit$, $(0 \wedge \diamondsuit) \equiv 0$, $(1 \vee \diamondsuit) \equiv 1$, $(0 \vee \diamondsuit) \equiv \diamondsuit$, etc.

Some cases where the output of the ternary operator *ite* is defined may not be immediately obvious:
$\mathit{ite}(\diamondsuit, 0, 0) \equiv 0$, and
$\mathit{ite}(\diamondsuit, 1, 1) \equiv 1$.
See the technical appendix for detailed treatment.

The *restriction* operator can be applied to pbfs with the same rules as for total boolean functions; that is, if G is an extension of pbf F to a total boolean function, then $G\lceil_\sigma$ is an extension of $F\lceil_\sigma$. This can be shown by straightforward induction.

Other cases are easily evaluated from these examples. □

4 Syntax of LDP-Resolution

The term *long-distance resolution* was coined by Zhang and Malik [23] for resolution that accepted pairs of complementary universal literals in its resolvents. Balabanov and Jiang gave a precise condition for allowing long-distance resolution [2]. We call operations that satisfy their conditions *LD-resolution*. What they call *LDQ-resolution* consists of LD-resolution and universal reduction. This section introduces ***LDP-resolution***, an operation that integrates long-distance resolution with partial Herbrand functions.

Definition 18 A ***mixed variable*** in a clause is a universal variable with both positive and negative literals in the same clause. (We simply list both literals on the mixed variable.) □

Definition 19 This paper uses a form of ***nonclausal resolution***, which a generalization of propositional clause resolution due to Manna and Waldinger [14]. We use the following

q_1	q_2	q_3	$(F_1)\lceil_{q_1}$
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1

q_1	q_2	q_3	$(G_1)\lceil_{q_1}$
0	0	0	0
1	0	0	0

q_1	q_2	q_3	$(G_2)\lceil_{q_1}$
0	0	1	1
1	0	1	1

Figure 2: Restrictions on pbfs discussed in Example 15.

specific notation:

$$\mathsf{res}_e(F_0, F_1) \quad = \quad F_0\lceil_e \vee F_1\lceil_{\overline{e}} \qquad (8)$$

The clausal case has $\overline{e} \in F_0$ and $e \in F_1$. For this paper e and $\overline{e}$ are always existential literals, called the ***clashing literals***. Note that the second operand is always the clause that contains e.

The general case is only useful if the resolvent is not tautologously true, $F_0\lceil_e$ properly subsumes F_0 and $F_1\lceil_{\overline{e}}$ properly subsumes F_1; otherwise the resolvent is weaker than an operand.

A logically equivalent definition is:

$$\mathsf{res}_e(F_0, F_1) \quad = \quad \mathit{ite}(e, F_0, F_1) \qquad (9)$$

This is clear from Definition 11 and shows that the resolvent expressions do not actually depend on e. □

Definition 20 The ***LDP-resolution criteria*** apply when the operands are QBF clauses and the resolvent contains u and $\overline{u}$. Say the clashing variable was $|e|$. We say $|u|$ is a ***mixed variable***. Unlike most papers, we do not "merge" such complementary pairs, for reasons explained in Section 4. LDP permits mixed $|u|$ in a resolvent **if (A)** all u-literals came from a single resolution operand, **or if (B)** ($|e| \prec |u|$) holds, i.e., $|e|$ is outer to $|u|$ in the prenex. (see Definition 2). Mixed existential variables are never permitted. □

It is not useful to regard LDP resolvents (or LD resolvents) with mixed variables as logical formulas because they are tautolous; their value lies in using them as a *notation* in solver implementations [23].

5 Partial Herbrand Functions

The intuitive meaning of partial Herbrand functions (phfs) are at the core of LDP methodology. We explain this before proceeding to the technical definitions. We have an original closed PCNF $\Phi = \overrightarrow{Q}\,.\,\mathcal{F}$ and possibly some derived clauses. For clause $C \in \mathcal{F}$ and a universal literal $u \in C$, the A-player (trying to falsify *some* clause) thinks:

The value of literal u matters for C only if the assignment to existential literals outer to u (call it τ) falsifies ($\mathsf{exist}(C) \prec u$), and then the useful value is $u = 0$. (10)

This explains the base definitions

$$\mathit{phf}(u, C) \quad = \quad \begin{cases} 0 & \text{if all literals in } (\mathsf{exist}(C) \prec u) \\ & \text{are assigned } 0 \\ \diamondsuit & \text{otherwise.} \end{cases} \qquad (11)$$

If u is a negative literal, i.e., $\overline{|u|}$; then $\mathit{phf}(u, C)$ tells when variable $|u|$ should be 1.

The "intended meaning" of $\mathit{phf}(u, C)$ in (10) is generalized in a natural way by the phfs of derived clauses, but now with the goal to detect whether the value of u matters to the A-player anywhere in the proof of a derived clause D. For this purpose we may restrict attention to the support of D, which we denote as $\mathcal{G}$. Let M be any model tree for $\Psi = \overrightarrow{Q}\,.\,\mathcal{G}$. The analogous goal of the A-player is to show that some branch of M, call it τ, falsifies D. If such τ exists in M, it shows D is *not* a logical consequence of Ψ.

Two kinds of partial Herbrand functions (phfs) are defined for LDP-resolution. The definitions depend implicitly on the original formula Φ. To avoid excessive verbiage we use "phf" to refer to either $\mathit{phf}()$ or $\mathit{phfm}()$. The name *phf* indicates that the first parameter is a universal *literal* and the output is 0 or $\diamondsuit$.

The name *phfm* indicates that the first parameter is a universal *variable* (v in (12) below), and the output is 0, 1, or $\diamondsuit$. Although phfms are not part of any LDP derivation, they facilitate *verifying* that a derivation of the empty clause is correct, which is considered important in some circles.

In all cases *phfm* is defined naturally in terms of *phf* as follows:

$$\mathit{phfm}(v, D) \quad = \quad (\neg\mathit{phf}(\overline{v}, D) \cup \mathit{phf}(v, D))\,. \qquad (12)$$

Recall that "$\cup$" is not the same as "or" for pbfs. It is easily shown that "$\cup$" is consistent in Eq. (12) in all the cases of Definition 21 where it is used, below. Note that $\mathit{phfm}(v, D)$ is *not* defined recursively.

Phfs for derived clauses are defined recursively in terms of the operand(s) of the proof operation that derived the clause. For any form of universal reduction on v, the phf is copied from the operand if $|v| \neq |u|$, or is $\diamondsuit$ if $|u|$ itself is being reduced.

The general formula for phfs of resolvents, where $D = \mathsf{res}_e(D_0, D_1)$, is:

$$\mathit{phf}(u, D) \quad = \quad \mathit{ite}(e, \mathit{phf}(u, D_0), \mathit{phf}(u, D_1)) \qquad (13)$$

where u may be a postive or negative literal. Then compute $\mathit{phfm}(|u|, D)$ with Eq. (12). Some of the phfs in an *ite* may be $\diamondsuit$.

An important identity for *all* resolution cases is:

$$\mathit{phf}(u, D) \quad \equiv \quad \mathit{phf}(u, D_0)\lceil_e \vee \mathit{phf}(u, D_1)\lceil_{\overline{e}}. \qquad (14)$$

This shows that $\mathit{phf}(u, D)$ does not actually depend on the clashing literal e, although the *ite* operands in Eq. 13 do. Therefore it does not matter if $u \prec e$; the phf only depends on existential variables outer to u, as required for Herbrand functions.

It is important to remember that in the examples a pair of literals $u, \overline{u}$ is shorthand for the pbf ($\mathit{phf}(u, D) \cup$

phf$(\overline{u},\, D)$), which must be consistent because the only boolean value taken by phfs is 0. In implementations, a pair of pointers to the phfs probably suffices.

Resolution simplifies in several special cases, detailed below. The complicated cases are when u is mixed in D and/or when D was derived by LDS-resolution with a clashing variable e such that $u \prec e$ but *indep*$(u,\, e)$ holds.

Definition 21 Let $D = \mathsf{res}_e(D_0,\, D_1)$. For each universal variable $|u| \in$ *vars*(D):

1. If $|e|$ depends on $|u|$ (i.e., $|u| \prec |e|$ and *indep*$(|u|,\, |e|)$ does *not* hold), and literals on $|u|$ occur in only one operand, then the phfs in the operands cannot depend on e. Copy the phfs of the operand that contains literals on $|u|$. This includes the case that $|u|$ is mixed. One of the phfs is $\diamondsuit$ unless $|u|$ is mixed. The *phfm* may use Eq. (12), but it will be the same as in the operand that had the literals on $|u|$.
2. If $|u| \prec |e|$ and u occurs in both operands, then $|u|$ cannot be mixed, and
$$phf(u,\, D) \quad = \quad (phf(u,\, D_0) \cap phf(u,\, D_1)) \quad (15)$$
because $0 \vee \diamondsuit = \diamondsuit$. Then compute *phfm*$(|u|,\, D)$ with Eq. (12).
3. If $|e| \prec u$ and u occurs in only one operand, then copy the phfs and phfm of that operand. This includes the case that $|u|$ is mixed.
4. If $|e| \prec u$ and u occurs in both operands, and u is *not* mixed in D, then *phf*$(u,\, D)$ is given by Eq. 13. One of the phfs in the *ite* may be $\diamondsuit$. Then compute *phfm*$(|u|,\, D)$ with Eq. (12).
5. If $|e| \prec |u|$ and $|u|$ *is* mixed in D, then *phf*$(u,\, D)$ is given by Eq. 13. Then compute *phfm*$(|u|,\, D)$ with Eq. (12). Some of the phfs in an *ite* may be $\diamondsuit$.

□

6 Logical Implications in QBF Derivations

This section considers the logical properties of derivations from closed PCNFs. The main result of this section is Theorem 24, which states that LDP-resolution derives only logical consequences; it follows from the more technical Lemma 23. We begin with a helpful definition.

Definition 22 Let Π_D be an LDP-resolution derivation of clause D from Φ. Let τ be a prenex-ordered assignment for Φ. Then τ is said to ***effectively falsify*** D if: **(1)** For each literal $p \in D$ that is *not* mixed, $\tau(p) = 0$ (p may be universal or existential). **(2)** For each *mixed* universal variable $v \in$ *vars*(D), define $\sigma(v)$ to be the subset of τ that assigns values to *existential* variables outer to v. Then *phf*$(v,\, D)\lceil_{\sigma(v)} = \diamondsuit$ or $\tau(v) =$ *phfm*$(v,\, D)\lceil_{\sigma(v)}$.

That is, τ assigns the variable v the value obtained by evaluating *phfm*$(v,\, D)$ at the point $(\mathsf{exist}(\tau) \prec v)$ if that value is defined. If *phfm*$(v,\, D)\lceil_{\sigma(v)} = \diamondsuit$, then either value for $\tau(v)$ allows τ to effectively falsify D. □

Lemma 23 Let clause D be derived by LDP-resolution (see Definition 4) from an original QBF $\Phi = \overrightarrow{Q}.\mathcal{F}$. Let S_D be the *support* of D in Π_D, that is, the clauses used in its derivation. Let T be an E-assignment tree for Φ such that some branch $\tau \in T$ effectively falsifies D. Then there is some branch $\tau_f \in T$ that falsifies some original clause $C_j \in S_D$.

Proof: (Sketch) Adopt the method of Goultiaeva *et al.* [9] for extracting certificates (see also [21]). ■

Theorem 24 Any non-tautologous clause D derived by LDP-resolution from PCNF $\Phi = \overrightarrow{Q}.\mathcal{F}$ is logically implied by Φ.

Proof: (Sketch) Contrapositive of Lemma 23. ■

Corollary 25 If PCNF Φ has an LDP-resolution refutation, then $\Phi =$ *false*. ■

7 Conclusion

The proof system LDP-resolution was introduced and shown to derive only logical consequences, when the derived clause has no mixed phfs. It uses a novel version of long-distance resolution with partial Herbrand functions to avoid tautologous universal literals. Certain aggressive dependency schemes are shown in Appendix B to derive clauses that are *not* logical consequences in some cases.

Future work should develop more integration of QBF reasoning with applications. More than a one-bit result is needed in practice. Incremental formulas have been productive in the propositional domain.

Partial Herbrand functions and partial Skolem functions might be useful other fragments of first-order logic. Quantified constraint satisfaction problems in which variables have finite domains are one possibility. Modeling procedurally defined functions with preconditions is another possibility, such as `first` and `rest` in Common Lisp, which require a non-empty list for their parameter.

A Logical Operators on Pbfs

Boolean operators can be extended to accept partial boolean functions (pbfs) as parameters. The input pbfs should all be defined on the same set of propositional variables $\mathbf{V}$. The idea is that for every extension of the input pbfs to total boolean functions the output should be an extension of the output pbf on the same $\mathbf{V}$.

Definition 26 Let $\mathbf{V}$ be a fixed nonempty set of k propositional variables. Let *op* be an m-ary boolean operator (m may be 0) with parameters P_i, $1 \leq i \leq m$, where the P_i are pbfs on $\mathbf{V}$. Then $\mathit{op}(P_1, \ldots, P_m)$ is the pbf P on $\mathbf{V}$ defined as follows:

For each total assignment σ on $\mathbf{V}$ there is a $b \in \{0, 1\}$ such that the following two statements are equivalent:

1. For every extension of P_i to a total boolean function on $\mathbf{V}$, call it F_i,

$$\mathit{op}(F_1(\sigma), \ldots, F_m(\sigma)) \quad = \quad b.$$

2. P contains exactly one row of the form (σ, b).

In other words, $P(\sigma)$ is defined if any only if the input pbfs have sufficient known information at σ to determine the output of *op* at σ. □

It suffices to consider the cases that one or more operands are the empty pbf $\diamondsuit$. Also we adopt the convention that an output value of $\diamondsuit$ means there is no matching row in the output pbf. The following relationships are straightforward applications of Definition 26:

$$\mathit{cp}(\diamondsuit) \quad \equiv \quad \diamondsuit \tag{16}$$

$$\neg(\diamondsuit) \quad \equiv \quad \diamondsuit \tag{17}$$

$$\bigwedge_{1 \leq i \leq m} (P_i(\sigma)) \quad = \quad \begin{cases} 0 & \text{if some } P_i(\sigma) = 0; \\ 1 & \text{if every } P_i(\sigma) = 1; \\ \diamondsuit & \text{otherwise.} \end{cases} \tag{18}$$

$$\bigvee_{1 \leq i \leq m} (P_i(\sigma)) \quad = \quad \begin{cases} 1 & \text{if at least one } P_i(\sigma) = 1; \\ 0 & \text{if every } P_i(\sigma) = 0; \\ \diamondsuit & \text{otherwise.} \end{cases} \tag{19}$$

Also, $\mathit{xor}(\diamondsuit, P) \equiv \diamondsuit$ for all pbfs P.

The extension of *ite* and other operators on three or more parameters requires care. By application of Definition 26:

$$\mathit{ite}(\diamondsuit, 0, 0) \quad = \quad 0 \tag{20}$$

$$\mathit{ite}(\diamondsuit, 1, 1) \quad = \quad 1 \tag{21}$$

However, use of the DNF in (6) fails to give (21), and use of the CNF in (7) fails to give (20). The solution is to close a CNF that represents the operator on total boolean functions under resolution (see Definition 27 below). The next theorem shows that the resulting CNF correctly represents the corresponding operator on pbfs. Closure of a DNF under consensus (also called *term resolution*) accomplishes the same thing by a similar argument.

Definition 27 A propositional CNF $\mathcal{F} = \{C_j\}$ is said to be ***closed under resolution*** if for every pair of resolvable clauses $C_i \in \mathcal{F}$ and $C_j \in \mathcal{F}$, The resolvent is either tautologous or is subsumed by a clause already in $\mathcal{F}$. A propositional DNF $\mathcal{G} = \bigwedge_j (T_j)$ is said to be ***closed under consensus*** if for every pair of resolvable terms $T_i \in \mathcal{G}$ and $T_j \in \mathcal{G}$, The consensus is either contradictory or is subsumed by a term already in $\mathcal{G}$. □

Theorem 28 Let $\mathbf{V}$ be a fixed nonempty set of k propositional variables. Let the boolean operator $\mathit{op}(P_1, \ldots, P_m)$ be defined or represented for total boolean functions on $\mathbf{V}$ as parameters by a propositional CNF $\mathcal{F}$ that treats the P_i as boolean variables and has no other variables. We abbreviate $\{P_i \mid 1 \leq i \leq m\}$ to $\{P_i\}$. Then $\mathcal{F}$ represents the corresponding operator for pbfs on $\mathbf{V}$ as parameters (as defined in Definition 26) if and only if $\mathcal{F}$ is closed under resolution.

Proof: ($\Rightarrow$) Assume for purposes of contradiction that $\mathcal{F}$ is not closed under resolution. There is some non-tautologous clause $D = \mathsf{res}_p(C_i, C_j)$ such that $C_i \in \mathcal{F}$, $C_j \in \mathcal{F}$, and D is not subsumed by any clause in $\mathcal{F}$. Consider the partial assignment on $\{P_i\}$ defined by $\sigma = \neg(D)$. Treat σ as a pbf on $\{P_i\}$ (i.e., every variable not assigned by σ is $\diamondsuit$). Every extension of σ to a total assignment τ on $\{P_i\}$ causes $\mathcal{F}\lceil_\tau = 0$ (because either $C_i\lceil_\tau = 0$ or $C_j\lceil_\tau = 0$), so $\mathcal{F}(\sigma)$ should be 0. However, every clause $C \in \mathcal{F}$ contains a literal not mentioned in σ (or D would be subsumed), so $C(\sigma) = \diamondsuit$ or $C(\sigma) = 1$ by (19). In particular, $C_i(\sigma) = \diamondsuit$ and $C_j(\sigma) = \diamondsuit$, so $\mathcal{F}(\sigma) = \diamondsuit$ by (18). This contradicts the hypothesis that $\mathcal{F}$ represents *op* for pbfs.

($\Leftarrow$) By hypothesis, $\mathcal{F}$ is closed under resolution. Let σ be a partial assignment on $\{P_i\}$ and let τ range over extensions of σ to total assignments on $\{P_i\}$. If there is some $C_j \in \mathcal{F}$ such that $C_j\lceil_\sigma \neq 1$, then there is some τ such that $C_j\lceil_\tau = 0$. Therefore, if $\mathit{op}(\tau) = 1$ for all τ, then $\mathit{op}(\sigma) = 1$.

Now assume $\mathit{op}(\tau) = 0$ for all τ. It remains to show that $\mathit{op}(\sigma) = 0$. This is immediate if $C_j\lceil_\sigma = 0$ for any $C_j \in \mathcal{F}$, so assume this is not the case. Define:

$$\mathcal{G} \quad = \quad \{C_j \mid C_j \in \mathcal{F} \text{ and } C_j\lceil_\sigma = \diamondsuit\}. \tag{22}$$

If $\mathcal{G}$ has no clauses, $\mathit{op}(\sigma)$ must be 1, contradicting the hypothesis that all $\mathit{op}(\tau) = 0$, so assume $\mathcal{G}$ has some clauses. In this case $\mathcal{G}\lceil_\sigma$ must be unsatisfiable or else there would be some total extension τ that satisfies $\mathcal{G}$ and also satisifies $\mathcal{F}$. Let Π be a resolution refutation of $\mathcal{G}\lceil_\sigma$. Removing the restriction converts Π to a resolution derivation on $\mathcal{G}$ of some clause $D \subseteq \neg(\sigma)$. But D is also derivable from $\mathcal{F}$, which is closed under resolution. Therefore some $C_j \in F$ subsumes D, and $C_j\lceil_\sigma = 0$, so $\mathit{op}(\sigma) = 0$. ■

Corollary 29 Let $\mathbf{V}$ be a fixed nonempty set of k propositional variables. Let the boolean operator $\mathit{op}(P_1, \ldots, P_m)$ be defined or represented for total boolean functions on $\mathbf{V}$ as parameters by a propositional DNF $\mathcal{G}$ that treats the P_i as boolean variables and has no other variables. We abbreviate $\{P_i \mid 1 \leq i \leq m\}$ to $\{P_i\}$. Then $\mathcal{G}$ represents the correponding operator for pbfs on $\mathbf{V}$ as parameters (as defined in Definition 26) if and only if $\mathcal{G}$ is closed under consensus (also called term resolution).

Proof: Define $\mathcal{F} = \neg(\mathcal{G})$ and apply Theorem 28 to $\mathcal{F}$. ■

Corollary 30 Let $\mathbf{V}$ be a fixed nonempty set of k propositional variables. Let C, T, F be pbfs on $\mathbf{V}$. Then

$$\mathit{ite}(C, T, F) \quad \equiv \quad [\neg(C), T] \wedge [C, F] \wedge [T, F] \tag{23}$$

$$\mathit{ite}(C, T, F) \quad \equiv \quad (C \wedge T) \vee (\neg(C) \wedge F) \vee (T \wedge F) \tag{24}$$

$$\mathit{ite}(\diamondsuit, T, F) \quad \equiv \quad (T \cap F) \tag{25}$$

Note that the rightmost clause in (23) and the rightmost term in (24) have no effect when C is a total boolean function. ■

B Derivations That Are Not Logical Consequences

Slivovsky and Szeider [20] define the ***reflexive resolution-path*** dependency scheme (denoted D^{rrs} in that paper). A universal literal u is called *rrs-tailing* if no existential literal in the same clause depends on it in the *rrs* scheme. Such universal literals are deleted from that clause. That paper shows that Q-resolution with universal reductions based on rrs-tailing is refutationally sound. The rrs scheme is strictly more aggressive than the *standard dependency scheme* (D^{std}) that was introduced and studied theoretically in the pioneering work of Samer and Szeider [17] so refutational soundness of Q-resolution with universal reductions based on std-tailing is a corollary.[1]

More aggressive dependency schemes such as Samer's generalized triangle dependencies [16], *strict* standard dependencies [22], and reflexive resolution-path dependencies [20] may derive clauses that are not logically implied by Φ. A counter-example with four clauses shows that universal reduction can delete some model trees.

Example 31 Let $\Phi = \overrightarrow{Q}\,.\,\mathcal{F}$ be given in chart form:

Φ	$\exists b$	$\exists c$	$\forall u$	$\exists f$	$\exists g$
C_1		c	u	f	
C_2	b		$\overline{u}$		g
C_3	$\overline{b}$			$\overline{f}$	
C_4		$\overline{c}$			$\overline{g}$

M_1 :

$\overline{b}$	$\overline{c}$	$\overline{u}$	f	g
		u	$\overline{f}$	$\overline{g}$

M_2 :

$\overline{b}$	$\overline{c}$	$\overline{u}$	f	$\overline{g}$
		u	$\overline{f}$	g

Two of several model trees are M_1 and M_2 as shown above. Clauses $D_1 = [\overline{b},\, c]$ and $D_2 = [b,\, \overline{c}]$ are derivable by Q-resolution, so they are logically implied by Φ. Under the standard dependency scheme f and g depend on u, so the only independence is that implied by the prenex order.

In the more aggressive dependency schemes mentioned above, f and g are independent of u. Under these schemes $D_3 = [b,\, g]$ and $D_4 = [c,\, f]$ are derivable by universal reduction from C_2 and C_1, respectively. Adding D_3 and D_4 to Φ is *safe* (its truth-value does not change); M_1 is still a model tree. However, M_2 is *not* a model tree for either $\Phi + D_3$ or $\Phi + D_4$,[2] so neither is a logical consequence. □

References

[1] Balabanov, V., Jiang, J.H., Janota, M., Widl, M.: Efficient extraction of QBF (counter)models from long-distance resolution proofs. In: Proc. AAAI. pp. 3694–3701 (2015)

[2] Balabanov, V., Jiang, J.R.: Unified QBF certification and its applications. Formal Methods in System Design 41, 45–65 (2012)

[3] Beyersdorff, O., Blinkhorn, J.: Dependency schemes in QBF calculi: Semantics and soundness. In: Proc. CP, LNCS 9892. pp. 96–112. Springer (2016)

[4] Bjorner, N., Janota, M., Klieber, W.: On conflicts and strategies in QBF. In: LPAR-20 (2015)

[5] Blinkhorn, J., Beyersdorff, O.: Shortening QBF proofs with dependency schemes. In: Proc. SAT, LNCS 10491. pp. 263–280. Springer (2017)

[6] Bryant, R.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (Aug 1986)

[7] Bubeck, U., Kleine Büning, H.: Nested boolean functions as models for quantified boolean formulas. In: Proc. SAT 2013, LNCS 7962, pp. 267–275. Springer (2013)

[8] Egly, U., Lonsing, F., Widl, M.: Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In: LPAR, LNCS 8312. pp. 291–308 (2013)

[9] Goultiaeva, A., Van Gelder, A., Bacchus, F.: A uniform approach for generating proofs and strategies for both true and false QBF formulas. In: Proc. IJCAI (2011)

[10] Janota, M., Chew, L., Beyersdorff, O.: On unification of QBF resolution-based calculi. In: Proc. MFCS, LNCS 8635. pp. 81–93 (2014)

[11] Klieber, W., Janota, M., Marques-Silva, J., Clarke, E.: Solving QBF with free variables. In: Proc. CP, LNCS 8124. pp. 415–431. Springer (2013)

[KBKF95] H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117:12–18, 1995.

[13] Lonsing, F., Biere, A.: Integrating dependency schemes in search-based QBF solvers. In: SAT. pp. 158–171. Springer (2010)

[14] Manna, Z., Waldinger, R.: A deductive approach to program synthesis. TOPLAS 2, 90–121 (1980)

[15] Peitl, T., Slivovsky, F., Szeider, S.: Long distance q-resolution with dependency schemes. In: Proc. SAT (2016)

[16] Samer, M.: Variable dependencies of quantified CSPs. In: LPAR, LNCS 5330. pp. 512–527 (2008)

[17] Samer, M., Szeider, S.: Backdoor sets of quantified boolean formulas. JAR 42, 77–97 (2009)

[18] Samulowitz, H., Davies, J., Bacchus, F.: Preprocessing QBF. In: Proc. CP 2006 (LNCS 4204). pp. 514–529 (2006)

[19] Samulowitz, H., Bacchus, F.: Dynamically partitioning for solving QBF. In: Theory and Applications of Satisfiability Testing (SAT). pp. 215–229 (2007)

[20] Slivovsky, F., Szeider, S.: Soundness of Q-resolution with dependency schemes. Th. Comp. Sci. 612, 83–101 (2016)

[1]The expression "D_1 is *more aggressive* than D_2" means that the *independence* found by D_1 is a superset of that found by D_2.

[2]$\Phi + D$ abbreviates $\overrightarrow{Q}\,.\,(\mathcal{F} \cup D)$.

[21] Van Gelder, A.: Input distance and lower bounds for propositional resolution proof length. In: Theory and Applications of Satisfiability Testing (SAT) (2005)

[22] Van Gelder, A.: Variable independence and resolution paths for quantified boolean formulas. In: Proc. CP (LNCS 6876). pp. 789–803. Springer, Perugia, Italy (2011)

[23] Zhang, L., Malik, S.: Conflict driven learning in a quantified boolean satisfiability solver. In: Proc. ICCAD. pp. 442–449 (2002)