Bounded Universal Expansion for Preprocessing QBF

Uwe Bubeck[1] and Hans Kleine Büning[2]

[1] International Graduate School
Dynamic Intelligent Systems,
Universität Paderborn,
33098 Paderborn (Germany)
bubeck@upb.de
[2] Department of Computer Science,
Universität Paderborn,
33098 Paderborn (Germany)
kbcsl@upb.de

Abstract. We present a new approach for preprocessing Quantified Boolean Formulas (*QBF*) in conjunctive normal form (*CNF*) by expanding a selection of universally quantified variables with bounded expansion costs. We describe a suitable selection strategy which exploits locality of universals and combines cost estimates with goal orientation by taking into account unit literals which might be obtained.
Furthermore, we investigate how Q-resolution can be integrated into this method. In particular, resolution is applied specifically to reduce the amount of copying necessary for universal expansion.
Experimental results demonstrate that our preprocessing can successfully improve the performance of state-of-the-art *QBF* solvers on well-known problems from the QBFLIB collection.

1 Introduction

Quantified Boolean Formulas (*QBF*) generalize propositional formulas by allowing variables to be quantified either existentially or universally, whereas all variables are implicitly existentially quantified in propositional logic. This enhancement makes *QBF* a concise and natural modeling language for problems in many areas, such as planning, scheduling or verification [13, 15], and many Boolean functions have compact representations in *QBF*.

On the other hand, however, determining the satisfiability of formulas in *QBF* is PSPACE-complete, which is assumed to be significantly harder than the NP-completeness of the propositional SAT problem. But continued research and technical advances have already enabled impressive progress [14] towards the goal of developing powerful *QBF* solvers suitable for practical use. Some of those state-of-the-art solvers (e.g. [10] and [18]) are extensions of the well-known DPLL backtracking search algorithm for propositional logic [8]. Several other techniques have also been successfully applied to *QBF* solving, such as symbolic skolemization [2] or resolution and quantifier expansion [4], to name only a few.

There are in particular two characteristics of *QBF* which make it so difficult to solve. The first is the fact that for every universal variable, the solver must consider both possible values the variable might have. This obviously affects the DPLL-based search algorithms, but also the other approaches like symbolic skolemization, because an existential variable y_i can be assigned different values depending on the value of a universal whose quantifier precedes the quantifier of y_i. That behavior leads to the second inherent characteristic of *QBF*: the variable ordering imposed by the nesting of the quantifiers must be respected when solving the formula. In fact, the order of the quantifiers matters so much that the complexity of the decision problem for *QBF* formulas in conjunctive normal form (*CNF*) is assumed to become more difficult with each alternation of quantifier blocks in the prefix, resulting in the so-called polynomial hierarchy [12].

QBF instances from various application domains typically have significantly less universal quantifiers than existentials. And if those formulas have multiple alternations of quantifiers, the universal blocks usually tend to be rather short. It therefore appears rewarding to tackle the problem of solving *QBF* formulas by getting rid of the universally quantified variables. After all, a universally quantified formula $\forall x\ \phi(x)$ is just an abbreviation for $\phi(0) \wedge \phi(1)$, where the matrix of the formula is duplicated for x being either 0 or 1. As explained later in more detail, special care has to be taken for existentials which depend on x: those have to be duplicated as well.

This expansion of universal quantifiers has been used successfully for *QBF* solving by Ayari and Basin in QUBOS [1] and in Biere's solver Quantor [4]. Both systems are based on the approach of ultimately expanding all universals and then solving the remaining purely existentially quantified formula with an ordinary SAT solver. In addition, Quantor can also eliminate existential variables by Q-resolution whenever this is cheaper than expansion.

Unfortunately, expanding many universals can quickly lead to rapid growth of the resulting formula. In this paper, we suggest an approach which does not involve eliminating all universals in the formula. Instead, we restrict ourselves to preprocessing *QBF* formulas in *CNF* form by eliminating certain universally quantified variables with bounded expansion costs before feeding the resulting formulas to an ordinary *QBF* solver. The method is based on the idea that we can probably make it significantly easier for the solver when we take out some specially selected cheap or particularly rewarding universals. On the other hand, we avoid the costs of expanding expensive universals which might each require copying almost the whole formula and trigger an exponential explosion. We present a suitable selection strategy which exploits locality of universals and combines cost estimates with goal orientation. Furthermore, we discuss how Q-resolution can be integrated into this method. In particular, we apply resolution specifically to reduce the amount of copying required in a subsequent universal expansion step. This adds another strategic element to our variable elimination procedure. We finish with an experimental evaluation and a conclusion with suggestions for further improvements.

The previous work most closely related to ours is Biere's resolve and expand method [4] as implemented in Quantor. The most obvious difference is that we do only preprocessing with selective expansion under bounded expansion costs. In addition, Quantor only chooses quantifiers from the innermost universal scope for expansion, and we generalize the idea by selecting universals from the whole prefix. To make this work, we use tighter cost estimates and add goal orientation by taking into account unit literals which might be obtained from the expansion. Another major difference is our use of Q-resolution. While Quantor attempts to balance resolution and expansion, we focus specifically on expansion and use resolution only as a strategic means of reducing the expansion costs. Notice that the solver QUBOS which was also mentioned above is very different from both our approach and Quantor. It appears to be geared towards non-*CNF* formulas or circuits and does not perform cost calculations, but just expands the universals in the given order starting with the innermost. Furthermore, QUBOS does not perform Q-resolution at all.

2 Preliminaries

A quantified Boolean formula $\Phi \in QBF$ in *prenex form* is a formula

$$\Phi = Q_1 v_1 ... Q_k v_k \, \phi(v_1, ..., v_k)$$

with quantifiers $Q_i \in \{\forall, \exists\}$ and a propositional formula $\phi(v_1, ..., v_k)$ over variables $v_1, ..., v_k$. We call $Q := Q_1 v_1 ... Q_k v_k$ the *prefix* and ϕ the *matrix* of Φ.

Unless mentioned otherwise, we assume that *QBF* formulas are always in prenex form. In addition, we assume that the matrix is in *conjunctive normal form* (*CNF*), where ϕ is a conjunction of clauses, with each clause being a disjunction of negated or non-negated variables (*literals*).

A universally quantified formula $\forall x \; \phi(x)$ is defined to be true if and only if $\phi(0)$ is true *and* $\phi(1)$ is true. Variables which are bound by universal quantifiers are called *universal variables* and are usually given the names $x_1, ..., x_n$. Similarly, an existentially quantified formula $\exists y \; \phi(y)$ is true iff $\phi(0)$ *or* $\phi(1)$. Variables in the scope of an existential quantifier are *existential variables* and have names $y_1, ..., y_m$. We write $\Phi = Q \; \phi(\mathbf{x}, \mathbf{y})$ or simply $\Phi = Q \; \phi$. Variables which are not bound by quantifiers are *free* variables. In this paper, we do not allow free variables in order to simplify the discussion. But we would like to point out that universal expansion is in fact an equivalence-preserving transformation when formulas with free variables are considered.

Without loss of generality, we require that no variable appears twice in Q (i.e. that all variable names are unique). We call successive quantifiers of the same kind in Q a *quantifier block* S. Blocks are defined to be maximal, such that subsequent blocks S_i and S_{i+1} are always labelled with different kinds of quantifiers. We usually write $\Phi = \forall X_1 \exists Y_1 ... \forall X_r \exists Y_r \; \phi(X_1, ..., X_r, Y_1, ..., Y_r)$ for a *QBF* formula with universal quantifier blocks $\forall X_i = \forall x_{i,1}, ..., x_{i,n_i}$ and existential blocks $\exists Y_i = \exists y_{i,1}, ..., y_{i,m_i}$.

According to their sequence in the prefix, quantifier blocks are ordered linearly $S_1 < ... < S_s$. We call S_s the innermost and S_1 the outermost block. The order of the quantifier blocks also induces a partial order on the variables. Let l_1 and l_2 be two literals in Φ, then we define $l_1 < l_2$ if the variable in l_1 occurs in a quantifier block which precedes the block in which the variable of l_2 appears. If both variables occur in the same block, the order of the literals is undefined.

With $|\Phi|$, we denote the size of a formula $\Phi = Q\ \phi \in QBF$, which we calculate by adding the numbers of literals in all clauses of ϕ. Based on [4], we also introduce notation to describe occurrences of variables and literals in the formula. Given a literal l, we let $o(l)$ denote the number of occurrences of l in a given formula, and $s(l)$ is defined to be the sum of the sizes of all clauses in which l occurs. We further extend the latter notation to sets V of variables by letting $s(V)$ be the sum of the sizes of all clauses in which a variable in V occurs. Consider the example formula $\Phi = \forall x_1 \exists y_1 \exists y_2\ (x_1 \vee \neg y_2) \wedge (\neg y_2 \vee y_1) \wedge \neg y_1$. Here, we have $o(y_1) = 1$, $s(y_1) = 2$ and $o(\neg y_2) = 2$, $s(\neg y_2) = 4$. Furthermore, $s(\{y_1, y_2\}) = 5$.

3 The Basic Preprocessing Algorithm

3.1 Universal Expansion

Consider a QBF formula

$$\Phi = \forall X_1 \exists Y_1 ... \forall X_r \exists Y_r\ \phi(X_1, ..., X_r, Y_1, ..., Y_r)$$

with universal quantifier blocks $\forall X_i = \forall x_{i,1}, ..., x_{i,n_i}$ and existential blocks $\exists Y_i = \exists y_{i,1}, ..., y_{i,m_i}$. In order to expand a universal variable $x_{i,j}$ from the i-th universal block, we have to generate two copies of the matrix ϕ, one where $x_{i,j}$ is 0, and one where $x_{i,j}$ is 1. Furthermore, we must take into account that the existentials in the subsequent blocks $Y_i, ..., Y_r$ depend on $x_{i,j}$ and can have different truth values assigned depending on whether $x_{i,j} = 0$ or $x_{i,j} = 1$. Accordingly, we have to duplicate these existentials to reflect that degree of freedom. We get the expanded formula

$$\begin{aligned}\Phi' = Q'\ &\phi(x_{1,1}, ..., x_{i,j-1}, 0, x_{i,j+1}, ..., x_{r,n_r}, Y_1, ..., Y_r)\ \wedge \\ &\phi(x_{1,1}, ..., x_{i,j-1}, 1, x_{i,j+1}, ..., x_{r,n_r}, Y_1, ..., Y_{i-1}, Y_i', ..., Y_r')\end{aligned}$$

with the new prefix Q' which we obtain from the original prefix when we drop $x_{i,j}$ and replace blocks $\exists Y_k$, $k = i, ..., r$, with $\exists Y_k, Y_k' = \exists y_{k,1}, ..., y_{k,m_k}, y_{k,1}', ..., y_{k,m_k}'$. Of course, not all clauses are affected by the expansion. An implementation of the algorithm will only have to touch clauses in which $x_{i,j}$ occurs and clauses which must be copied due to the renaming of the existentials.

Strictly adhering to this algorithm might produce lots of redundant copies when universal variables and their dependent existentials are only used locally, which is typical for linearizations of formulas in non-prenex form. Consider the example $\Phi = \forall x_1 \exists y_1 \forall x_2, x_3 \exists y_2, y_3\ \phi(x_1, y_1, x_2, y_2) \wedge \psi(x_1, y_1, x_3, y_3)$. The universal x_1 is used globally in the whole formula, but x_2 and x_3 and the dependent

existentials y_2 and y_3 are only used locally in subformulas ϕ and ψ. However, expanding x_2 with the given procedure would require us to duplicate not only y_2, but also y_3 and clauses in ψ with y_3 in them. Of course, this is redundant, since $\psi(x_1, y_1, x_3, y_3)$ and $\psi(x_1, y_1, x_3, y_3')$ are clearly satisfiability-equivalent.

What we need to do is take into account how variables are actually connected in common clauses. In [4], Biere introduces a suitable concept. His original formulation was not meant for expanding universals from the whole prefix, therefore we have to clarify that universals alone never propagate dependencies (because $\forall v\ (\phi \wedge \psi) \approx (\forall v\ \phi) \wedge (\forall v'\ \psi[v/v'])$). Our formulation is as follows:

We denote a variable v *locally connected* to another variable w if both occur in a common clause, and we write $v \sim w$. Given a universal variable x from the i-th universal quantifier block, we now define

$$\begin{aligned} D_x^{(0)} &:= \{y \in Y_i \cup ... \cup Y_r \mid y \sim x\} \\ D_x^{(k+1)} &:= \{y \in Y_i \cup ... \cup Y_r \mid y \sim y' \text{ for some } y' \in D_x^{(k)}\},\ k \geq 0 \\ D_x &:= \bigcup_k D_x^{(k)} \end{aligned}$$

We call the set D_x the *dependent existentials* of x. When expanding x, we only need to duplicate those existentials and the clauses in which they occur.

3.2 Bounded Expansion

Even when observing locality of universals, repeated application of universal expansion can easily lead to rapid formula growth. It is thus important for our preprocessing to impose strict bounds:

- a *global size limit* C_{global} places an upper bound on the size of the preprocessing output. Variables are only expanded while $|\Phi_{cur}| < C_{global} \cdot |\Phi|$, where Φ is the original input formula (after some initial simplifications as described below) and Φ_{cur} the current formula after some expansions.
  In our experiments, we found rather small values between 2 and 4 to work well without blowing up the formula too much.
- an *individual cost limit* C_{single} is enforced for each single expansion step. We only expand a universal x if the predicted expansion costs c_x (see Section 4) are bounded by $c_x \leq C_{single} \cdot |\Phi_{cur}|$, where Φ_{cur} is the current formula. The idea here is to expand the cheap universals and leave the expensive ones to the solver, since the solver might be able to handle them at lower costs with different strategies.
  We achieved best results with $C_{single} = 0.5$. If no universals have expansion costs below this threshold, the preprocessing will not do anything (except for the initial simplifications). It is due to this strategy of avoiding unfavorable steps that our preprocessing usually does not have noticeable negative effects on the performance of the *QBF* solver.

Listing 1 shows the basic structure of the preprocessor's main loop and illustrates where the bounds are applied. For completeness, we have also included the two occasions where Q-resolution is invoked. This is discussed in Section 5.

Listing 1: The Main Loop of the Preprocessor

```
preprocess (Φ, C_global, C_single) {
  simplify Φ;
  Φ_cur = Φ;
  while (|Φ_cur| < C_global · |Φ|) {
    resolve existentials with negative resolution costs;
    choose universal x with smallest predicted costs c_x;
    if ((x ≠ null) && (c_x ≤ C_single · |Φ_cur|)) {
      reduce dependencies D_x by resolution;
      expand x in Φ_cur;
      simplify Φ_cur;
    } else return Φ_cur;
  }
  return Φ_cur;
}
```

3.3 Simplifications

To reduce the actual costs of universal expansion, we have included the usual simplification rules: unit propagation, pure literal elimination, universal reduction, detection of dual binary clauses and subsumption checking. As they are standard techniques, we do not recall them here and refer the reader to [7, 4].

We apply the rules in a circular fashion where one simplification may trigger the application of another simplification rule, until we reach closure. Initially, we attempt to simplify the whole input formula. Later, we check for specific simplifications as necessary. For the initial simplification, universal reduction is probably the most important operation and allows us to assume for the remaining process that all clauses are cleansed from trailing universal variables which do not dominate any existentials in the same clause. Whenever we later modify clauses or add new ones, we will make sure they are cleansed as well.

In the beginning, we also perform a full subsumption check. Inside the main loop, however, we apply only the cheaper backward subsumption where old clauses are checked for being subsumed by newly generated clauses. The dual case where old clauses might subsume new clauses is not relevant to universal expansion, since expansion never produces longer clauses.

4 Selection Strategy

Making good choices for the universals to be expanded is crucial to the success of the preprocessing. Each expansion of a universal x produces expansion costs $c_x = |\phi'| - |\phi|$, where $|\phi|$ is the size of the matrix of the formula before the expansion and $|\phi'|$ the size of the matrix afterwards, so c_x indicates by how many literals the size of the formula will increase when expanding x. For unsimplified formulas, c_x may also be negative. Since we want to select the universals with the lowest expansion costs, we need a tight cost estimate for each universal in the formula.

4.1 Estimation Scheme

Given a *QBF* formula $\Phi = \forall X_1 \exists Y_1 ... \forall X_r \exists Y_r\ \phi$ with universal quantifier blocks $\forall X_i = \forall x_{i,1}, ..., x_{i,n_i}$ and existential blocks $\exists Y_i = \exists y_{i,1}, ..., y_{i,m_i}$, Quantor [4] estimates the costs of expanding a universal x from the innermost universal quantifier block X_r by considering all existentials in Y_r as dependent on x. Then all clauses in which an existential $y \in Y_r$ occurs need to be duplicated. Using the notation from Section 2, this means that $s(Y_r)$ literals must be added. Furthermore, clauses from the original matrix ϕ in which x occurs negatively are removed from $\phi(x/0)$, as well as clauses in $\phi(x/1)$ where x occurs positively. Finally, all occurrences of x in $\phi(x/0)$ and $\neg x$ in $\phi(x/1)$ are deleted. In total, Quantor's cost estimate is

$$c_x \leq s(Y_r) - s(\neg x) - s(x) - o(x) - o(\neg x)$$

4.2 Including Locality

In our approach, we do not want to restrict ourselves to the innermost universal quantifier block. We also want to be able to expand universals from quantifier blocks X_i with $i < r$. With the cost estimate given above, universals from further outside have higher expansion costs, because we would need to add $s(Y_i \cup ... \cup Y_r)$. Accordingly, choosing universals further outside can only be rewarding if additional factors are considered. For example, it might happen that the expansion of a universal further outside produces valuable unit literals. Or we might encounter the linearization of a non-prenex formula like $\Phi = \forall x_1 \exists y_1\ ((\forall x_2 \exists y_2 \forall x_3 \exists y_3\ \phi)\ \wedge\ (\forall x'_2 \exists y'_2 \forall x'_3 \exists y'_3\ \psi))$. If ϕ and ψ are not balanced in terms of size or difficulty, it may very well make sense to expand, e.g. x_2 before x'_3, although x_2 will be further outside than x'_3 in the linearized prefix.

As described in [4], Quantor's scheduling cannot take into account the locality of universals at this point due to performance considerations. It only uses locality during the actual expansion after a particular universal has already been selected.

Fortunately, our preprocessing scenario requires less frequent scheduling in comparison to a full solver like Quantor. On the one hand, this is due to the fact that we do not have to schedule resolutions. On the other hand, the bounds C_{global} and C_{single} are so tight that we will only expand a rather limited number of universals, therefore we execute much less expansion cycles. Accordingly, we can spend more time on selecting the variables and afford to actually compute the sets D_{x_i} of dependent existentials for the universals x_i in each expansion cycle. This needs time $O(e \cdot m \cdot |\Phi|)$, where e is the number of expansion cycles (iterations of the preprocessor's main loop) and m the number of universals in Φ. Our experiments show that this is still feasible: the total time spent for preprocessing is typically only a small fraction of the time required for the successive run of the solver. Furthermore, we assume that novel data structures like Benedetti's quantifier trees [3] might be applied here with great benefit in future versions of our preprocessor.

Let $D_x \subseteq Y_i \cup ... \cup Y_r$ be the existentials which depend on x. Then we have

$$c_x \leq s(D_x) - s(\neg x) - s(x) - o(x) - o(\neg x)$$

4.3 Goal Orientation

Expanding variables just because it is cheap to do so is a method without much foresight. It turns out that we can further improve our selection strategy by taking into consideration not only costs, but also goals which we might reach by expanding certain universals. A rewarding goal in solving satisfiability problems is to obtain unit literals. Propagating them helps keeping clauses short and might lead to discovering even more unit literals. This is in particular true for formulas with *2-CNF* subformulas, which might just collapse. Consider the following example:

$$\Phi = \forall x_1, x_2 \exists y_1, y_2 \ (x_1 \vee y_1) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee \neg y_1 \vee \neg y_2)$$

The universals are pure variables, but we ignore this here for simplicity (perhaps, Φ is embedded into a larger formula). Then $D_{x_1} = D_{x_2} = \{y_1, y_2\}$ and $c_{x_1} = 7 - 2 - 1 = 4$ and $c_{x_2} = 7 - 3 - 1 = 3$, so we expand x_2. After simplifying by removing pure existential literals, we obtain the new matrix $\Phi' = \forall x_1 \exists y_1, y_2 \ (x_1 \vee y_1) \wedge (\neg y_1 \vee y_2) \wedge (\neg y_1 \vee \neg y_2)$ (there are no renamed existentials, because they were simplified away). Had we expanded x_1 instead, the whole matrix would have collapsed to the empty clause after propagating the unit literals y_1 and y_2 when $x = 0$ and removing the pure existentials when $x = 1$. Of course, the same happens when we continue on Φ'. But since our preprocessing only expands a limited number of universals, we might stop after x_2 and miss out on this.

We did not want to have separate measures for expansion costs and benefits, because it would be necessary to balance them somehow. Fortunately, unit literals which are immediately obtained from expanding a universal also have a direct impact on the expansion costs of that universal, as seen in the example. We can therefore simply subtract from the expansion costs the reductions through immediate unit literals, making our cost estimates even tighter and allowing us to continue using costs as our single measure for choosing universals.

In order to do so, we need to know those unit literals. In each iteration of the preprocessor's main loop, we have to perform for each universal variable x a complete unit propagation under the assumption that $x = 0$, and then under the assumption $x = 1$. Since unit propagation can be performed in linear time, this needs $O(e \cdot m \cdot |\Phi|)$, where e is the number of expansion cycles (iterations) and m the number of universals in Φ. As with the calculation of the variable dependencies above, we claim this is still feasible due to the small values of e.

Let U_0 be the unit literals induced by assuming $x = 0$ and U_1 the units when $x = 1$. Then it might happen that U_0 (or analogously U_1) contains unit literals $l_i = \pm y_i$ with existentials y_i whose quantifier precedes the quantifier of x. But since the y_i do not depend on x, such l_i must also be unit literals when $x = 1$. That means we can propagate those units immediately (and remove them from U_0), even without actually expanding x (similar to [16]). Obtaining units in that way without expansion is a small additional benefit of our unit calculations.

Now let $s_{U_0 \setminus \pm x}$ be the sum of the sizes of all clauses in which a unit literal from U_0 occurs, but not $\pm x$ (we do not want to count those clauses twice). Those clauses are removed from the expansion. Furthermore, let $o_{\neg U_0 \setminus \neg x}$ be the number

of clauses in which the negation of a unit literal from U_0 occurs, but not $\neg x$. In those clauses, the negation of the unit literal will be removed. With $s_{U_1 \setminus \pm x}$ and $o_{\neg U_1 \setminus x}$ defined analogously, our cost estimate c_x is finally given as

$$c_x \leq s(D_x) - s_{U_0 \setminus \pm x} - s_{U_1 \setminus \pm x} - o_{\neg U_0 \setminus \neg x} - o_{\neg U_1 \setminus x} - s(\neg x) - s(x) - o(x) - o(\neg x)$$

5 Integrating Q-Resolution

Q-Resolution [11] extends the concept of propositional resolution to *QBF*. We can use it to eliminate an existential variable y in a formula $\Phi \in QBF$ by performing all possible resolutions on y. We can then drop the clauses in which y occurs positively or negatively and replace them with the set of resolvents after performing universal reduction. One problem with this approach is that it may produce large clauses. An even more serious problem is the huge number of resolvents which might be generated when an existential occurs frequently in both phases and we must resolve all positive occurrences with all negative ones.

Accordingly, our preprocessing focuses mainly on universal expansion. Nevertheless, a limited amount of resolution has proven helpful as well. There are two cases when we will apply resolution:

1. Whenever we can eliminate existentials without increasing the formula size.
2. If we can use resolution specifically to reduce costs of a scheduled expansion.

In order to estimate the costs c_y of eliminating an existential y by resolution, we use the upper bound given in [4]:

$$c_y \;\leq\; o(\neg y) \cdot (s(y) - o(y)) \;+\; o(y) \cdot (s(\neg y) - o(\neg y)) \;-\; (s(y) + s(\neg y))$$

At the beginning of each iteration through the preprocessor's main loop, we check whether there are existentials y_i for which this cost estimate c_{y_i} is negative, so that we can be sure not to increase the size of the formula. We then choose the cheapest such existential, i.e. the one for which the cost estimate is the most negative, and eliminate it by resolution. The process is repeated as long as there are existentials with negative cost estimates.

Performing those resolutions before a universal expansion cycle is like a general cleanup that reduces the number of existentials we have to consider and to copy. But we also suggest a more specific application of resolution which only takes place after we have chosen a particular universal x for expansion. Our goal is to reduce its expansion costs c_x. A quick glance at the cost estimates from the last section shows that there are basically two components which determine the value of c_x: the occurrences of $\pm x$ itself and the occurrences of dependent existentials. We are now going to apply resolution to attack the latter.

The idea is to resolve only on dependent existentials in D_x immediately before expanding x. Eliminating such an $y \in D_x$ yields a double benefit, because we do not only get rid of y itself, but also of its soon-to-be-created copy y'. In addition, we may also save copying some clauses during the following expansion. For example, a clause $(y \vee y_2)$ with $y \in D_x$ and $y_2 \notin D_x$ must be duplicated

when x is expanded, but when we resolve on y with $(\neg y \vee y_3)$ and $y_3 \notin D_x$ before expanding x, the resolvent $(y_2 \vee y_3)$ does not need copying, since both literals do not depend on x. Of course, resolution usually produces many resolvents, some of which probably still require copying. In our example, the formula might also contain a clause $(\neg y \vee y_4)$ with $y_4 \in D_x$, so that we obtain a second resolvent $(y_2 \vee y_4)$ which is still dependent on x.

Let δ be an estimate of the average fraction of resolvents which must be duplicated ($0 \leq \delta \leq 1$). Then we can estimate the costs $c_{y|x}$ of resolving an existential $y \in D_x$ before x is expanded:

$$c_{y|x} \;\approx\; (1+\delta) \;\cdot\; (o(\neg y) \cdot (s(y) - o(y)) + o(y) \cdot (s(\neg y) - o(\neg y))) \;-\; 2 \;\cdot\; (s(y) + s(\neg y))$$

We obtain this estimate from the upper bound for resolution given above. The factor 2 reflects the assumption that each clause in which y occurs would have been copied in the subsequent universal expansion (for simplicity, we do not take into account that y and x might occur in common clauses). The factor $(1 + \delta)$ indicates the costs of duplicating in the expansion a portion δ of the resolvents. In our experiments, we found $\delta = 0.5$ to work well when we resolve away all existentials $y \in D_x$ for which the cost estimate $c_{y|x}$ is negative before actually expanding x.

Resolution also reveals an interesting special case. Consider a scenario where we have a universal x_j and two sets D'_{x_j} and D''_{x_j} of existentials which are locally connected to x_j. Further assume that one of those existentials, say $\tilde{y}$, has the property that it constitutes the only link which propagates the local connectivity from x_j and D'_{x_j} on the one hand to D''_{x_j} on the other hand.

Can we destroy that link to make the existentials in D''_{x_j} independent from x_j? If $\tilde{y}$ occurs positively in clauses with existentials from D'_{x_j} and negatively in clauses with existentials from D''_{x_j}, resolving on $\tilde{y}$ will directly link D'_{x_j} and D''_{x_j}, so nothing is gained in this case. But assume $\tilde{y}$ only occurs positively with both D'_{x_j} and D''_{x_j}. Also assume that all negative occurrences of $\tilde{y}$ are in clauses with universals other than x_j and existentials which do not depend on x_j. Now we can resolve away $\tilde{y}$, and the existentials in D'_{x_j} and D''_{x_j} will not be connected anymore, since $\tilde{y}$ has been replaced with variables which do not propagate the dependency.

In this scenario, the special property is that we have an existential y in $D_{x_j} = D'_{x_j} \cup D''_{x_j}$ where one phase of y occurs only in clauses with variables $v \notin D_{x_j}$ and $v \neq x_j$. A closer investigation of this special case reveals that we do not need to perform the actual resolution. Instead, we can simply remove y from D_{x_j}, because it does in fact not depend on x_j: assume that with fixed assignments to $x_1, ..., x_{j-1}$, a given formula Φ is satisfiable if y is assigned different values in the two cases $x_j = 0$ and $x_j = 1$, i.e. $y = \epsilon$ for $x_j = 0$ and $y = \neg\epsilon$ for $x_j = 1$. Without loss of generality, assume that $\neg y$ is the phase of y which occurs only in clauses with variables $v \notin D_{x_j}$ and $v \neq x_j$. Then none of those clauses contains a variable which depends on x_j, yet those clauses remain satisfied when y flips from ϵ to $\neg\epsilon$ as x_j changes. That means those clauses are true regardless of the value of y. Then we can choose $y = 1$ for both $x_j = 0$ and $x_j = 1$, and all clauses with positive y will be satisfied as well, which means the whole formula

is satisfiable. With the obvious argument that if Φ is unsatisfiable when we allow different values for y depending on x_j, this also implies the unsatisfiability of Φ when y must have the same value for $x_j = 0$ and $x_j = 1$, we have the following theorem:

Theorem 1. *Given $\Phi \in QBF$, let x be a universal and y be an existential variable in the scope of x where one phase of y only occurs in clauses with $v \notin D_x$ and $v \neq x$. Then universal expansion of x does not need to duplicate the variable y.*

For performance reasons, our implementation does not check this condition while computing the dependency sets during the scheduling of the expansions, but only prior to executing a scheduled expansion.

6 Implementation and Experiments

We have implemented our preprocessing approach in Java on top of our existing logic framework ProverBox [5, 6]. Using the framework's data structures and basic algorithms has allowed us to quickly build a working preprocessor, although we sacrifice some performance for genericity, as the primary goal of the framework is to integrate different logics and different theorem proving algorithms. In addition, our preprocessor itself it not optimized yet.

The choice of the two bounds C_{global} and C_{single} which control the amount of preprocessing performed (see Section 3.2) obviously has a large impact on the performance of our preprocessor. For space considerations, we do not compare different parameter settings against each other. Instead, we have chosen one successful setting for all of the following experiments.

For the global size limit C_{global} which determines how much larger than the original formula the preprocessed formula may be, we found a value of 2 to perform best. We observed that when formulas are growing, the solver performance is often increasingly dominated by the sheer formula size rather than its complexity. By keeping C_{global} quite low, we try to avoid this effect.

While restricting the resulting formula to about twice the size of the input formula means that in the worst case, only $1 - 2$ universals may be expanded, we can typically expand up to $5 - 10$ of them. For various QBFLIB formulas, the number of expanded universals is even higher: for example, it is usually around 30 for the *ASP* problems, and in some *Adder* formulas, we can expand up to 130 universals. Of course, this number does not include universals merely declared in the prefix, but not used at all in the formula.

The individual cost limit C_{single} determines how expensive a single expansion can be. We achieved best overall results with a value of 0.5, where each expansion is allowed to copy at most half of the current formula. Universals with higher expansion costs are probably better handled by the *QBF* solver itself.

We have conducted our experiments with two state-of-the-art *QBF* solvers, sKizzo [2] and SQBF [17], on an Athlon64 3400+ with 2GB RAM running Windows XP/Cygwin and Java 6. Each solver has been executed under Cygwin in

its latest publicly available release and with default parameters. We have applied both solvers, each with and without preprocessing, on a selection of 12 families of benchmarks with a total of 688 instances from the QBFLIB collection [9]. We tried to choose benchmark families of such a difficulty level that the solvers could solve most, but not all formulas of a family within a time limit for each formula of 300 seconds. As expected, when the preprocessor was used, the time limit and the time recorded were for running both the preprocessor and the solver.

As usual, if an instance cannot be solved, e.g. due to a timeout or an out-of-memory condition, it is counted as the timeout value. To make the experiments less time-consuming, we have performed them in a give-up mode where a (sub)family of formulas is quit whenever we encounter an instance solved by neither the solver nor the solver with preprocessor. For example, neither sKizzo itself nor sKizzo with preprocessor can solve the instance *adder-14-sat*, so we skip *adder-16-sat* (without counting it as timeout) and continue with *adder-2-unsat*. Of course, this requires that the instances are approximately sorted in order of ascending difficulty. Where this was not already the case by default, we grouped formulas in obvious subfamilies (e.g. *adder-sat*, *Adder2-sat*, ... or *cnt*, *cnt-r*, ...).

An overview of the results is given in Table 1. It provides for each benchmark family the number of instances solved and the time (in seconds) required by the original solver as well as the combination of preprocessor and solver (sKizzo+pre resp. SQBF+pre). We can observe that both solvers show noticeable overall gains from the preprocessing: sKizzo+pre could solve $13,5\%$ more problems in $47,9\%$ of the time originally recorded, and SQBF+pre did even $49,5\%$ more problems in $23,4\%$ of the time. This appears to back our basic assumption that *QBF* solvers can indeed benefit from selectively removing universals beforehand.

Table 1: Benchmark Results

		sKizzo		sKizzo+pre		SQBF		SQBF+pre	
Benchmark Family	#inst	solved	time	solved	time	solved	time	solved	time
Adder	32	13	1,568	13	1,655	4	1,203	4	1,201
ASP	40	26	5,181	**40**	1,068	0	12,000	**40**	1,030
Blocks	13	9	368	9	371	10	309	10	323
Connect3 cf_3_3*	21	**7**	381	6	610	11	1,816	**16**	474
Counter	88	52	3,939	**56**	2,563	37	3,101	**38**	2,732
CounterFactual ncf_4*	320	108	1,540	**109**	1,295	87	14,819	**124**	1,244
Evader-Pursuer 4x4-log	7	1	320	1	319	7	13	7	14
k_branch_n	21	**6**	661	5	698	4	455	4	356
k_path_n	21	21	164	21	167	5	1063	**7**	500
RobotsD2 *.2, *.4, *.8	29	10	6,180	**29**	94	20	2,897	**29**	110
Sorting_networks	84	24	1,658	**26**	1,499	9	1,240	**11**	879
Szymanski	12	4	335	4	348	0	300	0	300
Total	688	281	22,295	319	10,687	194	39,216	290	9,163

For a closer look at the results, we have marked the numbers of solved instances in bold whenever one contestant could solve more problems than the other. We observe that there are only two cases (*k_branch_n* and *Connect3* with sKizzo+preprocessing) where the number of solved instances is lower by one with preprocessing. This seems to justify our hypothesis that enforcing tight bounds on the expansion can largely prevent negative effects. On the other hand, there are various families where preprocessing helped solve more problems.

7 Conclusion

Making it easier for *QBF* solvers by selectively removing universal variables in a preprocessing step - a simple yet intriguing idea. We have successfully realized it on the basis of the proven universal expansion method, which we have made bounded and more general by choosing universals from the whole prefix while giving consideration to the locality of variables. In addition, we have added an element of goal orientation to the variable selection by rewarding the generation of unit literals. We have also integrated Q-resolution into our approach and shown how it can be applied specifically to reduce the amount of copying necessary for universal expansion. In concluding experiments with two state-of-the-art solvers on QBFLIB problems, our preprocessing showed noticeable performance gains.

In the future, we would like to evaluate the inclusion of further strategic elements into our variable selection, such as attempting to eliminate complete universal quantifier blocks if they are small, or giving preference to universals that appear in short clauses, or trying to make local areas of the formula completely free of universals. In addition, we will attempt to further optimize our implementation.

References

[1] A. Ayari and D. Basin. *QUBOS: Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers.* Proc. 4th Intl. Conf. on Formal Methods in Computer-Aided Design (FMCAD'02). Springer LNCS 2517, 2002.

[2] M. Benedetti. *Evaluating QBFs via Symbolic Skolemization.* Proc. 11th Intl. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04). Springer LNCS 3452, 2005.

[3] M. Benedetti. *Quantifier Trees for QBFs.* Proc. 8th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'05). Springer LNCS 3569, 2005.

[4] A. Biere. *Resolve and Expand.* Proc. 7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04). Springer LNCS 3542, 2005.

[5] U. Bubeck. *Design of a Modular Platform for Automated Theorem Proving in Multiple Logics.* M.S. Thesis, San Diego State University, 2003.

[6] U. Bubeck. *ProverBox Automated Reasoning Environment.* Website `http://www.ub-net.de/cms/proverbox.html`, 2006.

[7] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. *An Algorithm to Evaluate Quantified Boolean Formulae and Its Experimental Evaluation.* Journal of Automated Reasoning, 28(2):101–142, 2002.

[8] M. Davis, G. Logemann, and D. Loveland. *A machine program for theorem-proving.* Communications of the ACM, 5(7):394–397, 1962.

[9] E. Giunchiglia, M. Narizzano, and A. Tacchella. *Quantified Boolean Formulas satisfiability library (QBFLIB).* Website `http://www.qbflib.org`, 2001.

[10] E. Giunchiglia, M. Narizzano, and A. Tacchella. *QUBE: A system for deciding quantified boolean formulas satisfiability.* Proc. 1st Intl. Joint Conf. on Automated Reasoning (IJCAR'01). Springer LNCS 2083, 2001.

[11] H. Kleine Büning, M. Karpinski, and A. Flögel. *Resolution for Quantified Boolean Formulas.* Information and Computation, 117(1):12–18, 1995.

[12] A. Meyer and L. Stockmeyer. *The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space.* Proc. 13th Symp. on Switching and Automata Theory, 1972.

[13] M. Mneimneh and K. Sakallah. *Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution.* Proc. 6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03). Springer LNCS 2919, 2004.

[14] M. Narizzano, L. Pulina, and A. Tacchella. *Report of the Third QBF Solvers Evaluation.* Journal of Satisfiability, Boolean Modeling and Computation, 2:145–164, 2006.

[15] J. Rintanen. *Constructing Conditional Plans by a Theorem-Prover.* Journal of Artificial Intelligence Research, 10:323–352, 1999.

[16] J. Rintanen. *Improvements to the evaluation of quantified Boolean formulae.* Proc. 16th Intl. Joint Conf. on Artificial Intelligence (IJCAI'99). Morgan Kaufmann Publishers, 1999.

[17] H. Samulowitz and F. Bacchus. *Using SAT in QBF.* Proc. 11th Intl. Conf. on Principles and Practice of Constraint Programming. Springer LNCS 3709, 2005.

[18] L. Zhang and S. Malik. *Towards Symmetric Treatment of Conflicts and Satisfaction in Quantified Boolean Satisfiability Solver.* Proc. 8th Intl. Conf. on Principles and Practice of Constraint Programming (CP'02), 2002.