Moving Definition Variables
in Quantified Boolean Formulas

Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant

Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
{jereeves,mheule,randy.bryant}@cs.cmu.edu

Abstract. Augmenting problem variables in a quantified Boolean formula with definition variables enables a compact representation in clausal form. Generally these definition variables are placed in the innermost quantifier level. To restore some structural information, we introduce a preprocessing technique that moves definition variables to the quantifier level closest to the variables that define them. We express the movement in the QRAT proof system to allow verification by independent proof checkers. We evaluated definition variable movement on the QBFEVAL'20 competition benchmarks. Movement significantly improved performance for the competition's top solvers. Combining variable movement with the preprocessor BLOQQER improves solver performance compared to using BLOQQER alone.

1 Introduction

Boolean formulas or circuits can be translated into conjunctive normal form (CNF) by introducing *definition variables* to augment the existing *problem variables*. Definition variables are introduced through a set of *defining clauses*, given by the Tseitin transformation [19] or the Plaisted-Greenbaum transformation [16]. The variables occurring in the defining clauses are the *defining variables*, and they determine the value of the definition variable. In CNF, definitions are not an explicit part of the problem representation, preventing solvers from using this structural information. Quantified Boolean formulas (QBF) extend CNF into prenex conjunctive normal form (PCNF) with the addition of quantifier levels. In practice, definition variables are usually placed in the innermost quantifier level. However, placing a definition variable in the quantifier level immediately following its defining variables can benefit solvers.

We propose a preprocessing technique for moving definition variables to the quantifier level of their innermost defining variables. As a starting point, existing tools KISSAT and CNFTOOLS can detect candidate definitions in a CNF formula. We move the definition variables through a series of clause additions and deletions following the QRAT proof system. This effectively replaces the old definition variable with a new variable in the formula, where the new variable is placed in the desired quantifier level.

In Boolean satisfiability (SAT), most solvers generate proofs of unsatisfiability for independent checking [7, 9, 20]. This has proved valuable for verifying solutions independent of a (potentially buggy) solver. Proof generation is difficult for QBF and relatively uncommon in solvers. The QBF preprocessor BLOQQER [2] generates QRAT proofs [8] for all of the transformations it performs. We also generate QRAT proofs

for variable movement. This allows verification with the independent proof checker QRAT-TRIM, ensuring the movement preserves equivalence with the original formula.

Clausal-based QBF solvers rely on preprocessing to improve performance. Almost every top-tier solver in the QBFEVAL'20 competition[1] used some combination of BLO-QQER, HQSPRE [21], or QBFRELAY [15]. Some solvers incorporate preprocessing techniques into the solving phase, e.g., DEPQBF's [14] use of dynamic quantified blocked clause elimination. Unlike other preprocessing techniques, variable movement does not add or remove clauses or literals. However, it can prompt the removal of literals through universal reduction and may guide solver decisions in a beneficial way.

The contributions of this work include: (1) adapting the SAT solver KISSAT and CNF preprocessor CNFTOOLS to detect definitions in a QBF for variable movement, (2) giving an algorithm for moving variables that maximizes variable movement, and (3) formulating steps for generating a QRAT proof of variable movement. In the evaluation, variable movement is shown to significantly improve the performance of top solvers from the QBFEVAL'20 competition. Moving variables before applying BLO-QQER further improves solver performance.

2 Preliminaries

2.1 Quantified Boolean Formulas

Quantified Boolean formulas (QBF) can be represented in prenex conjunctive normal form (PCNF) as $\Pi.\psi$, where Π is a prefix of the form $Q_1 X_1 Q_2 X_2 \cdots Q_n X_n$ for $Q_i \in \{\forall, \exists\}$ and the matrix ψ is a CNF formula. The formula ψ is a conjunction of clauses, where each clause is a disjunction of literals. A literal l is either a variable $l = x$ or negated variable $l = \overline{x}$, and $Var(l) = x$. The formula $\psi(l)$ is the clauses $\{C \mid C \in \psi, l \in C\}$. The set of all variables occurring in a formula is given by $Var(\psi)$. Substituting a variable y for x in ψ, denoted as $\psi[y/x]$, will replace every instance of x with y and $\overline{x}$ with $\overline{y}$ in the formula. The sets of variables X_i are disjoint, and we assume every variable occurring in ψ is in some X_i. A variable x is *fresh* if it does not occur in $\Pi.\psi$. The quantifier for literal l with $Var(l) \in X_i$ is $\mathcal{Q}(\Pi, l) = Q_i$, and l is said to be in *quantifier level* $\lambda(l) = i$. If $\mathcal{Q}(\Pi, l) = Q_i$ and $\mathcal{Q}(\Pi, k) = Q_j$, then $l \leq_\Pi k$ if $i \leq j$. $Q_1 X_1$ is referred to as the outermost quantifier level and $Q_n X_n$ is the innermost quantifier level.

2.2 Inference Techniques in QBF

Given a clause C, if a literal $l \in C$ is universally quantified and for all existentially quantified literals $k \in C$, $k <_\Pi l$, then l can be removed from C. This process is called *universal reduction* (UR). Given two clauses C and D with $x \in C$ and $\overline{x} \in D$, the *Q-resolvent* over *pivot* variable x is $\mathrm{UR}(C) \cup \mathrm{UR}(D) \setminus \{x, \overline{x}\}$ [12]. The operation is undefined if the result is tautological. This extends *resolution* for propositional logic by applying UR to the clauses before combining them, while disallowing tautologies. Adding or removing non-tautological Q-resolvents preserves logical equivalence.

---

[1] available at `http://www.qbflib.org/qbfeval20.php`

Given a prefix Π and clauses C and D with $l \in C$ and $\bar{l} \in D$, the *outer resolvent* over existentially quantified *pivot* literal l is $C \cup \{k \mid k \in D, k \neq \bar{l}, k \leq_\Pi l\}$. Given a QBF $\Pi.\psi$, a clause C is *Q-blocked* on some existentially quantified literal $l \in C$ if for all $D \in \psi(\bar{l})$ the outer resolvent of C with D on l is a tautology. This extends the *blocked* property for CNF with the additional restriction on the conflicting literal's quantifier level.

A clause C *subsumes* D if $C \subseteq D$. The property *Q-blocked-subsumed* generalizes Q-blocked by requiring the outer resolvents be tautologies or subsumed by some clause in the formula.

Given a QBF $\Psi = \Pi.\psi$, if a clause C is Q-blocked-subsumed then C is QRAT w.r.t. Ψ. In this case, C can be added to ψ or deleted from ψ if $C \in \psi$ while preserving equivalence. A series of clause additions and deletions resulting in the empty formula is a *satisfaction proof* for a QBF if all clause deletions are QRAT. A series of clause additions and deletions deriving the empty clause is a *refutation proof* for a QBF if all clause additions are QRAT. In a *dual proof* both clause additions and deletions are QRAT, so each step preserves equivalence regardless of the truth value of the QBF. The QBF Ψ' that results from applying the dual proof steps to Ψ is equivalent to Ψ.

2.3 Definitions

A variable x is a definition variable in $\Psi = \Pi.\psi$ with defining clauses $\delta(x)$ containing x, $\delta(\bar{x})$ containing $\overline{x}$, and defining variables $Z_x = Var[\delta(x) \cup \delta(\overline{x})] \setminus \{x\}$ when two properties hold: (1) the definition is *left-total*, meaning that x has a value for every assignment of Z_x, and (2) the definition is *right-unique*, meaning that x has a unique value for every assignment of Z_x. The clauses $\delta(x) \cup \delta(\overline{x})$ are left-total iff they are Q-blocked on variable x. This implies that the definition variable comes after the defining variables w.r.t. Π. The definition is right-unique if the SAT problem $\{C' \mid C \in \delta(x) \cup \delta(\bar{x}), C' = C \setminus \{x, \overline{x}\}\}$ is unsatisfiable. We can assume that any right-unique variable is universally quantified, otherwise the formula would be trivially false.

The *remaining clauses* of x are $\rho(x) = \psi(x) \setminus \delta(x)$ and $\rho(\overline{x}) = \psi(\overline{x}) \setminus \delta(\bar{x})$. If x occurs as a single polarity in the remaining clauses, it can be encoded as a *one-sided* definition: if $\rho(x)$ is empty only $\delta(x)$ are needed to define x, similarly with $\rho(\overline{x})$ and $\delta(\overline{x})$. This is a stronger condition than *monotonicity* used for the general Plaisted-Greenbaum transformation [16].

Example 1. $x \leftrightarrow a \wedge b$ is written in CNF as $(x \vee a \vee b) \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee b)$. Given $\rho(x) = \{(x \vee c), (x \vee d \vee e)\}$ and $\rho(\overline{x}) = \{\}$, $x = a \wedge b$ can be written as a one-sided definition with clauses $(\overline{x} \vee a) \wedge (\overline{x} \vee b)$.

In some definitions including exclusive-or (XOR denoted by $\oplus$), multiple variables are left-total and right-unique. Determining the definition variable requires information about how definition variables are nested within the formula.

Q-resolution can be generalized to sets of clauses $\mathcal{C}$ and $\mathcal{D}$, denoted $\mathcal{C} \otimes_x \mathcal{D}$, by generating the non-tautological resolvents from clauses in $\mathcal{C}(x)$ and $\mathcal{D}(\overline{x})$ on pivot variable x pairwise. Given a definition variable x and defining variables $\{z_1, \ldots, z_n\}$, let x' be a fresh variable with $\theta_x = \delta(x)$ and $\theta_{\overline{x}'} = \delta(\overline{x})[x'/x]$. The procedure

defining variable elimination applies set-based Q-resolution in the following way: set $\theta_1 = \theta_x(z_1) \otimes_{z_1} \theta_{\overline{x}'}(\overline{z}_1) \wedge \theta_x(\overline{z}_1) \otimes_{z_1} \theta_{\overline{x}'}(z_1)$ and compute $\theta_2 = \theta_1(z_2) \otimes_{z_2} \theta_1(\overline{z}_2)$; continue the process until $\theta_n = \theta_{n-1}(z_n) \otimes_{z_n} \theta_{n-1}(\overline{z}_n)$. UR is not applied because x is in the innermost quantifier level with respect to its defining variables. The first step ensures all clauses in θ_1 will contain both x and x'. θ_n will either be $\{(x \vee \overline{x}')\}$ or empty. If $\theta_n = \{(x \vee \overline{x}')\}$, linearizing the sets of resolvents θ_i forms a Q-resolution derivation of $(x \vee \overline{x}')$. This is similar to Davis Putnam variable elimination [4].

3 Definition Detection

Given a QBF, we first detect definitions to determine which variables can be moved. Variable movement depends on the defining clauses, the definition variables, and the nesting of definition variables. At a minimum, definition detection must produce the defining clauses, and the rest can be inferred during movement.

Since the seminal work by Eén and Biere [5], bounded variable elimination (BVE) has been an essential preprocessing technique in SAT solving. The technique relies on definitions, so most SAT solvers incorporate some form of definition detection. The conflict-driven clause learning SAT solver KISSAT [1] extends the commonly used syntactic pattern matching with semantic definition detection. The detection is applied to variables independently. Alternatively, the preprocessor CNFTOOLS [10] performs hierarchical definition detection, capturing additional information about definition variable nesting and monotonic definitions.

These tools run on CNF formulas. Stripping the prefix from a QBF yields a CNF, but not all definitions in the CNF are valid w.r.t. the prefix. For example, some definitions will not be left-total because of the quantifier level requirements for the Q-blocked property. Such definitions can be avoided during variable movement, and there is no need to modify the tools in this respect.

3.1 Hierarchical Definition Detection in CNFTOOLS

The hierarchical definition detection in CNFTOOLS employs a breadth first search (BFS) to recurse through nested definitions in a formula. Root clauses are selected heuristically, then BFS begins on the variables occurring in those clauses. All unit clauses are selected as root clauses. The *max-var* root selection selects variables based on their naming. This exploits the natural tendency of naming definition variables after problem variables. The more involved *min-unblocked* root selection finds a minimally unblocked literal. The heuristic takes more time to calculate but does not rely on variable naming.

When a variable is encountered in the BFS, CNFTOOLS checks if the defining clauses are blocked. If so, the following detection methods are applied: pattern matching for BiEQ, AND, OR, and full patterns, monotonic checking, and semantic checking. BiEQ refers to an equivalence between two variables.

A definition is a full pattern if $\forall C \in \delta(x) \cup \delta(\overline{x}), |C| = n + 1$ where n is the number of defining variables and there are 2^n defining clauses. The full pattern includes some common encodings for XOR, XNOR, NOT, and Majority3, but is often avoided. Since

4

the detection follows the hierarchical nesting of definitions, definition variables can be distinguished from defining variables online.

CNFTOOLS detects monotonic definitions, and some are either fully-defined or one-sided. The advantage of hierarchical detection is the ability to detect these one-sided definitions. The one-sided property is a necessary restriction on monotonic definitions because in some situations, monotonic definition variables may occur both positively and negatively in the defining clauses of other definitions. These additional clauses can prevent variable movement w.r.t. the QRAT proof system.

Semantic checking involves a right uniqueness check on the SAT problem described in the preliminaries. As definitions are detected the defining clauses are removed from the formula for the following iterations. This does not affect fully defined definition detection, but it can produce problematic one-sided definitions. For example, a variable may occur both positively and negatively in the defining clauses of other definitions, and removing those clauses makes the variable one-sided. Similar to the monotonic case, the additional defining clauses can prevent movement w.r.t. the QRAT proof system.

3.2 Independent Definition Detection in KISSAT

KISSAT uses definition detection to find candidates for BVE. Starting with the 2021 SAT Competition, semantic definition detection was incorporated into the BVE in-processing stage [6]. The semantic check is called after the syntactic pattern matching for BiEQ, AND, OR, ITE, and XOR definitions. An internal SAT solver KITTEN with low overhead and limited capabilities performs a right-uniqueness check on the formula $\psi(x) \cup \psi(\overline{x})$ after removing all occurrences of x and $\overline{x}$. This formula includes $\rho(x)$ and $\rho(\bar{x})$ as the set of defining clauses are not known in advance. If the formula is unsatisfiable, the unsatisfiable core is extracted (potentially after reduction) and returned as the set of defining clauses.

Core extraction does not guarantee the defining clauses are blocked. Internally KISSAT generates resolvents over the defining clauses. We modify KISSAT to only detect semantic definition where zero resolvents are generated. We ignore built-in heuristics for selecting candidate variables and instead iterate over all variables.

No nesting information is gathered during definition detection in KISSAT. This makes it impossible to distinguish between defining variables and the definition variable for XORs. Further, if a variable is checked only once and the XOR was detected first additional definitions would be ignored. We modified KISSAT so that when an XOR or semantic definition is detected for a variable, the clauses from that definition are marked as inactive and the definition detection procedure is rerun. This will determine if a variable is defined elsewhere. The clauses are marked active when detection begins on the next variable.

4 Moving Variables

Given a set of definitions, we attempt to move as many definition variables as close to their defining variables as possible. To do this, we introduce empty existential quantifier levels, denoted with T_i, following each $Q_i X_i$ in the original prefix yielding

5

$Q_1 X_1 \exists T_1 Q_2 X_2 \exists T_2 \cdots Q_{n-1} X_{n-1} \exists T_{n-1} Q_n X_n$. There is no T_n because variables are not moved inwards. For each definition variable x that can be moved, a fresh variable x' is placed in the quantifier level T_m for $m = \max\{\lambda(z) \mid z \in Z_x\}$. That is, x' will be placed in the existential block that immediately follows the innermost defining variable. Finally, x will be removed from the prefix, and the new formula will be $\psi[x'/x]$.

Example 2. In the formula $\exists x_3 \forall x_1 \exists x_4 \forall x_2 \exists x_5.(x_5 \vee \overline{x}_4 \vee \overline{x}_3) \wedge (\overline{x}_5 \vee x_3) \wedge (\overline{x}_5 \vee x_4) \wedge (x_5 \vee x_1) \wedge (x_2 \vee x_5)$, the variable x_5 is defined as $x_5 \leftrightarrow x_3 \wedge x_4$, with defining variables $\{x_3, x_4\}$. A fresh variable x_5' is introduced to replace x_5. x_5' is placed in an existential quantifier level following the innermost defining variable x_4. Then, x_5' is substituted for x_5 in the formula giving $\exists x_3 \forall x_1 \exists x_4 \exists x_5' \forall x_2.(x_5' \vee \overline{x}_4 \vee \overline{x}_3) \wedge (\overline{x}_5' \vee x_3) \wedge (\overline{x}_5' \vee x_4) \wedge (x_5' \vee x_1) \wedge (x_2 \vee x_5')$. After moving x_5, x_2 can be removed from the clause $(x_2 \vee x_5')$ by universal reduction.

New variables are introduced for movement because QRAT steps either add or delete clauses and cannot affect the quantifier level of an existing variable. In the checker QRAT-TRIM, when definitions are added the new definition variables are placed in a quantifier level based on their defining variables. For a definition variable x, if the innermost defining variable $z \in X_i$ is existentially quantified ($Q_i = \exists$) the definition variable is placed in X_i, and if z is universally quantified ($Q_i = \forall$) the definition variable is placed in the existential level X_{i+1}, So, new definition variables are placed in the desired quantifier level. Because contiguous levels with the same quantifier can be combined, the introduction of T levels does not change the semantics.

4.1 Moving in Order

Moving an individual definition variable is simple, but the general algorithm must consider definition dependencies to be efficient and make various checks to be correct. The tools for definition detection run on CNF instances, so, some definitions may not be left-total when considering the prefix. These quantifier related problems can be checked during movement. This can occur if the definition variable is in a level outer to one of its defining variables. Also, some monotonic definitions may not satisfy the one-sided property. This is a more complicated check that occurs during proof generation. If proof generation fails, then a variable is not moved.

The variable movement algorithm starts at the outermost quantifier level and sweeps inwards, at each step moving all possible definition variables to the current level. A definition variable x can be moved if $x >_\Pi z$ for all $z \in Z_x$, and x is not universally quantified. It can be moved to T_m where $m = \max\{\lambda(z) \mid z \in Z_x\}$, and will be moved during iteration m of the algorithm. Once a definition variable has been moved, if it was a defining variable for some other definitions, those definitions are checked for movement. Since the iteration starts at the outermost level, it guarantees variables that can be moved within our framework are moved as far as possible. This requires a single pass, so moved definitions will not be revisited.

4.2 XOR Processing

It is not initially clear how to determine which are the defining variables for an XOR definition. If a variable is defined elsewhere and appears in an XOR, it must be a defin-

ing variable in the XOR. In addition, universal variables must be defining variables. However, a distinction cannot be made between the remaining variables without information about movement.

Example 3. Given the QBF, $\exists_1 x_1, x_2 \forall y_1 \exists_2 x_3 \forall y_2 \exists_3 x_4 \forall y_3 \exists_4 x_5 \forall y_4 \exists_5 x_6, x_7.(x_6 \leftrightarrow x_1 \wedge x_i) \wedge (x_3 \oplus x_4 \oplus x_5) \wedge (x_1 \oplus x_5 \oplus x_6) \wedge \ldots$, determining the definition variables for the XOR definitions will hinge on the movement of x_6. **Case 1**, Let $x_i = x_7$ in the AND definition, x_6 cannot be moved. Then, x_5 can be moved to $\exists_3$ as the definition variable of $(x_3 \oplus x_4 \oplus x_5)$. No other variables can be moved. **Case 2**, Let $x_i = x_2$ in the AND definitions, x_6 can be moved to $\exists_1$. Then, x_5 can be moved to $\exists_1$ as the definition variable of $(x_1 \oplus x_5 \oplus x_6)$. Next, x_4 can be moved to $\exists_2$ as the definition variable of $(x_3 \oplus x_4 \oplus x_5)$. The possible movement of x_6 will determine how the XOR definitions are moved. This information is not known until runtime, so the definition variable of an XOR cannot be determined before variable movement is performed.

As seen in the example, movement of definition variables can affect what variable in an XOR is eventually moved. The definition variable for an XOR must be determined online during the movement process. During initialization and propagation, we reset the definition variable as the innermost variable in the XOR. If that variable is defined elsewhere, it cannot be moved. Otherwise, we perform the same check as the general case to see if the definition variable can be moved. With XOR definitions, the algorithm is still deterministic and produces optimal movement, since all variables that can be moved are moved to their outermost level.

4.3 Proving Variable Movement

In this section we describe how to modify a formula through a series of QRAT clause additions and deletions to achieve variable movement. Moving a definition variable x in the formula $\Pi.\psi$ involves:

- Introducing a new definition variable x' to replace x.
- Deriving an equivalence between x' and x.
- Transforming the formula to ψ to $\psi[x'/x]$ with x removed from Π and x' placed in the existential quantifier level following its innermost defining variable.

The algorithm for moving a definition variable x proceeds in five steps, each involving some clause additions or deletions. Some of the steps can be simplified depending on the type of definition. Moving a one-sided definition requires slight modifications to a few steps, and these are discussed following each of the relevant steps.

1. *Add the defining clauses $\delta(x')$ and $\delta(\overline{x}')$.*
   We introduce a fresh existential variable x' and add the defining clauses $\delta(x)[x'/x]$ and $\delta(\bar{x})[x'/x]$. Each clause is Q-blocked on x' or $\overline{x}'$ since the definition is left-total and variable x' is in the quantifier level following its innermost defining variable.
2. *Add the equivalence clauses $x \leftrightarrow x'$.*
   Both x and x' are fully defined by the same set of variables, so it is possible to derive the equivalence clauses $(\overline{x} \vee x')$ and $(x \vee \overline{x}')$. The first implication added is

Q-blocked-subsumed. Take $(\overline{x} \vee x')$, for each clause $C' \in \delta(\overline{x}')$, the outer resolvent of C' with $(\overline{x} \vee x')$ on x is subsumed by the corresponding $C \in \delta(\overline{x})$. This is not the case for $(x \vee \overline{x}')$ because the outer resolvent of $(x \vee \overline{x}')$ with $(\overline{x} \vee x')$ is not subsumed by the formula. The clause $(x \vee \overline{x}')$ is QRAT for certain definitions, in particular AND/OR. In the general case we generate a chain of Q-resolutions that imply $(x \vee \overline{x}')$. We use defining variable elimination to eliminate Z_x from the formula $\delta(x) \cup \delta(\overline{x}')$. The procedure produces the clause $(x \vee \overline{x}')$. The resolution tree rooted at $(x \vee \overline{x}')$ is traversed in post-order giving the list of clauses $C_1, ..., C_n, (x \vee \overline{x}')$. We add the clauses in order, deriving $(x \vee \overline{x}')$. The clauses are subsumed by $(x \vee \overline{x}')$ and deleted. If defining variable elimination does not produce $(x \vee \overline{x}')$, then the definition is not right-unique. The variable x cannot be moved in this case. ONE-SIDED: assuming for the one-sided definition that x occurs positively in the defining clauses, the implication $(\overline{x}' \vee x)$ is added. The implication is Q-blocked-subsumed for the same reasons as the first implication above. If x occurs negatively the implication $(\overline{x} \vee x')$ is added. We will continue the remaining steps under the assumption that x occurs positively in the defining clauses for the one-sided case.

3. *Add and remove the remaining clauses $\rho(x)$ and $\rho(\overline{x})$.*
   Let $\rho(x') = \rho(x)[x'/x]$. For all clauses $C \in \rho(x)$, $C' \in \rho(x')$ is the Q-resolvent of C with $(\overline{x} \vee x')$ on pivot x, so C' can be added. C can be deleted because it is the Q-resolvent of C' with $(\overline{x}' \vee x)$ on pivot x'. Similar reasoning is used for $C \in \rho(\overline{x})$. ONE-SIDED: All $C' \in \rho(x')$ are added with the same reasoning as above. However, there is no $(\overline{x} \vee x')$ so $C \in \rho(\overline{x})$ cannot be deleted until step 5.

4. *Remove the equivalence clauses $x \leftrightarrow x'$*
   Equivalence clauses $(x \vee \overline{x}'), (\overline{x} \vee x')$ are deleted. $(x \vee \overline{x}')$ is Q-blocked-subsumed on variable x since for all $D \in \delta(\overline{x})$, the outer resolvent of $(x \vee \overline{x}')$ and D is subsumed by the defining clause $D' \in \delta(\overline{x}')$, and the outer resolvent of $(x \vee \overline{x}')$ with $(\overline{x} \vee x')$ is a tautology. Similarly, $(\overline{x} \vee x')$ is Q-blocked-subsumed. ONE-SIDED: the definition clauses need the implication in order to be deleted, so this step is skipped.

5. *Remove the defining clauses $\delta(x)$ and $\delta(\bar{x})$.*
   The defining clauses on x are all Q-blocked and are deleted. ONE-SIDED: The defining clauses $D \in \delta(x)$ can be deleted because they are Q-resolvents of $D' \in \delta(x')$ with $(\overline{x}' \vee x)$ on x'. Now the clauses $(\overline{x}' \vee x)$ and $\rho(\overline{x})$ are Q-blocked on x because x only occurs negatively. They are deleted.

Given the QBF $\Pi.\psi$, applying the transformation sequentially with definition variables $x_1, \ldots, x_n$ will yield the QBF $\Pi.\psi'$ where all definition variables x_i have been replaced by new variables x_i' and the new variables are in the appropriate quantifier levels. The concatenated series of clause additions and deletions generated for each definition variable gives a QRAT proof of the equivalence between $\Pi.\psi$ and $\Pi.\psi'$

The steps above can also be used to move a definition variable to some existential quantifier between the variable and its innermost defining variable. In addition, a definition variable that is inside its defining variables can be moved further inwards by reversing the steps, but it is not clear when this would be useful.

Example 4. Given the QBF $\exists x_1 \forall x_2.(x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_2)$, we have the definition $x_1 \leftrightarrow x_2$. The definition is right-unique but the defining clauses are not Q-blocked on

x_1 since x_1 is at an outer quantifier level. The QBF is false but moving x_1 inward would make it true. To avoid this, we only move variables outward.

Example 5. Given the definition $x_1 \oplus x_2 \oplus x_3$ with x_1 as the definition variable we have $\delta(x_1) = \{(x_1 \vee x_2 \vee x_3), (x_1 \vee \overline{x}_2 \vee \overline{x}_3)\}$ and $\delta(\overline{x}_1') = \{(\overline{x}_1' \vee \overline{x}_2 \vee x_3), (\overline{x}_1' \vee x_2 \vee \overline{x}_3)\}$. Defining variable elimination will perform the following steps:

Eliminate x_2 :$\{(x_1 \vee x_2 \vee x_3) \otimes_{x_2} (\overline{x}_1' \vee \overline{x}_2 \vee x_3), (\overline{x}_1' \vee x_2 \vee \overline{x}_3) \otimes_{x_2} (x_1 \vee \overline{x}_2 \vee \overline{x}_3)\}$

$$\theta_1 = \{(x_1 \vee \overline{x}_1' \vee x_3), (x_1 \vee \overline{x}_1' \vee \overline{x}_3)\}$$

Eliminate x_3 :$\{(x_1 \vee \overline{x}_1' \vee x_3) \otimes_{x_3} (x_1 \vee \overline{x}_1' \vee \overline{x}_3)\}$

$$\theta_2 = \{(x_1 \vee \overline{x}_1')\}$$

The clause additions to derive the second implication in step 2 would be $(x_1 \vee \overline{x}_1' \vee x_3), (x_1 \vee \overline{x}_1' \vee \overline{x}_3), (x_1 \vee \overline{x}_1')$. Each subsequent clause in the list is implied by Q-resolution. With more defining variables, the resolution tree becomes more complex. The derivation will be of the form $\theta_1', ..., \theta_{n-1}'$ for $\theta_i' \subset \theta_i$ where θ_i' will include only the clauses needed to derive $(x_1 \vee \overline{x}_1')$. These can be determined by working through the resolution chain backwards from $(x_1 \vee \overline{x}_1')$.

Example 6. Given the formula $\exists x_1 x_2 x_3 \forall x_5 x_6 \exists x_4 (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_3 \vee \overline{x}_4) \wedge (x_4 \vee \overline{x}_5) \wedge (x_4 \vee \overline{x}_6)$, we show the steps generating the QRAT proof of movement for variable x_4 with the pivot appearing as the first literal in the clause. Clauses following a d are deleted from the formula.

1. $(x_4' \vee x_1 \vee x_2 \vee x_3), (\overline{x}_4' \vee \overline{x}_1), (\overline{x}_4' \vee \overline{x}_2), (\overline{x}_4' \vee \overline{x}_3)$
2. $(\overline{x}_4' \vee x_4), (x_4' \vee \overline{x}_4)$
3. $(x_4' \vee \overline{x}_5), d(x_4 \vee \overline{x}_5), (x_4' \vee \overline{x}_6), d(x_4 \vee \overline{x}_6)$
4. $d(x_4 \vee \overline{x}_4'), d(\overline{x}_4 \vee x_4')$
5. $d(x_4 \vee x_1 \vee x_2 \vee x_3), d(\overline{x}_4 \vee \overline{x}_1), d(\overline{x}_4 \vee \overline{x}_2), d(\overline{x}_4 \vee \overline{x}_3)$

The definition variable x_4 is replaced by the fresh variable x_4' which will be placed in the prenex as $\exists x_1 x_2 x_3 \exists x_4' \forall x_5 x_6$ achieving the desired movement. The QRAT proof system uses a stronger redundancy notion that avoids auxiliary clauses for an AND definition in step 2.

We verified all instances of variable movement on QBFEVAL'20 benchmarks using QRAT-TRIM [8]. By default, QRAT-TRIM will check a satisfaction proof with forward checking, verifying the clause deletion steps are correct in order they appear. A refutation proof is checked with backward checking, verifying the clause addition steps are correct starting at the empty clause and working backwards. Since we do not know whether the problem will be true or false at the variable movement stage, we must check both clause addition and deletion steps to preserve equivalence. To do this, we modified QRAT-TRIM with a DUAL-FORWARD mode that performs a forward check on both clause additions and deletion. We verified several end-to-end proofs for formulas solved by BLOQQER after variable movement. We append the BLOQQER proof onto our variable movement proof, and verify it against the original formula with QRAT-TRIM. All formulas that BLOQQER solved were verified in this way.

5   Evaluation

Variable movement is evaluated on the QBFEVAL'20 benchmarks. We compare the definition detection tools KISSAT and CNFTOOLS. Then we evaluate the affect of variable movement on solver performance. The repository can be found at `https://github.com/jreeves3/qbf-definition-variable-movement-Artifact`.

We ran our experiments on StarExec [18]. The specs for the compute nodes can be found online. The compute nodes that ran our experiments were Intel Xeon E5 cores with 2.4 GHz, and all experiments ran with 32 GB.

5.1   Evaluating Definition Detection

The tools are given 10 seconds to detect definitions. KISSAT attempts to check if each variable, whereas CNFTOOLS will iterate through root clauses until the time limit. Root clause selection is split into the *max-var* (mv) and *blocked* (mb) heuristics. We consider all definitions extracted up to a timeout if one occurs. The combined approach takes the union of definitions found in each tool, and each tool is still allotted 10 seconds.

Figure 1 shows the number of definitions found (top) and moved (bottom) compared to the combined approach. None of the tools goes above the diagonal in either plot because the combined approach takes a union of found definitions and movement cannot be diminished by additional detected definitions. In many cases multiple tools contribute to the combined found and combined movement, shown by a column of points where none are on the diagonal. There is a noticeable pattern between CNFTOOLS (mb) and (mv) where (mb) performs slightly worse due to the cost of computing the minimally-blocked root clause. But there are some instances where the minimally-blocked heuristic is advantageous and leads to more movement. For combined, definitions were found in 494 instances and moved in 157 instances Comparing the plots to each other, the number of definitions found is not a strict predictor of movement. KISSAT finds a similar number of definitions as CNFTOOLS for many instances but consistently moves more. Table 1 shows the breakdown of definitions found and moved by type. Many more of the definitions moved are AND/OR definitions.

Table 1 shows the breakdown of variable movement by definition type. CNFTOOLS has similar syntactic definition detection similar to KISSAT for BiEQ, AND/OR, XOR, but fails to move a fraction of the XOR definitions. CNFTOOLS does detect tens of XORs as monotonic definitions with the wrong definition variable, meaning the BFS picked up nested definitions in the wrong direction w.r.t. quantifier levels. But, the reason for the large gap between CNFTOOLS and KISSAT is efficiency. CNFTOOLS does not detect the vast majority of XOR definitions moved by KISSAT within the time limit, and the same is true for the other definitions. KISSAT uses the entire 10 seconds on 11 formulas whereas CNFTOOLS times out on 111 (mv) and 99 (mb). Increasing the timeout for each tool in the combined approach to 50 seconds produces only 780 more moved variables over 2 formulas. It is clear from the bottom plot in Figure 1 that CNFTOOLS contributes to the movement of the combined approach in a handful of cases where KISSAT is not on the diagonal. Combining the output of the tools makes use of KISSAT's speed in detecting many simple definitions and CNFTOOLS's ability to find one-sided definitions using complex heuristics and search.
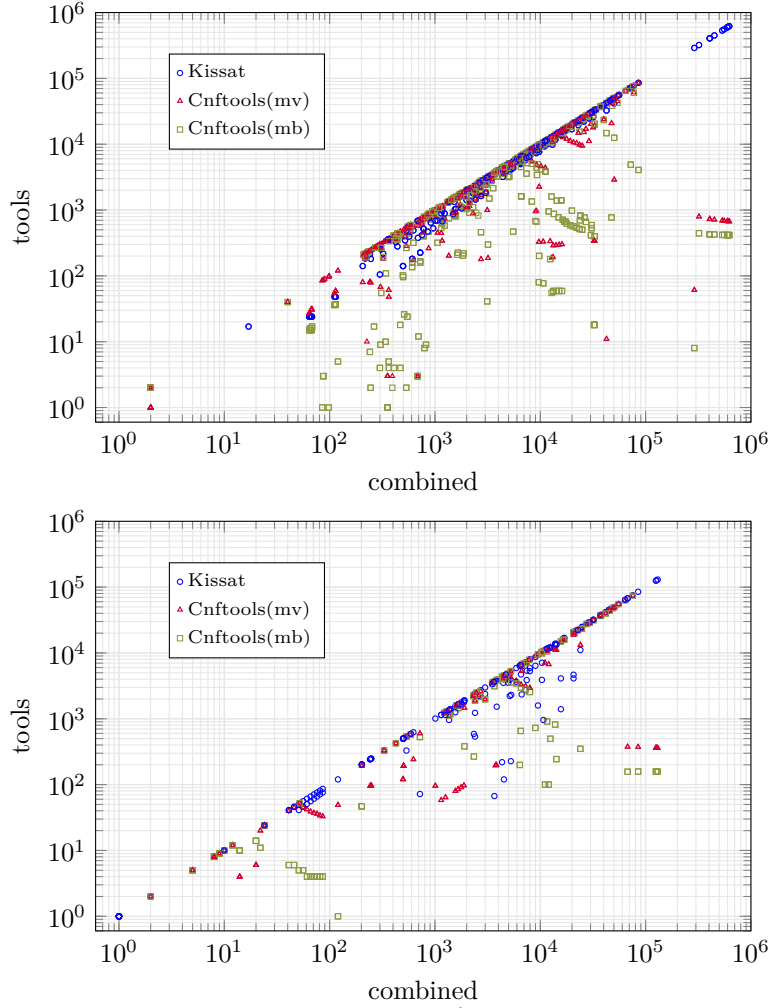
Fig. 1. Comparison of definitions found (top) and moved (bottom) per instance between combined and the individual tools.

No variables found by semantic detection were moved in KISSAT and only 88 were moved in CNFTOOLS (mb). KISSAT found 159,544 right-unique definitions with KITTEN, but only 23,457 were left-total. Of those, the majority had defining variables in the same level as the definition variable, and a smaller fraction had the definition variable at an outer level. Similarly with CNFTOOLS, 48,715 (mb) and 147,170 (mv) semantic definitions were detected via. right-uniqueness checks. These semantic definitions may not be introduced or manipulated by users in the same way as the standard definitions, and this would explain why they already occur in the desired quantifier level.

By far the most common reason definitions cannot be moved is that they already appear in the same quantifier level as some of their defining variables. For example,

Table 1. The number of definitions found and moved over all instances. Definitions moved are broken down by a selection of the types, omitting ITE and semantic. Some one-sided definitions CNFTOOLS moves are fully-defined, and combined will move them based on the fully-defined definition provided by KISSAT. So, the missing one-sided definitions for combined are spread across the other definition types.

Detection Tool	Found	Moved	BiEQ	AND/OR	One-Sided	XOR
CNFTOOLS(mv)	3,525,559	1,032,807	21,198	969,630	37,642	0
CNFTOOLS(mb)	2,856,306	935,336	4,619	891,027	39,863	0
KISSAT	9,243,158	1,567,746	308,987	1,215,036	—	42,364
combined	9,624,654	1,664,655	309,793	1,273,381	37,646	42,476

Table 2. The number of definitions found that were not left-total, split by existentially and universally quantified variables, along with monotonic definitions that could not be moved because they were not one-sided. If any universally quantified variable was left-total, the formula would be trivially false.

Detection Tool	Existential	Universal	One-sided
CNFTOOLS(mv)	43,278	11,360	1,107
CNFTOOLS(mb)	23,690	3,771	1,421
KISSAT	32,681	3,219	—

many formulas have only two quantifier levels, so there would be no possible movement with all existential variables in the same level. Table 2 shows other reasons a variable may not be moved. A definition is not left-total when the definition variable is at a level outer to some of its defining variables. The tools detected several of these definitions on both universally and existentially quantified variables. Example 2 shows why these variables cannot be moved inwards. Additionally, some of the monotonic definitions extracted by CNFTOOLS are neither fully-defined nor one-sided. This is determined online when a definition variable becomes a candidate for moving. Waiting to make these checks until moving is optimal because many definitions are detected, and a large fraction will be filtered out because of their quantifier level placement.

CNFTOOLS detect $2,038,407$ (mv) and $1,897,482$ (mb) monotonic definitions, but this does not match up with the number of one-sided definitions moved. The majority of monotonic definitions found and moved are actually fully defined. This means for many of the definitions, either $\delta(x)$ or $\delta(\bar{x})$ can be removed from the QBF while preserving equivalence. This can be done in QRAT by recursing through the monotonic definitions and deleting the redundant defining clauses. The large number of fully defined monotonic definitions shows that formulas written in QBF generally do not take advantage of efficient encodings including the Plaisted-Greenbaum transformation.

5.2 Evaluating Solvers

We used the following solvers to evaluate the impact of variable movement.

- RAREQS (Recursive Abstraction Refinement QBF Solver) [11] pioneered the adoption of counterexample guided abstraction refinement (CEGAR)-driven recursion and learning in QBF solvers. The 2012 version has comparable performance to current top-tier solvers.
- CAQE (Clausal Abstraction for Quantifier Elimination) [17] is the first place winner of the 2017, 2018, and 2020 competitions. The solver is written in RUST and based on the CEGAR clausal abstraction algorithm.
- DEPQBF uses the adapted DPLL algorithm QDPLL, relying on dependency schemes to select independent variables for decision making [14]. DEPQBF incorporates QBCE [2] as inprocessing which complicates its relation to preprocessors like BLOQQER.
- GHOSTQ is a non-clausal QBF solver [13]. The solver attempts to convert CNF or QCIR to the GhostQ format which introduces Ghost variables, dual of Tseitin variables. The structural information gained by the conversion is important to GHOSTQ's performance. The conversion relies on the discovery of definitions, which is significantly hampered by preprocessors that delete or change clauses. GHOSTQ also supports a CEGAR extension.

Table 3. The number of instances solved within the 5,000 time-limit over benchmarks where variable movement was possible.

Solver	Original	Moved	BLOQQER	Moved-BLOQQER
CAQE	74	84	99	**103**
GHOSTQ(p)	55	**61**	47	52
GHOSTQ(ce)	77	**80**	65	70
RAREQS	72	72	94	**98**
DEPQBF	64	70	64	**71**

Figure 2 shows the performance of QBF solvers on the original (-o) and moved (-m) formulas using the combined definition detection. The times include definition detection and proof generation, adding 50 seconds on average. In moved formulas, adjacent quantifier levels of the same type were conjoined into a single quantifier level because of GHOSTQ's internal definition detection. This did not impact the other solvers. Movement significantly improves performance of CAQE, DEPQBF, and GHOSTQ-p (plain mode). GHOSTQ-ce (CEGAR mode) and RAREQS improve slightly with movement. Since both GHOSTQ modes use the same conversion to the GHOSTQ format, the impact of variable movement on the conversion does not explain the difference. GHOSTQ in CEGAR mode and RAREQS use similar solving techniques, and this may explain why they improve little. Separate experiments moving all definitions except XORs did improve the performance of GHOSTQ in both modes while not affecting other solvers. This is because the conversion to the GHOSTQ format only checks the innermost quantifier level for XOR definitions, and cannot find them if they have been moved.

Most state-of-the-art QBF solvers make use of preprocessors. The exception is GHOSTQ because its definition detection suffers after the application of QBCE. Fig-

13
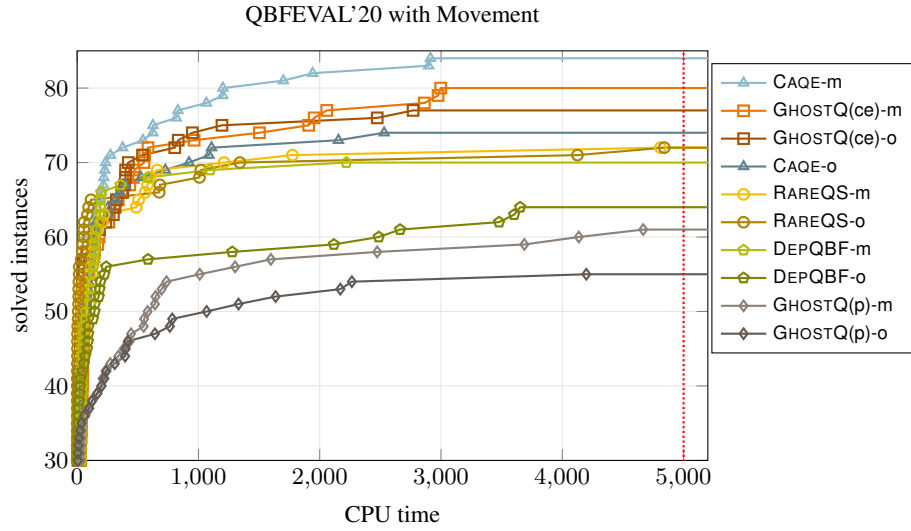
Fig. 2. Cumulative number of solved instances considering only the 157 benchmarks which had variables that could be moved.
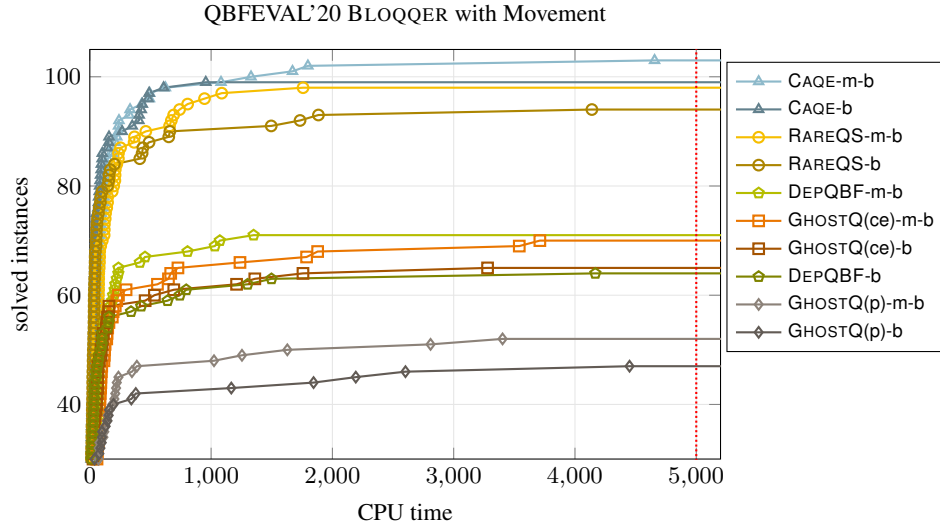
Fig. 3. Cumulative number of solved instances after applying BLOQQER for 100 seconds considering only the 157 benchmarks with movement.

ure 3 shows solver performance with moving variables before applying BLOQQER (m-b) and only applying BLOQQER (-b). The solving time includes the variable movement and BLOQQER runtime within a 100 second timeout. After moving variables, BLOQQER solved 3 formulas and those data are reflected in the plot. In addition, each of the

14

14 formulas BLOQQER solved before movement, BLOQQER also solved after move-
ment. This data is constant across solvers so is not included in the plot. Performance
improved for all solvers when applying variable movement before BLOQQER. One rea-
son for this is movement may allow for more applications of universal reduction. We
also experimented with moving variables after BLOQQER preprocessed the formulas.
Few variables were moved, and it did not affect solver performance. This is likely due
to QBCE removing defining clauses from the formula.

6 PGBDDQ Case Study

PGBDDQ is a binary-decision diagram (BDD)-based proof-generating QBF solver [3].
PGBDDQ starts at the innermost quantifier and performs bucket elimination, lineariz-
ing variables and eliminating them through a series of BDD operations that are equivalence-
preserving. As BDDs are manipulated, PGBDDQ generates a dual proof through a
series of clause additions and deletions. To achieve good performance, the definition
variables must be placed in quantifier levels close to their defining variables. If def-
inition variables were all placed in the innermost quantifier level, PGBDDQ would
eliminate them first, requiring the BDDs to encode the solver operation in terms of the
complete history of all the moves, rather than in terms of the effect each move has on
the board state.

Two player games can be succinctly represented in QBF, thought of as an existential
player versus a universal player. Moves alternate between quantifier levels, and defini-
tion variables carry information as moves are played over time. Given a $1 \times N$ board,
the *linear domino placement* game involves two players taking turns placing a 1×2
domino onto the board until it is filled. The first player that cannot place a domino loses.
The game can be encoded with around $N^2/2$ problem variables and $3N^2/2$ definition
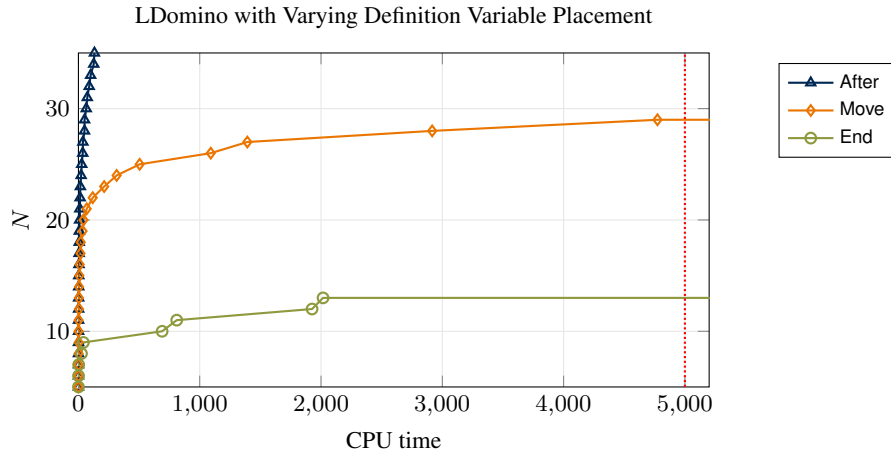
Fig. 4. Performance on boards of size N for false formulas where player two wins. The Move
placement times out at $N = 30$ and the End placement runs out of memory at $N = 14$.

15

variables. PGBDDQ can solve the problem with polynomial performance when definitions are placed after their defining variables (After). The performance deteriorates when definition variables are placed in the innermost quantifier level (End).

Figure 6 shows the performance of PGBDDQ on false formulas where the second player will win. In each configuration, the same hand-crafted BDD variable ordering was used. Without variable movement, PGBDDQ runs out of memory on 32 GB RAM at $N = 12$. With movement, the solver performs significantly better and solves all formulas up to $N = 30$ before timeouts occur. This shows how the general problem of memory inefficiency within a BDD can be eased by moving definition variables across quantifier levels. The gap in performance between the Move placement and the After placement may be due to the ordering of variable within a quantifier block or moving variables too far outward. When a variable is moved it can be placed anywhere within a quantifier level as this does not change semantics. Also, variables do not need to be moved all the way to their innermost defining variable. Exploring these options in the context of a structurally dependent solver PGBDDQ may lead to improvements that affect other QBF solvers.

7 Conclusion and Future Work

We presented a technique for moving definition variables in QBFs. The movement can be verified within the QRAT proof system, and we validated all proofs in the evaluation with QRAT-TRIM. Using the tools KISSAT and CNFTOOLS to detect definitions, we created a tool-chain for variable movement. On the QBFEVAL'20 benchmarks, one quarter of formulas had definitions that could be moved, and the movement increased solver performance. In addition, we found that movement followed by BLOQQER was more effective than preprocessing with BLOQQER.

For future work, incorporating quantifier level information into definition detection could reduce the costs. For example, the hierarchical detection could recurse outwards based on quantifier levels, reducing the number of root clauses explored and reducing the number of incorrect definitions detected. Passing the definition information to solvers could help guide search and in-processing.

In addition, there are ways to expand on variable movement. It is possible to place variables anywhere within a given quantifier level and also to adjust how far variables are moved. Separately, monotonic definitions that are not one-sided present an interesting challenge for variable movement, as they occur in both polarities outside of the definition. It might also be possible to move the approximately 160,000 semantic definitions found be KITTEN that were right-unique but not left-total.

8 Acknowledgements

References

1. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. Tech. rep. (2020)
2. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Automated Deduction (CADE). pp. 101–115. Springer (2011)
3. Bryant, R.E., Heule, M.J.H.: Dual proof generation for quantified Boolean formulas with a BDD-Based solver. In: Automated Deduction (CADE). pp. 433–449. Springer (2021)
4. Davis, M., Putnam, H.: A computing procedure for quantification theory. ACM **7**(3), 394–397 (Jul 1962)
5. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Theory and Applications of Satisfiability Testing (SAT). LNCS, vol. 3569, pp. 61–75. Springer (2005)
6. Fleury, M., Biere, A.: Mining definitions in Kissat with Kittens. In: Proceedings of Pragmatics of (SAT) (2021)
7. Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension free proof systems. In: Journal of Automated Reasoning. vol. 64, pp. 533–544 (2020)
8. Heule, M.J.H., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: Automated Reasoning. pp. 91–106. Springer, Cham (2014)
9. Heule, M.J., Hunt, W.A., Wetzler, N.: Trimming while checking clausal proofs. In: 2013 Formal Methods in Computer-Aided Design (FMCAD). pp. 181–188 (2013)
10. Iser, M.: Recognition and Exploitation of Gate Structure in SAT Solving. Ph.D. thesis, Karlsruhe Institute of Technology (KIT) (2020)
11. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. Artificial Intelligence **234**, 1–25 (2016)
12. Kleine Buning, H., Karpinski, M., Flogel, A.: Resolution for quantified boolean formulas. Inf. Comput. **117**(1), 12–18 (Feb 1995)
13. Klieber, W., Sapra, S., Gao, S., Clarke, E.: A non-prenex, non-clausal QBF solver with game-state learning. In: Theory and Applications of Satisfiability Testing (SAT). pp. 128–142. Springer (2010)
14. Lonsing, F.: Dependency Schemes and Search-Based QBF Solving: Theory and Practice. Ph.D. thesis, Johannes Kepler University (JKU) (2012)
15. Lonsing, F.: QBFRelay, QRATPre+, and DepQBF: Incremental preprocessing meets search-based QBF solving. Journal on Satisfiability, Boolean Modeling and Computation **11**, 211–220 (09 2019)
16. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation **2**(3), 293–304 (1986)
17. Rabe, M.N., Tentrup, L.: CAQE: A certifying QBF solver. In: Formal Methods in Computer-aided Design (FMCAD). pp. 136–143 (September 2015)
18. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: International Joint Conference on Automated Reasoning (IJCAR). LNCS, vol. 8562, pp. 367–373. Springer (2014)
19. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer (1983)
20. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Theory and Applications of Satisfiability Testing (SAT). pp. 422–429. Springer (2014)
21. Wimmer, R., Reimer, S., Marin, P., Becker, B.: HQSpre – an effective preprocessor for QBF and DQBF. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 373–390. Springer (2017)