Extended Failed-Literal Preprocessing for Quantified Boolean Formulas*

Allen Van Gelder[1], Samuel B. Wood[1], and Florian Lonsing[2]

[1] University of California, Santa Cruz
[2] Johannes Kepler University

Abstract. Building on recent work that adapts failed-literal analysis (FL) to Quantified Boolean Formulas (QBF), this paper introduces extended failed-literal analysis (EFL). FL and EFL are both preprocessing methods that apply a fast, but incomplete reasoning procedure to abstractions of the underlying QBF. EFL extends FL by remembering certain binary clauses that are implied by the same reasoning procedure as FL when it assumes one literal and that implies a second literal. This extension is almost free because the second literals are implied anyway during FL, but compared to analogous techniques for propositional satisfiability, its correctness involves some subtleties. For the first time, application of the universal pure literal rule is considered without also applying the existential pure literal rule. It is shown that using both pure literal rules in EFL is unsound. A modified reasoning procedure for QBF, called Unit-clause Propagation with Universal Pure literals (UPUP) is described and correctness is proved for EFL based on UPUP. Empirical results on the 568-benchmark suite of QBFEVAL-10 are presented.

Keywords: quantified boolean formulas, QBF, failed literals, extended failed literals, 1-saturation, look-ahead, preprocessing

1 Introduction

With the advent of capable solvers for Quantified Boolean Formulas (QBFs), their use for encoding problems from industrial applications is increasing rapidly. As with propositional satisfiability, preprocessors have been found to be an important part of the QBF solving toolkit. Preprocessors typically do a predictable (polynomially bounded) amount of work to simplify the original formula, making it more amenable to the complete solver. The newer complete QBF solvers are typically based on search, a form of back-chaining, whereas preprocessors use forward reasoning. The two approaches often complement each other nicely.

The essence of failed-literal analysis is to add an assumption that some literal is true to a given formula and use incomplete (but usually fast) forward reasoning to see if the formula can now be proven false; if so, then the *negation* of the assumed literal can soundly be added to the formula. This idea was introduced for propositional SAT solving by Jon Freeman [Fre93].

* `http://www.cse.ucsc.edu/{~avg,~sbwood}, http://fmv.jku.at/lonsing`

This paper builds upon recent work of Lonsing and Biere [LB11] that adapts failed-literal analysis to QBF solving. That work uses the QBF pure-literal rule [GNT04] heavily in its incomplete forward reasoning, which they call *QBCP*. We re-examine the QBF pure-literal rule, which applies to both existential and universal literals, and consider those parts separately. We observe that the two parts operate quite differently and have different properties. We show that neither part is a *super-sound inference rule* (in the sense that tree models are preserved, as specified in Section 2); they are only *safe heuristics* (in the sense that the truth value of a closed QBF is not changed by their use). (This fact is well-known for propositional formulas and the *existential* pure-literal rule, but seems not to have been considered for the *universal* pure-literal rule in QBF.) We show that using them both as though they were super-sound logical inferences can lead to fallacious reasoning in QBF. We believe this is the first time that application of the universal pure literal rule has been considered without also applying the existential pure literal rule.

Then we develop an enhancement of failed literal analysis based on propositional techniques called *1-saturation*, described by Groote and Warners [GW00] and *double unit-propagation look-ahead* described by Le Berre [LB01]. We show that using the universal pure-literal rule *and not* the existential pure-literal rule is safe for 1-saturation in QBF.

The essence of *1-saturation* in propositional logic is the observation that, with a given formula, if assuming some literal q allows some other literal p to be soundly derived, and if assuming $\overline{q}$ also allows p to be soundly derived, then p can soundly be added to the formula as a unit clause. Unfortunately for QBF, literals derived with *QBCP* are not necessarily super-soundly derived.

We introduce an incomplete forward reasoning procedure that employs unit-clause propagation (including universal reduction) and the *universal pure-literal rule*. We show that this procedure super-soundly derives literals. We call it *UPUP* for Unit-clause Propagation with Universal Pure literals.

Although *UPUP* is weaker than *QBCP* in the sense that it assigns values to fewer literals, we found experimentally that it is considerably faster, for simple failed-literal analysis. In addition, it serves as the basis for adapting *1-saturation* to QBF, and it can log proof steps that are verifiable as *Q-resolution* steps [KBKF95]. We call our adaptation of 1-saturation to QBF ***extended failed literal analysis* (EFL)**.

One reason for our interest in EFL is that it enables a significant fraction of the popular QBFLIB benchmark suite to be solved with preprocessing alone. The first QBF preprocessor to solve a significant number of benchmarks in this suite was `sQueezeBF`, described by Giunchiglia *et al.* [GMN10]. With their publicly available binary code, `QuBE-7.2`, we solved 40 of the 568 QBFLIB benchmarks. Lonsing and Biere reported [LB11] that their QBF failed-literal tool, publicly available as `qxbf`, processing the output of `sQueezeBF` 7.1 when it did not solve the instance, solved an additional 25 benchmarks. We confirmed the same result with `sQueezeBF` 7.2.

Another reason for our interest in EFL is that it can be used without any pure-literal rule to simplify QBFs without changing the set of tree models. This can be important in applications where the tree models themselves are important. Other popular preprocessors make changes that preserve the value (true or false) but add and delete tree models as they operate on a true QBF.

The paper is organized as follows.[3] Section 2 sets forth the notation and basic definitions. Section 3 reviews QBF forms of pure-literal rules. Section 4 reviews QBF abstraction and its combination with failed literal analysis. Extended failed literal analysis (EFL) is introduced. Existential and universal pure-literal rules are separated and soundness issues are examined. The main theoretical result is that EFL with abstraction and the universal pure-literal rule is safe. Experimental results are presented in Section 5. The paper concludes with Section 6.

2 Preliminaries

In general, *quantified boolean formulas* (QBFs) generalize propositional formulas by adding operations consisting of universal and existential quantification of boolean variables. See [KBL99] for a thorough introduction. This paper uses standard notation as much as possible. One minor variation is that we consider resolution and universal reduction separately, although some papers combine them. Also, we use tree models, which are not found in all QBF papers, to distinguish between *super-sound* and *safe* operations, and we define ordered assignments.

A *closed* QBF evaluates to either *invalid* (false) or *valid* (true), as defined by induction on its principal operator. We use 0 and 1 for truth values of literals and use true and false for semantic values of formulas.

1. $(\exists\, x\, \Phi(x))$ is true if and only if ($\Phi(0)$ is true or $\Phi(1)$ is true).
2. $(\forall\, x\, \Phi(x))$ is false if and only if ($\Phi(0)$ is false or $\Phi(1)$ is false).
3. Other operators have the same semantics as in propositional logic.

This definition emphasizes the connection of QBF to two-person games, in which player E (Existential) tries to set existential variables to make the QBF evaluate to true, and player A (Universal) tries to set universal variables to make the QBF evaluate to false. Players set their variable when it is outermost, or for non-prenex, when it is the root of a subformula (see [KSGC10] for more details). Only one player has a winning strategy.

We say that a QBF is in *prenex conjunction normal form* if all the quantifiers are outermost operators (the prenex, or quantifier prefix), and the quantifier-free portion (also called the matrix) is in CNF; i.e., $\Psi = \overrightarrow{Q}.\mathcal{F}$ consists of prenex $\overrightarrow{Q}$ and matrix $\mathcal{F}$. Clauses in $\mathcal{F}$ are called *input clauses*. For this paper QBFs are in prenex conjunction normal form.

For this paper a *clause* is a *disjunctively* connected set of literals. Literals are variables or negated variables, with overbar denoting negation. Clauses may be

[3] See `http://www.cse.ucsc.edu/~avg/EFL/` for a longer version of this paper and other supplementary materials.

written as literals enclosed in square brackets (e.g., $[p, q, \overline{r}]$), and [] denotes the empty clause. Where the context permits, letters e and others near the beginning of the alphabet denote existential literals, while letters u and others near the end of the alphabet denote universal literals. Letters like p, q, r denote literals of unspecified quantifier type. The variable underlying a literal p is denoted by $|p|$ where necessary.

The quantifier prefix is partitioned into maximal contiguous subsequences of variables of the same quantifier type, called *quantifier blocks*. Each quantifier block has a unique *qdepth*, with the outermost block having qdepth = 1. The *scope* of a quantified variable is the qdepth of its quantifier block. We say scopes are *outer* or *inner* to another scope to avoid any confusion about the direction, since there are varying conventions in the literature for numbering scopes.

Definition 2.1 (Assignment) An *assignment* is a partial function from variables to truth values, usually represented as the set of literals mapped to 1. A *total assignment* is an assignment to all variables. Assignments are denoted by ρ, σ, τ. Application of an assignment σ to a logical expression is called a ***restriction*** and is denoted by $q\lceil_\sigma$, $C\lceil_\sigma$, $\mathcal{F}\lceil_\sigma$, etc. Quantifiers for assigned variables are deleted in $\Psi\lceil_\sigma$.

An ***ordered assignment*** is a special term that denotes a total assignment that is represented by a sequence of literals that are assigned 1 and are in the same order as their variables appear in the quantifier prefix. □

A *winning strategy* can be presented as an unordered directed tree. If it is a winning strategy for the E player, it is also called a ***tree model***, which we now describe. We shorten *unordered directed tree* to *tree* throughout this paper. The qualifier "unordered" means that the children of a node do not have a specified order; they are a set. Recall that a ***branch*** in a tree is a path from the root node to some leaf node. A tree can be represented as the set of its branches. We also define a a ***branch prefix*** to be a path from the root node that might terminate before reaching a leaf.

Definition 2.2 (Tree model) Let a QBF $\Phi = \overrightarrow{Q} \cdot \mathcal{F}$ be given. In this definition, σ denotes a (possibly empty) branch prefix of some ordered assignment for Φ. A *tree model* M for Φ is a nonempty set of ordered assignments for Φ that defines a tree, such that

1. Each ordered assignment makes $\mathcal{F}$ true, i.e., *satisfies* $\mathcal{F}$ in the usual propositional sense.
2. If e is an existential literal in Φ and some branch of M has the prefix (σ, e), then *no* branch has the prefix $(\sigma, \overline{e})$; that is, treating σ as a tree node in M, it has only one child and the edge to that child is labeled e.
3. If u is an universal literal in Φ and some branch of M has the prefix (σ, u), then *some* branch of M has the prefix $(\sigma, \overline{u})$; that is, treating σ as a tree node in M, it has two children and the edges to those children are labeled u and $\overline{u}$.

Although the wording is different, if M is a tree model by this definition, it is also a tree model by definitions found in other papers [SB07,LB11]. If τ is a partial

assignment to all variables outer to existential variable e, then requirement 2 ensures that the "Skolem function" $e(\tau)$ is well defined as the unique literal following τ in a branch of M. If the formula evaluates to false, the set of tree models is empty. □

Definition 2.3 (Safe, Super sound) For this paper purposes, an operation on a closed QBF is said to be ***safe*** if it does not change the truth value of the formula. An operation on a closed QBF is said to be ***super sound*** if it preserves (i.e., does not change) the set of tree models. Clearly, preserving the set of tree models is a sufficient condition for safety. □

The proof system known as *Q-resolution* consists of two operations, *resolution* and *universal reduction*. Q-resolution is of central importance for QBFs because it is a *refutationally* complete proof system [KBKF95]. Unlike resolution for propositional logic, Q-resolution is not *inferentially* complete. That is, a (new) clause C might be a super-sound addition to a closed QBF Φ (i.e., C evaluates to true in every tree-model of Φ), yet no subset of C is derivable by Q-resolution (see [LB11], example 6).

Definition 2.4 (Resolution, Universal reduction) Resolution is defined as usual. Let clauses $C_1 = [q, \alpha]$ and $C_2 = [\overline{q}, \beta]$, where α and β are literal sequences without conflicting literals among them and without q and $\overline{q}$. (Either or both of α and β may be empty.) Also, let the *clashing literal* q be an existential literal. Then $\mathsf{res}_q(C_1, C_2) = \alpha \cup \beta$ is the *resolvent*, which cannot be tautologous in Q-resolution.

Universal reduction is special to QBF. Let clause $C_1 = [q, \alpha]$, where α is a literal sequence without conflicting literals and without q and $\overline{q}$, the *reduction literal* q is a universal literal, and q is *tailing* for α. A universal literal q is said to be *tailing* for α if its quantifier depth is greater than that of any existential literal in α. Then $\mathsf{unrd}_q(C_1) = \alpha$. □

Lemma 2.5 Resolution and universal reduction are super-sound operations.

Proof: Straightforward application of the definitions. ■

3 Pure Literals in QBF

A literal is called *pure* or *monotone* if it appears in (the matrix of) the formula and its negation does not appear in the formula. In QBF the *pure-literal rule* consists of setting any existential pure literal to 1 and setting any universal pure literal to 0. As far as we know, the two parts of this rule have not been considered separately. It is well known that the *existential* pure-literal rule does not preserve tree models, since it does not preserve models in the propositional case. That is, if e is an existential pure literal, there may be tree models in which e is assigned 0 on some branches. In such a case, e is not a necessary assignment. Similarly, if u is a universal pure literal, deleting all occurrences of u in the matrix eliminates some tree models, in general. Combining abstraction, defined next, with the pure-literal rule can lead to fallacious conclusions if done carelessly.

4 QBF Abstraction

The idea of formula abstraction was introduced by Lonsing and Biere [LB11]. The idea is to create a formula that is easier to reason about for preprocessing purposes.

Definition 4.1 Let Φ be a closed QBF.

1. Let e be an existential variable in Φ. Then the ***abstraction of Φ with respect to*** e, denoted $abst(\Phi, e)$, is the formula in which all universally quantified variables with scopes outer to e are changed to existential variables.
2. Let u be a universal variable in Φ. Then the ***abstraction of Φ with respect to*** u, denoted $abst(\Phi, u)$, is the formula obtained as follows. First, transpose u to be outermost within its own quantifier block and change it to an existential variable. Then, change all universally quantified variables with scopes outer to u to existential variables. Any other variables with the same original scope as u remain universal in $abst(\Phi, u)$.

Thus the outermost scope of $abst(\Phi, p)$ is existential and contains p, whether p is existential or universal.[4] □

Theorem 4.2 [LB11] Let Φ be a closed QBF. Let p be a variable in Φ. Then the set of tree models of $abst(\Phi, p)$ is a superset of the set of tree models of Φ. □

We define a ***necessary assignment*** to be an assignment to a variable that occurs in every branch of every tree model. Adding a necessary assignment as a unit clause in the matrix is clearly super sound. The main idea is to find necessary assignments for existential variables in the outermost scope of abstractions of Φ. By Theorem 4.2, these are also necessary assignments for Φ when the variable is existential in Φ, as well. In the case that a necessary assignment is found for a universal variable of Φ that became existential due to abstraction, Φ must be false because every tree model has branches for both assignments to any universal variable.

Lonsing and Biere detect necessary assignments by using ***failed literal*** analysis: If the assumption that literal $p = 1$ in the outermost scope of $abst(\Phi, p)$ derives the empty clause using incomplete forward reasoning, then $\overline{p}$ is a necessary assignment for both $abst(\Phi, p)$ and Φ. For incomplete forward reasoning, they use the *QBCP* procedure, which consists of unit-clause propagation (including universal reduction), and pure literal propagation.

This paper shows how to extend this approach to include ***1-saturation***: Separately, assume p and assume $\overline{p}$ in the outermost scope of $abst(\Phi, p)$. If neither assumption derives the empty clause by incomplete forward reasoning,

[4] Our definition is worded slightly differently from [LB11], but is the same in practice. If p is universal, by their definition p remains universal in $abst(\Phi, p)$, whereas in our definition p becomes outer to other variables in its scope and switches to being existential. But p is assigned a truth value immediately after forming $abst(\Phi, p)$, so it does not matter whether p is considered existential or universal.

Φ	$\forall d$	$\forall v$	$\exists b$	$\exists c$	$\exists e$	$\exists f$	$\exists m$	$\forall w$	$\exists g$	$\exists h$	$\exists j$	$\exists k$	$\exists n$
C_1	d		b										
C_2									g	$\overline{h}$			
C_3				$\overline{c}$								k	
C_4							$\overline{m}$						$\overline{n}$
C_5			b								$\overline{j}$		
C_6					$\overline{e}$						j		
C_7								$\overline{w}$		h			
C_8	$\overline{d}$			c									
C_9		v	$\overline{b}$					w		$\overline{h}$			
C_{10}				c		$\overline{f}$							
C_{11}					e	f							
C_{12}		$\overline{v}$			e								n
C_{13}							m		$\overline{g}$			$\overline{k}$	
C_{14}		v					m						n

Fig. 1. QBF for Example 4.3.

intersect the sets of *variables* that were assigned during the two propagations to find additional necessary assignments and equivalent literals. If a variable q was assigned the same value after both assumptions, we want to add q as an additional necessary assignment. If the assumption of p derives literal q and the assumption of $\overline{p}$ derives $\overline{q}$, we want to add the constraint $p = q$ (as two binary clauses, say). Our extension needs to be done in such a way that these additions to Φ are at least *safe* and preferably *super sound*.

The following example shows that the combination of *QBCP*, abstraction, and 1-saturation is *unsafe*.

Example 4.3 Consider $abst(\Phi, b)$, where Φ is shown in Figure 1. Variables d and v are temporarily existential. Only the crucial assignments are mentioned.

The assumption of $\overline{b}$ satisfies C_9. Now $\overline{w}$ is universal pure, which implies h.

The assumption of b satisfies C_5. Existential pure literal propagation assigns true to j, e, v in that order, satisfying C_9. Now $\overline{w}$ is again universal pure, which implies h.

Consequently, EFL using both pure-literal rules would derive $h = 1$ as a necessary assignment. However, adding $[h]$ to Φ changes it from true to false. In particular, if $h = 1$ the A player can choose $d = 0$, $v = 0$, and $w = 0$, after which C_1 and C_9 form a contradiction.

The original Φ is true, as shown by the E player's strategy: $b = 1$, $c = 1$, $e = 1$, $f = 0$, $m = 1$, $g = 1$, $j = 1$, $k = 1$, $n = 0$, $h = w$. (The A player must choose w before the E player chooses h.) □

The main theoretical result of the paper is that the *universal* pure-literal rule *can* be combined safely with abstraction and 1-saturation. This is non-trivial because assigning a universal pure literal to 0 might *not* be a necessary assignment.

We now describe an incomplete forward reasoning procedure that employs unit-clause propagation (including universal reduction) and the *universal pure-literal rule*.

Definition 4.4 The procedure ***UPUP*** (for Unit-clause Propagation with Universal Pure literals) consists of applying the following steps to a QBF until none are applicable or an empty clause is derived.

1. **(Unit Prop)** If a clause C has exactly one unassigned existential literal e and all other literals in C are 0 by previous assignment or universal reduction, then assign $e = 1$.
2. **(Empty Clause)** If all literals in clause C are 0 by previous assignment or universal reduction, derive the *empty clause*.
3. **(Univ Pure)** If u is an unassigned *pure universal literal* based on previous assignments, then u may be deleted from all clauses in which it occurs, as if by universal reduction. (For bookkeeping, $u = 0$ might be assigned on this computation path, but we avoid treating it as a *derived* literal). □

Theorem 4.5 Let $\Phi = \overrightarrow{Q}.\mathcal{F}$ be a closed QBF, let e be an existential literal, and let p be a literal in Φ other than e or $\overline{e}$. Let $\Phi_1 = \overrightarrow{Q}.(\mathcal{F} \cup \{[p, \overline{e}]\})$. If assuming e in $abst(\Phi, |e|)$ derives p by UPUP, then Φ_1 has the same truth value as Φ. If the universal pure-literal rule is not used, then Φ_1 has the same set of tree models as Φ.

Proof: To show that Φ_1 has the same truth value as Φ, it suffices to show that if Φ_1 is false, then Φ is false, because Φ_1 has more constraints. To show that Φ_1 has the same set of tree models as Φ, it suffices to show that if M is a tree model of Φ, then M is a tree model of Φ_1. In this proof we identify the literal $\overline{p}$ with the assignment $p = 0$ and identify p with $p = 1$.

W.l.o.g., let e be innermost in its own quantifier block. If Φ_1 is false, then *every* tree model of Φ (if there are any) contains *some* branch on which $p = 0$ and $e = 1$. Let M be any tree model of Φ, represented as its set of ordered assignments, each ordered assignment being a branch in M (Definition 2.2). Let M_e be the subset of branches on which $e = 1$. By hypothesis, some branch of M_e has $p = 0$. Follow the steps by which p was derived in $abst(\Phi, |e|)$ after assuming e. Let the sequence of partial assignments σ_i, for $i \geq 1$, denote $e = 1$ followed by the derived assignments of UPUP, beginning with $\sigma_1 = \{e\}$. Each literal derived before any use of universal reduction or universal pure-literal rule is 1 on every branch of M_e. Universal reduction can only apply on literals with quantifier scope inner to e, and cannot falsify a clause or M would not be a tree model. If step i consists of universal reduction, $\sigma_i = \sigma_{i-1}$.

Consider the first application of the universal pure-literal rule, say at step $k + 1$. Suppose it deletes all occurrences of u in $\mathcal{F}$ because all occurrences of $\overline{u}$ are in satisfied clauses at this point. We treat this as an operation on $\mathcal{F}$, not an assignment, so $\sigma_{k+1} = \sigma_k$.

Group the branches of M_e by their literals outer to u. Let ρ be the (partial) assignment for any such group. Note that ρ specifies a branch prefix in M to a specific node whose children are u and $\overline{u}$; let us call this node N_ρ. Then $(\rho,\, u = 0)$ and $(\rho,\, u = 1)$ produce subtrees of N_ρ, which we denote as $T_{\rho,u=0}$ and $T_{\rho,u=1}$, respectively. That is, the set of branches in M_e consistent with ρ is equal to $((\rho, T_{\rho,u=0}) \cup (\rho, T_{\rho,u=1}))$, where ρ, T denotes a tree in which every branch has ρ as a prefix.

For each group replace $T_{\rho,u=1}$ by $T_{\rho,u=0}$, except that $u = 1$ replaces $u = 0$ in their ordered assignments. The assignment in the replacement branch still satisfies every clause of the matrix. Retain the branches in $M_{\overline{e}}$ unchanged. This gives another tree M'. This is a tree model of Φ because the variables that changed from universal to existential in $abst(\Phi, |e|)$ are all outer to e and u.

By the hypothesis that Φ_1 is false, some branch of M'_e has $p = 0$. Therefore, the $T_{\rho,u=0}$ corresponding to some ρ has $p = 0$ on some branch. Every branch of M' is consistent with σ_{k+1}. Use M' in the role of M for further tracking of the UPUP computation. If the universal pure-literal rule is not used, the original M is a tree model of Φ_1.

Continuing in this way, we see that after each stage i in the computation, there is some tree model of Φ that is consistent with σ_i on *every* branch that contains $e = 1$, and further, has $p = 0$ on *some* branch that has $e = 1$. Thus $p = 1$ cannot be derived by UPUP after assuming $e = 1$ if Φ is true and Φ_1 is false. (Recall that we do not treat p as "derived" if $\overline{p}$ is processed as a pure universal literal.) Similarly, if the universal pure-literal rule is not used, $p = 1$ cannot be derived by UPUP after assuming $e = 1$ if M is a tree model of Φ and not of Φ_1. This contradicts the hypothesis of the theorem, completing the proof. ■

Several corollaries follow from Theorem 4.5 and Lemma 2.5.

Corollary 4.6 Let $\Phi = \overrightarrow{Q}.\mathcal{F}$ be a closed QBF, let v be a universal literal, and let p be a literal in Φ other than v or $\overline{v}$. Let $\Phi_1 = \overrightarrow{Q}.(\mathcal{F} \cup \{[p, \overline{v}]\})$. If assuming v in $abst(\Phi, |v|)$ derives p by UPUP, then Φ_1 has the same truth value as Φ. □

Corollary 4.7 Let $\Phi = \overrightarrow{Q}.\mathcal{F}$ be a closed QBF, let q be a literal, and let p be a literal in Φ other than q or $\overline{q}$. Let $\Phi_3 = \overrightarrow{Q}.(\mathcal{F} \cup \{[p]\})$. If the assumption that $q = 1$ in $abst(\Phi, |q|)$ derives p by UPUP and the assumption that $q = 0$ in $abst(\Phi, |q|)$ derives p by UPUP, then Φ_3 has the same truth value as Φ. □

Corollary 4.8 Let $\Phi = \overrightarrow{Q}.\mathcal{F}$ be a closed QBF, let q be a literal, and let p be a literal in Φ other than q or $\overline{q}$. Let $\Phi_4 = \overrightarrow{Q}.(\mathcal{F} \cup \{[p, \overline{q}], [\overline{p}, q]\})$. If the assumption that $q = 1$ in $abst(\Phi, |q|)$ derives p by UPUP and the assumption that $q = 0$ in $abst(\Phi, |q|)$ derives $\overline{p}$ by UPUP, then Φ_4 has the same truth value as Φ. □

5 Experimental Results

This section describes our experimental procedures and shows the results. Several procedural issues are discussed first.

As reported, `qxbf` monitors its own CPU time for certain operations and discontinues that kind of operation if its budget is used up. The budget can be supplied by the user on the command line; otherwise a default (e.g., 40 seconds) is used. This leads to unrepeatable behavior, even among runs on the same platform, and obviously gives different outcomes across platforms. To make runs repeatable and platform independent, we introduced functions to *estimate* CPU

Table 1. Replication of selected `qxbf` data in SAT 2011, Table 1, using `qxbfCntrs`.

QBFEVAL-10: 568 formulas						
Preproc.: `qxbfCntrs`	*Solver*	*Solved*	*Time*	(*Preproc.*)	*SAT*	*UNSAT*
`sQueezeBF`+(ABST+SAT)	`depqbf`	434	238.24	(42.00)	201	233
SAT	`depqbf`	380	320.02	(8.34)	168	212
ABST+SAT	`depqbf`	377	321.84	(7.24)	167	210
ABST	`depqbf`	375	325.11	(2.84)	168	207
Preproc.: `qxbf`	*Solver*	*Solved*	*Time*	(*Preproc.*)	*SAT*	*UNSAT*
`sQueezeBF`+(ABST+SAT)	`depqbf`	434	239.84	(42.79)	201	233
SAT	`depqbf`	379	322.31	(7.17)	167	212
ABST+SAT	`depqbf`	378	323.19	(7.21)	167	211
ABST	`depqbf`	375	327.64	(3.33)	168	207

time based on various counters maintained in the program. The resulting program is called `qxbfCntrs`.

The user interface is unchanged; `qxbfCntrs` simply compares its (repeatable, platform independent) *estimated* CPU time to its budget. The estimation model was arrived at using linear regression, and was designed to give the same results as the published paper [LB11], or very close. The estimation model is not particularly accurate, but the budget amounts are only heuristics, so the overall performance might be about the same.

To validate this conjecture, we attempted to replicate parts of the published results for `qxbf`, using `qxbfCntrs` with all the same parameters. We chose those results that were most relevant for the topics of this paper. The replication is compared with the original in Table 1. For the replication to be meaningful, due to externally imposed time limits, it was carried out on the same platform, in the same environment, as the published table. Times shown in Table 1 are measured by the system; they are not estimates.

In addition, all counts in Tables 2 and 3 of the SAT 2011 paper were confirmed during replication. These tables analyze the preprocessor only, giving it 900 seconds, without internal budgets, so high correspondence is expected. Here, `qxbfCntrs` times were about 4% faster and one additional instance was completed in just under 900 seconds. This can be attributed to compiling with a newer compiler. One factor helping the close correspondence is that the limits imposed by the internal budget are often not reached.

In summary, we conclude that the functions for estimating time are adequate for producing useful repeatable experiments. For all remaining experiments we use `qxbfCntrs` for the baseline against which variations are compared, use the same estimation functions for monitoring against internal budgets, and drop the suffix `Cntrs` from here on.

The main goal of this research is to implement and evaluate 1-saturation for QBF. There are many preprocessing techniques known in the literature, so we are interested to know whether 1-saturation adds new capabilities, or

Table 2. Effect of pure-literal policies on `qxbf`, 420 preprocessed but unsolved instances.

	`qxbf`			*follow-on* `depqbf`		
	N Solved		average	*N Solved*		average
Pure-lit policy	true	false	seconds	true	false	seconds
exist. and univ.	0	24	76.50	162	134	380
none	0	24	42.11	162	133	380
only univ.	0	24	53.53	162	134	380

just finds mostly the same inferences and simplifications as other techniques. Therefore, our general approach is to apply a strong existing preprocessor to the raw benchmarks, then apply failed literals or 1-saturation to the result. We chose the recently reported **bloqqer** since it is open source [BSL11].

There are (at least) two distinct ways to apply preprocessing. The usual way is to give limited budgets to the preprocessor(s) with the idea that a complete solver will do the bulk of the work. Another choice is to give the preprocessor(s) all the resources and see how many instances they can solve completely. Ideally, for the latter approach, the preprocessor(s) would run to completion, not be stopped by time limits. This ideal is not completely far-fetched because preprocessors are expected to stop in polynomial time. Unfortunately, for some larger benchmarks, the time to completion is impractically long.

Our computing resource for Tables 2, 3, and Figure 2 is a pair of 48-core AMD Opteron 6174 computers running Linux with 180 GB of memory each and a 2.2 GHz clock, managed by SunGrid queueing software. Our tests show that this platform is about two times slower than than the platform used in [LB11] and for Table 1, which is a dedicated cluster running one job per two-core processor.

The principal benchmark suite for QBF currently is the 568 instances used for QBFEVAL-10. We follow this tradition. The preprocessor `bloqqer` solved 148 instances (62 true, 86 false), leaving 420. This initial run averaged 8.89 seconds per instance; all runs completed normally.

The main purpose of this study was to evaluate extended failed-literal analysis (EFL). This necessitated changing how pure literals were handled. To be sure any experimental differences are attributable to EFL, we checked the influence of changes only in pure-literal processing.

The published program `qxbf` uses pure literals, although it has a command-line switch to disable all pure literal processing. To incorporate 1-saturation, it is only necessary to disable *existential* pure literal processing. To measure the effect of each change in the procedure, we created `qxbf_noepure`, which has the same logic as `qxbf` except that existential pure literal processing is disabled. Table 2 shows the effect of disabling all pure literal processing and the effect of disabling existential pure literal processing. Indeed, the "only univ." row in Table 2 demonstrates UPUP results without 1-saturation.

The data suggests that the existential pure-literal rule did not help `qxbf` and the universal pure-literal rule helped a little. These rules have not been

Table 3. Additional completely solved instances among 420 instances that were preprocessed but unsolved by `bloqqer`. `qxbf` uses FL and both pure-literal rules. `eqxbf` uses EFL and the universal pure-literal rule.

	`bloqqer only`				`qxbf, bloqqer`				`eqxbf, bloqqer`			
	N Solved			avg.	*N Solved*			avg.	*N Solved*			avg.
Round	true	false	both	secs.	true	false	both	secs.	true	false	both	secs.
1	4	9	13	9	8	29	37	79	11	32	43	58
2	1	0	1	1	1	0	1	75	0	2	2	44
3	0	1	1	1	0	1	1	79	0	1	1	42
total	5	10	15	11	9	30	39	233	11	35	46	144

examined separately before. Empirical data shows that pure-literal rules are important for QDPLL solvers, but possibly that most of the contribution comes from the universal pure-literal rule (also a source of substantial overhead).

5.1 Complete Solutions with 1-Saturation

Table 3 compares the solving abilities of `bloqqer` alone, `qxbf` followed by `bloqqer`, and `eqxbf` followed by `bloqqer`. There are no time-outs in this table. The reason for following `qxbf` and `eqxbf` by `bloqqer` is that `qxbf` and `eqxbf` apply a single procedure and have no general simplification capabilities. In fact they can never detect that a formula is `true`. Our goal was to evaluate their solving capabilities in conjunction with other typical preprocessing techniques. They were run for several rounds in an attempt to reach a fixpoint. However, we discovered that `bloqqer` itself does not quickly reach a fixpoint, and stopped after three rounds.

By the criterion of complete solutions, we observe that extended failed literal analysis (EFL, `eqxbf`) produces moderate gains over failed literal analysis (FL, `qxbf`). We also confirm that FL produces about the same gains beyond the initial preprocessor as the 25 reported by Lonsing and Biere [LB11, Sec. 7]. We observed 24 additional solutions by `qxbf` alone. Overall, the data shows that 194 out of 568 QBFEVAL-10 benchmarks can be solved by preprocessing.

For completeness we ran `eqxbf` with all pure-literal processing disabled. The program slowed down slightly, but solved the same instances as `eqxbf` in round 1 of Table 3. This shows that super-sound preprocessing may be feasible, for applications in which preserving all tree models is important.

5.2 Timed Runs with 1-Saturation and a Complete Solver

This section evaluates whether extended failed literal analysis (EFL) makes the overall solving task faster. We established a total time limit of 1800 seconds, which is equivalent to about 900 seconds on the newest platforms. We ran `bloqqer` with adaptive command-line parameters so that it would not take too much of the 1800 seconds on very large benchmarks. The command-line parameters were set heuristically depending on the numbers of variables, clauses

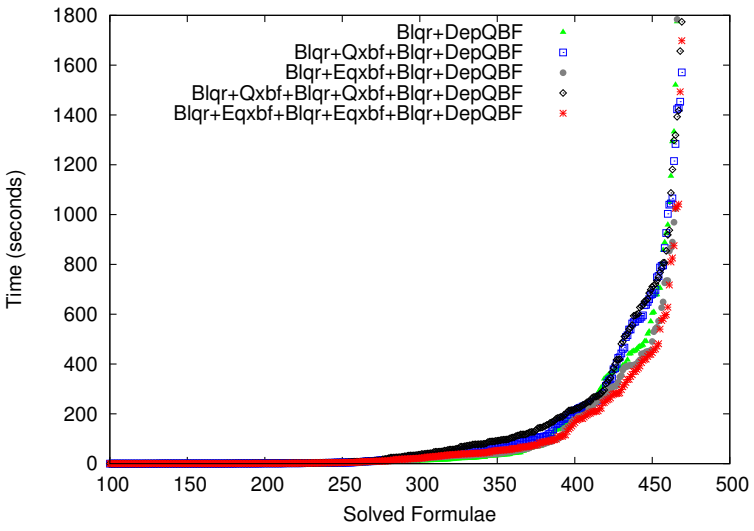

Fig. 2. Timed solving runs. (Larger figure at `http://www.cse.ucsc.edu/~avg/EFL`.)

and literals in the instance. No `bloqqer` run on an original QBFEVAL-10 instance took more than 140 seconds for the experiments in this section. After this common start on the 568 QBFEVAL-10 benchmarks, the solving attempts used three strategies. All three strategies used `depqbf` as the complete solver.

To be considered successful, the sum of all preprocessing times and the solving time had to be under 1800 seconds. The outcomes are summarized in Figure 2.

The first strategy simply runs `depqbf` on the preprocessed instances with a time limit of 1800 seconds. Results should be comparable to runs reported in [BSL11] with 900 seconds time limits.

The second strategy uses one or two rounds of `qxbf`, `bloqqer`, as described in Section 5.1, then runs `depqbf`. Recall that `qxbf` uses FL and both pure-literal rules. For this table `qxbf` is given a budget of 80 "estimated seconds" and is run with an external time limit of 1800 seconds. Time-outs by `qxbf` occurred in three instances during round 2. The time limit for `bloqqer` was 300 seconds, but it never came close to timing out.

The third strategy uses one or two rounds of `eqxbf`, `bloqqer`, as described in Section 5.1, then runs `depqbf`. Recall that `eqxbf` uses EFL and the universal pure-literal rule. The details are the same as for the second strategy, except that `eqxbf` timed out on only one instance during round 2.

In Figure 2 we observe that `eqxbf` (EFL), combined with `bloqqer` for one or two rounds, helps `depqbf` slightly more than `bloqqer` alone. This observation is confirmed using the *Careful Ranking* procedures reported in SAT-11 and used unofficially in that competition [VG11].

6 Conclusion

This paper presents theoretical analysis of the universal pure-literal rule in combination with formula abstraction and extended failed literal analysis. It shows that binary clauses can be safely derived. It further shows that the use of the existential pure-literal rule in combination with formula abstraction is generally unsafe. An incomplete forward reasoning procedure called UPUP was implemented and tested. Another result is that binary clauses can be super soundly derived if no pure-literal rules are used.

Experimental results suggest that EFL provides some improvement in overall solving performance on the QBFEVAL-10 benchmarks. The previous mark of 148 solved by preprocessing alone now stands at 191, although EFL does not get all the credit. Preliminary data indicates that EFL does almost as well without any pure-literal rules, which is important if preserving model trees is important on `true` QBFs. Future work should integrate EFL with a general preprocessor, such as `bloqqer`.

Acknowledgment We thank Craigslist Inc. for equipment donations that facilitated this research.

References

[BSL11] A. Biere, F. Lonsing, and M. Seidl. Blocked clause elimination for QBF. In *Proc. CADE (LNCS 6803*, pages 101–115, 2011.

[Fre93] J. W. Freeman. Failed literals in the Davis-Putnam procedure for SAT. In *DIMACS Challenge Workshop: Cliques, Coloring and Satisfiability*, 1993.

[GMN10] E. Giunchiglia, P. Marin, and M. Narizzano. sQueezeBF: An Effective Preprocessor for QBFs Based on Equivalence Reasoning. In *Proc. SAT (LNCS 6175)*, pages 85–98, 2010.

[GNT04] E. Giunchiglia, M. Narizzano, and A. Tacchella. Monotone literals and learning in QBF reasoning. In *Proc. CP (LNCS 3258)*, pages 260–273, 2004.

[GW00] J. F. Groote and J. P. Warners. The propositional formula checker Heer-Hugo. *J. Automated Reasoning*, 24(1):101–125, 2000.

[JBS+07] T. Jussila, A. Biere, C. Sinz, *et al.* A first step towards a unified proof checker for QBF. In *Proc. SAT (LNCS 4501)*, pages 201–214, 2007.

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

[KBL99] H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, 1999.

[KSGC10] W. Klieber, S. Sapra, S. Gao, and E. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *Proc. SAT, (LNCS 6175)*, pages 128–142, 2010.

[LB01] D. Le Berre. Exploiting the real power of unit propagation lookahead. In *IEEE/ASL LICS Satisfiability Workshop*, pages 59–80, 2001. Elsevier.

[LB11] Florian Lonsing and Armin Biere. Failed literal detection for QBF. In *Proc. SAT (LNCS 6695)*, pages 259–272, 2011.

[SB07] H. Samulowitz and F. Bacchus. Dynamically partitioning for solving QBF. In *Proc. SAT (LNCS 4501)*, pages 215–229, 2007.

[VG11] A. Van Gelder. Careful ranking of multiple solvers with timeouts and ties. In *Proc. SAT (LNCS 6695)*, pages 317–328, 2011.