Introduction to *QSpace*

Dan Pehoushek

danpehous@yahoo.com

Abstract

Given a boolean expression P, and an ordering of the n variables of P, there exists another boolean expression Q, on n variables, which exactly encodes, as satisfying assignments, all valid quantified sentences of P.

Q is always a monotone boolean function, answering All qbf sentences of P. Once the expression Q has been found, then each of P's 2^n QBF sentences is decidable simply by evaluating an assignment of Q. That is, checking a QBF of P reduces to checking an assignment of Q. Every P with a specific ordering on the variables maps to a specific monotone function Q. The problem of finding or constructing the expression Q, in Conjunctive Normal Form (cnf), for a given P, and order of the variables, is the canonical problem in $QSpace$. While it is expected that other PSpace-complete problems have QSpace analogs, this has not yet been explored.

In this paper, special and general methods for constructing the expression Q are given. The $\#P = \#Q$ theorem, the initial motivation for $QSpace$, is presented. The existence of Q for every satisfiable P is shown, along with some of Q's properties. Two polynomially easy special cases are described. The first special case shows that Monotone Quantified Boolean Formula is linearly decidable, a simple fact that has not been found in the literature. The second special case is an $O(n^2)$ method for representing all valid quantifications of 2-cnfs, via a simple mapping from any 2-cnf to a monotone 2-cnf. This special case may be considered a generalization of the linear method for validating a 2-qbf expression.

Expressions with small tree-width appear to have small Q expressions, but no true theorem to that effect has been proven.

A general method for producing the Q expression is given. The general method involves a "tree walk" over all satisfying assignments of P, so it is exponential. A case which is hard, because Q in cnf is simply too large, is also described.

The hard case, because of the size of Q, provides empirical, constructive evidence that $QSpace > NP$. Approximations to potentially exponential sized Q expressions are mentioned. Some reasoning and a possible path towards showing $PSpace > NP$, constructively, as opened by $QSpace$, is described.

As time allows, a description of the uses of $QSpace$ in debugging qtwiddle, a qbf solving platform, will be described. Currently, qtwiddle makes 180 errors out of close to 100 million test cases.

1 Introduction

This paper introduces the new problem class $QSpace$. The class arises from studying the $\#P = \#Q$ theorem, first proposed by the author, and proven by Ranan Fraer, on comp.theory, in July, 1997. That theorem exposes the close relationship among problems in $\#P$, $PSpace$, and NP. The theorem equates the number of satisfying assignments to a boolean expression to the number of valid quantified sentences of the expression.

In section 2, the $\#P = \#Q$ theorem and Fraer's elegant proof are given. In section 3, the existence and properties of an unquantified expression Q, which encodes all valid quantifications of another expression P, are discussed.

Section 4 presents two special cases (monotone and 2-cnf) in which it is easy, (in the first case, trivial), to construct Q. Section 5 presents a general method for constructing Q. Section 6 presents a hard case for constructing Q, because $|Q|$ (the size of Q) is large. Empirical evidence that the hard case produces large Q expressions is given, showing (but not quite proving) that $QSpace > NP$. Speculation on how to use $QSpace$ to show that $PSpace > NP$ (and therefore, $PSpace > P$), is discussed in the last section.

The ideas presented in this paper present a simplification of the complexity hierarchy. QBF is reduced to assignment evaluation, no matter how many alternations in the quantification. There is a significant possibility for constructively showing $PSpace > NP$ (and $PSpace > P$) indirectly, through the use of $QSpace$. The expression Q is essentially a (often large) linear time program for answering all QBF questions about P. The goal of this paper is to present the $QSpace$ ideas clearly and simply, to give others access to the potential of Q constructions for checking QBF solvers.

2 $\#P = \#Q$

The Satisfiability problem [GJ 84] is the canonical NP-complete problem: Given a boolean expression $P(x_1, ..., x_n)$, over n variables, does P have a satisfying assignment?

The problem of counting the number of satisfying assignments to a boolean expression is the canonical $\#P$-complete problem ($\#P$ is pronounced "number P". See [GJ 84]): Given a boolean expression $P(x_1, ..., x_n)$, over n variables, how many satisfying assignments are there, out of 2^n?

The Quantified Boolean Formula [GJ 84] problem is the canonical $PSpace$-complete problem: Given a sentence $S = q_1 q_2 ... q_n P(x_1, x_2, x_n)$, where P is a boolean expression over n variables, and the q_j's are quantifiers (q_j is either $\forall x_j$ or $\exists x_j$), is S valid?

When the q_j's are specific quantifiers, the sentence S is a QBF. In later sections, a boolean expression will be constructed that uses variables $q_1 ... q_n$; that expression will encode exactly, as satisfying assignments, all valid quantifications of P.

For a given variable ordering of an expression P, there are 2^n possible quantifications (2^n different quantified sentences), some true, the rest not. When a particular quantified sentence of P is true, the prefix shall be called a "valid quantification" of P. The problem of counting all valid quantifications of a boolean expression is called $\#Q$ (pronounced "number Q").

The $\#P = \#Q$ theorem shows that the two classes, $\#P$ and $\#Q$, are equivalent. Since they are equivalent, there is no new $\#Q$-complete problem class, although the descriptions of $\#P$ and $\#Q$ are superficially quite different, as one counts valid assignments of an expression, and the other counts valid QBFs.

Theorem 1 *Given a variable ordering, the number of satisfying assignments ($\#P$) to a boolean expression P is equal to the number of valid quantifications ($\#Q$) of P.*

Some simple cases illustrate the truth of the theorem. When P is a tautology, there are 2^n satisfying assignments, and all 2^n quantifications are valid. When P is unsatisfiable, there are no satisfying assignments and no valid quantifications. When P has a unique solution, then there is one valid quantification, where all the quantifiers are existential. And when the formula P has exactly 1 satisfying assignment, then there is only a single valid quantification, where all quantifiers are existential.

Fraer's proof is a linear induction on n, using very basic arithmetic operations, plus set union and intersection. Later, the general method for constructing Q will rely heavily on the inductive step, using set union and intersection to modify a tree.

Proof:

Base case: $n = 0$

As there are no variables there is only one (empty) assignment, and only one (empty) quantification. Both, the assignment and the quantification are valid if and only if $P()$ is equal to true.

Inductive case:

Supposing that the property is true for $n - 1$ we prove that it is still true for n, with $n > 0$.

Consider then the two following formulas on $n - 1$ variables obtained by setting $x_1 = true$ and respectively $x_1 = false$:

$$P_t(x_2, ..., x_n) = P(true, x_2, ..., x_n) \quad (1)$$

$$P_f(x_2, ..., x_n) = P(false, x_2, ..., x_n) \quad (2)$$

The number of valid assignments of P is then given by:

$$\#P = \#P_t + \#P_f = \#Q_t + \#Q_f \quad (3)$$

The number $\#Q$ of valid quantifications of P can be counted in the following way:

$\forall x_1 q_2 ... q_n$ is a valid quantification of P if and only if $q_2 ... q_n$ is a valid quantification of both P_t and P_f. This makes $\#(Q_t \cap Q_f)$ valid quantifications.

$\exists x_1 q_2 ... q_n$ is a valid quantification of P if and only if $q_2...q_n$ is a valid quantification of P_t or P_f. This makes $\#(Q_t \cup Q_f)$ valid quantifications.

$$\#(Q_t \cup Q_f) = \#Q_t + \#Q_f - \#(Q_t \cap Q_f) \tag{4}$$

$$\#Q = \#(Q_t \cap Q_f) + \#(Q_t \cup Q_f) = \#Q_t + \#Q_f \tag{5}$$

$$\#P = \#Q \tag{6}$$

A corrollary to the theorem is that, regardless of the ordering of the variables, there will be exactly $\#P$ valid quantifications. Switching the order of two variables can change which quantifications are valid, but it does not change how many are valid.

3 The expression Q

The classes NP and $PSpace$ have well-known boolean member problems, called Satisfiability and Quantified Boolean Formula. The class called $QSpace$ is presented by describing a member problem, called The Construction of Q. First, a QBF sentence S has the form:

$$S = q1q2...qnP(x1, ..., xn) \tag{7}$$

where P is a boolean formula, and the q_j are quantifiers, either $\forall x_j$, or $\exists x_j$. The QBF question is, is S valid?

By treating the quantifiers, $q_1...q_n$, as boolean variables, a boolean algebra can be constructed. The valid quantifications of a particular P can be separated from the invalid quantifications. Specifically, if there are any valid quantifications, a boolean expression that encodes exactly those valid quantifications can be constructed. That expression, on the variables $q_1...q_n$, in conjunctive normal form, is called Q. Each clause of a monotone Q expression encodes a particular constraint on $\forall$ quantifiers, stating that the variables of the clause cannot all simultaneously be $\forall$.

So, the canonical member problem in $QSpace$ is to construct the Q expression, exactly encoding the valid quantifications of P, in conjunctive normal form. Just as there are $PSpace$-complete problems other than QBF, there should be other $QSpace$-complete problems besides the Construction of Q. Finding and describing them is beyond the scope of this paper. In this section, the reasons for choosing cnf for the expression Q are outlined.

Later, with the general construction, it will be proven that Q on n variables is a monotone expression, representable without negation, iff P is satisfiable. There are several ways to deduce that Q is monotone. For didactic purposes, here's one way: If a particular quantification is valid, and it has some set U of variables quantified with $\forall$, then any "smaller" quantification, with only a subset of U quantified with $\forall$, must also be valid.

The following is a small example of a P and a Q, using DIMACS format for cnf expressions. In DIMACS form, a line beginning with "c" is a comment; a line with "p cnf n m" begins an n variable m clause conjunctive normal form expression. The next m lines are the disjunctions of the conjunction. The "-" sign indicates boolean negation. The variables are numbered $1...n$.

```
c P in CNF form
p cnf 5 4
    -1     3
     2    -4
    -3     4
     5

c Q in CNF form.
c 5 variables, 7 clauses
c Clause sizes:   1: 1  2: 6
p cnf 5 7
     1     2
     1     3
     1     4
     2     3
     2     4
     3     4
     5
```

The semantics of the P expression are normal, where the variables represented by $1...n$ take on the values $0/1$, $false/true$. The expression, when given an assignment, evaluates to $false/true$.

The semantics of the Q variables are extended slightly. First of all, there are no "-" signs in Q, only because the monotone function Q is encoded simply. Secondly, in addition to the normal boolean semantics of $0/1$ and $false/true$, we add $\forall/\exists$. "1 2" means that not both 1 and 2 can be $\forall$ at the same time, in any valid quantified sentence of P; at least one of the two variables must be existentially quantified. In any case, the traditional semantics of $0/1$ and $false/true$ still hold; there is just an added layer of meaning to the variables. So, each clause of Q says that not all the clause's variables can be $\forall$ simultaneously. A unit clause means the given variable must be existentially quantified, in any valid sentence of P.

By ignoring the question of how difficult it is to produce Q, and instead concentrating on the final size of Q, several questions can be simply addressed. That is, the complexity of constructing Q is at least as hard as $|Q|$, so the size of Q gives a lower bound on the complexity for building Q.

Is cnf the right form to use for Q? The expression Q could be presented in several forms. There is disjunctive normal form, where all the valid quantifications are disjoined together, one clause per valid quantification. That form can easily become very large, so is not very useful or interesting, and for the moment we will ignore it. Conjunctive normal form is much smaller in many cases. For some special cases of P, Q in cnf form is quite small. Specifically, there are 2 easy cases which will be covered later: If P is monotone, then Q is identical to P, and therefore linear in $|P|$; and if P is a 2-cnf expression, then $|Q| \leq n^2$.

However, a simple to describe class will be presented showing that Q in cnf can be large, at least empirically. An attempt will be made to prove that $|Q|$ in that case must be large (and so $QSpace > NP$).

Q could also be presented using a nested $\wedge/\vee$ expression. However, the nested $\wedge/\vee$ form saves at most $O(n)$ over cnf, and cnf is much simpler to manipulate.

A detail that still needs to be cleared up is whether extra variables and negation can be introduced, and used to represent Q in a smaller form, while still allowing the satisfying assignments of Q to correspond exactly to the valid quantifications. Q is Always one of the monotone functions on n variables. When represented without negation or extra variables, the boolean expressions for these functions can be quite large. Some arguments will be discussed, after the general construction is given, but this area needs work.

One example, the Majority function, illustrates that this detail could become a major hurdle to overcome. The monotone expression for Q of a Majority function is exponential. So, it is likely that, by restricting Q to be a monotone cnf, $QSpace > NP$, but, it is not entirely certain that such a restriction is a good thing to do. The subject needs some exploration.

One significant observation is that the expression P itself may be small, and it does encode all the information for constructing Q. However, it appears that, when Q explicitly encodes the valid quantifications, Q must sometimes be large. This is a fact; even though P and Q have the same number of satisfying assignments, the change in the semantics from P to Q causes, in some cases, an exponential enlargement in the presentation. A thorough proof that Q (as a monotone cnf) must sometimes be large (and thus $QSpace > NP$) is illustrated by examining the Majority function. A related topic to explore is: Since All boolean functions map to monotone functions, which one is the smallest boolean expression mapping to a particular Q? This is a topic for further study, and the question is open whether, for every monotone Q expression there exists a small P expression that maps to it.

For convenience and simplicity, in this paper, "expression Q" generally means a monotone boolean expression in conjunctive normal form. The fact that Q, when it exists, is a monotone function, can vaguely be deduced from the proof of $\#P = \#Q$, and will be seen explicitly in the general construction of Q.

The cnf expression Q is a powerful representation. Q describes **all** the $\forall$ constraints (as clauses) derivable from the expression P. Once the expression Q is obtained, checking a quantification corresponds directly to checking an assignment in Q. That is, all QBFs of a particular P are reduced to individual assignment evaluations in Q.

The expression Q is sufficient for evaluating any quantification of P. By checking all of the clauses in Q, by "evaluating" a particular quantification/assignment, that quantification can be checked for validity. If the given assignment is valid, all "subsets" (change any $\forall$ to $\exists$) of the assignment are valid. That is a property of the monotonicity of Q.

Q has many other properties to explore. Such as, how many, and which, quantifications have the most $\forall$ assignments. This is a maximal assignment question, but it is essentially applied to QBFs.

In summary, the representation of Q as a monotone formula in conjunctive normal form is very convenient. There are some indications that it might not be the last word on the matter, but it is sufficient for an introduction.

In the empirical portions of the paper, the DIMACS format will be used to present Q expressions. In easy cases, the entire P and Q expressions will be given. In the hard cases, where Q is large, only a fragment of the Q expression will be given.

4 Two Special Cases

In this section, two special cases in which it is easy to construct Q are given. These are two special cases of the general construction method, so no proof that they are correct will be given. The first special case is for monotone P, and is apparently unknown in the literature (but very easy), and the second is for 2-cnf P expressions. A linear method for deciding $2-QBF$ was given by Tarjan [Tarjan 79]. The method presented here for 2-cnf is quadratic, but it gives a representation of all valid quantifications.

A third special case, that of small tree-width P expressions, is hypothesized to generate polynomial sized Q expressions, but it has not yet been explored. Horn formulas also need exploration.

4.1 Monotone Expressions

When P is monotone, just let $Q = P$, and add $\forall/\exists$ to the variables' boolean semantics. To check a particular quantification, just plug in the appropriate $0/1$ assignment ($xj = 0$ for $\forall xj$, $xj = 1$ for $\exists xj$) and evaluate. Iff Q evaluates to 1, or $true$, the quantification is valid. So, "constructing" Q is linear in $|P|$, and evaluating a monotone QBF is also linear in $|P|$.

An inductive proof that the "evaluation" method correctly decides QBF for monotone formulae looks similar to the proof of $\#P = \#Q$. The method was mentioned by the author on comp.theory in early 1998; the author does not know of any earlier reference. This fact is just one of many pearls uncovered by this work.

The parallel time complexity of deciding QBF on a monotone cnf, with m clauses, is $O(lg(m) + lg(n))$, with as many processors as there are literals in the expression. When m is polynomial in n, the time is logarithmic. If m is exponential in n, the time is linear. The exponential case does come up, when Q is large.

4.2 2CNF Expressions

When P is a 2cnf, the corresponding Q expression can be constructed in at most quadratic time and space. Extend P to include all resolutions of size 0, 1, and 2, and remove subsumed clauses. That extended expression, P', is at most $O(n^2)$ times as large as P. From P', Q can be constructed in linear time, so the entire method is quadratic. First, if $P' = false$, then $Q = false$. Otherwise, scan P' to produce Q:

For each clause of size 1 in P', either j or $-j$, add j to the clauses of Q. Essentially, since j has a constant value in P', it is an existential variable.

For clauses of size 2, separate into two cases:

Case 1: On two variables j and k, there is only one clause. Then, no matter what the particulars of that clause are, add the clause $(j \vee k)$ to Q. This says not both j and k can be universal at the same time.

Lemma: There are never 3 or more different clauses on two variables, in P'.

Case 2: There are two clauses on j and k, $j < k$. Then, the clauses on j and k express an (in)equivalence relationship between j and k. So, add k to the clauses of Q, since it is existentially defined in terms of j, and $j < k$. In other words, k is not universal; whatever j is, k is defined in terms of j.

The expression Q constructed by this method is monotone, and has the same number of satisfying assignments as the original P expression. The "evaluation" method can then be used with Q to check the validity of QBFs on the original P expression. An inductive proof that the construction is correct would again look much like the proof given for $\#P = \#Q$.

The parallel complexity for deciding quantifications of a $2cnf$ is then given by the time to compute P' (compute all the resolutions and subsumptions of the initial expression P), plus log of the size of P', to produce Q, plus the time to evaluate an assignment of the resulting Q expression.

The construction phase takes $O(log(n))$ time, with between $O(n^2)$ and $O(n^3)$ processors (a transitive closure-like operation is necessary). The evaluation phase takes $O(log(n))$ time, with $O(n^2)$ processors, and may be used repeatedly, with different quantifications of P.

There are surely other special cases for which it is polynomially easy to construct Q. For example, if the expression P has the "small tree-width" property (Arnborg, Proskurowski, more recently, [Bodl 97]), then perhaps Q is also small. The preliminary investigation appears to confirm that when P has sufficiently small tree-width, Q also has size polynomial in P. Horn, and other special types of boolean expressions are also desrving of exploration.

In the next section, a general method for constructing Q is given, based largely on the inductive step of the $\#P = \#Q$ proof. There is no claim that this method is the most efficient. In fact, the above special cases show that it is not the most efficient. However, the author has used the general method, and studied the output of some simple cases. One hard case in particular will be presented, giving empirical evidence that the Q expression is not always small.

Additionally, the general method has been used to generate millions of 20 and 30 variable test cases, which are used to check the correctness of qtwiddle, a qbf solver.

5 Constructing Q

This section presents a general method for constructing Q in conjunctive normal form. This method is inefficient for the previously described special cases. In fact, it is also inefficient for many hard cases. However, for the initial analysis, the main concern is the size of the output, not the computational cost of how it is found. By examining the size of the output, a lower bound on construction of Q can be obtained, even when the most efficient methods are used to obtain it.

The construction takes time and space proportional to the number of satisfying assignments of P, so is only practically useful on fairly small instances. There may be much better ways to produce Q, ways that are polynomial in the size of Q, ways that use connected components, et cetera. Since Q, which decides all 2^n QBFs of P, is always a monotone boolean function, an intensive study of monotone boolean functions is recommended. The web page www.bu.edu/wcp/Papers/Logi/LogiWeav.htm is an excellent example.

While the general method is exponential, it is simple; and it can be used to study the size of Q, to study lower bounds on the most efficient method to produce Q, which are lower bounds on $QSpace$.

Roughly, the method is as follows: Convert a tree of satisfying assignments into a Qtree, encoding valid quantifications. Then walk over that tree to output the clauses describing it.

1. Begin at the root of a tree of all satisfying assignments to an expression P, with the variables in proper order.

2. To convert a node of the tree, first convert the two descendant nodes. Then, let the left node become the union and the right node become the intersection of the two nodes.

3. Walk through the Qtree, recursively building up a list of monotone conjunctive constraints that describe the Qtree.

In short, a converted node contains valid quantifications "under" that node: the left node contains valid existential quantifications, and the right node contains valid universal descriptions, for the node. It would be illustrative to run through a small example by hand, however paper length prohibits this easy excercise. There are also methods for approximating Q, but they need further study.

6 A Hard Case For Constructing Q

When the P expression does not have very many satisfying assignments, Q is small. To produce large Q expressions, the P expression must have superpolynomially many assignments.

The case studied in this section comes from random $2 - 1$ regular $3 - cnf$ expressions. These expressions are generated as follows: Place 2 positive and 1 negative copy of each variable in a pool. Randomly pull out 3 literals at a time to construct a clause. Except for a few simplifications, where the same variable appears more than once in the same clause, these expressions will have n clauses of size 3 over n variables.

The empirical study of this case, using the general method for producing Q, produces exponentially large Q expressions, on average.

The following is a small example of a P expression and the corresponding Q expression, generated from a 2-1 regular 3-cnf expression. After that, a table and some fragments from larger instances will be given.

```
c \#Variables:           5
c \#Solutions:           12
c Clause sizes:   1: 0  2: 1  3: 4
p cnf 5 5
     1     3     5
    -1     4     5
     1    -4    -5
    -2     3     4
     2    -3

c Q in CNF form.
c 5 variables, 6 clauses
c Clause sizes:   1: 0  2: 4  3: 2
p cnf 5 6
     1     2     4
     1     2     5
     1     3
     2     3
     3     4
     4     5
```

According to $\#P = \#Q$, both P and Q have 12 satisfying assignments. The first clause of Q expresses the fact that variables 1, 2, and 4 cannot all be universally quantified in P. Any quantification that has variables 1, 2, and 4 universally quantified will be invalid. Perhaps the most interesting clause is "1 3". It is short, but it is not obvious how to deduce that variables 1 and 3 cannot both be universal, given P.

The "1 3" clause is one of the last produced by the general construction method. Given the above lexicographic ordering on the Q clauses, each variable k has a "greatest clause" beginning with k. A fundamental question is: how much effort is required to produce that clause? That is, although this paper claims that the main interest is the number of clauses in Q, in reality we Are interested in the cost of producing the individual clauses of Q. Therefore, the computational cost of producing the greatest Q clause that begins with a variable k (say, 1) is of theoretical interest, but beyond the scope of this paper. There is much about Q that is beyond the scope of this paper.

In the following table, the size of several Qs and their clauses, results from random 2-1 regular P expressions, are presented. The first column is the number of variables; the second column is the number of clauses in Q. The succeeding columns give a breakdown of the number of clauses with a particular size k.

		#clauses with size k								
n	m	k= 1	2	3	4	5	6	7	8	9
8	31	0	1	26	4					
10	67	0	1	25	41					
12	164	0	1	24	135	4				
14	313	0	1	21	187	103	1			
16	734	0	2	26	96	577	33			
18	1470	0	0	23	84	491	732	133	7	
20	2486	0	0	24	97	271	1195	844	54	1
22	3306	0	2	28	102	348	1541	1209	76	

In general, as n gets larger, most of the combinations of n choose j variables, for small but increasing j, have some clause containing them.

Unless the behavior drastically changes with larger n, the increases in number of clauses of Q appears to increase exponentially in the number of variables. Specifically, for small but increasing j, most of the combinations of n choose j variables appear somewhere in the clauses of Q.

The following figures give a 20 variable instance of P, and just a fragment of the corresponding Q expression. In the given P expression, there are $41,930$ satisfying assignments, with 20 clauses. The corresponding Q expression also has $41,930$ satisfying assignments, corresponding to all the valid quantifications of P. It takes 2486 clauses to describe Q.

```
c #Solutions:              41,930
c P in CNF form
c 20 variables, 20 clauses
c Clause sizes:   1: 0  2: 0  3: 20
p cnf 20 20
     1     3    12
    -1     9   -16
     1   -14   -20
     2    -4   -17
    -2     7    13
     2    10    12
     3     5    -7
    -3   -12    17
     4     5    11
     4     6    14
    -5    11    17
     6     7    18
```

```
   -6   -11   -15
    8    -9   -18
    8    10    15
   -8    14    19
    9   -13    15
  -10    16    20
   13    16    19
   18   -19    20

c Q in CNF form.
c 20 variables, 2486 clauses
c Clause sizes: 1:0 2:0 3:24 4:97 5:271 6:1195 7:844 8:54 9:1
p cnf 20 2486
     1     2     3     4     5    14    19
     1     2     3     4     5    18    19
     1     2     3     4     5    18    20
     1     2     3     4     5    19    20
     1     2     3     4     6     7     9    19
     1     2     3     4     6     8    11    13    19
     1     2     3     4     6     8    11    16
     1     2     3     4     6     8    15    19
     1     2     3     4     7     8     9    19
     1     2     3     4     7     8    15    16
     1     2     3     4     7     8    15    19
     1     2     3     4     7     8    15    20
     1     2     3     4     7    16    18    19
     1     2     3     4     8     9    11    19
     1     2     3     4     8    11    15    19
     1     2     3     4     8    16    19
     1     2     3     4     9    20
     1     2     3     4    14    16
...
    13    14    18    19
    13    14    18    20
    13    14    19    20
    13    16    18    20
    13    16    19
    14    15    18    19
    14    15    18    20
    14    15    19    20
    14    16    18    19
    14    16    18    20
    14    16    19    20
    18    19    20
```

Each individual clause of k variables describes a universal constraint on the expression P. Even if all the other variables are existential, when the k variables of a given clause are all universally quantified, that quantification is invalid. Q describes all of these constraints, in a relatively compact, but still sometimes large, form.

For readers' information, the greatest Q clause on variable 1 is: ($\vee$ 1 15 16 19 20). It says that these 5 variables cannot all be universal in the given P. For those adept at QBFs, that clause has several quantifier alternations. The author would be very interested in the shortest proof that these variables cannot all be universally quantified (with the rest existential) in the given P. The author believes a short proof does not exist; and, while the simplistic claim in this paper is that the size of Q is most interesting, in reality there is substantial interest in the complexity of producing individual clauses of Q.

The empirical evidence that Q gets large is strong. With some effort, a non-random P which provably produces large Q expressions will be discovered, and that will show that $QSpace > NP$ (as $QSpace$ is currently defined).

7 Questions and Conclusions

In this paper, a novel approach to problems beyond NP has been given. A method for exactly representing all valid quantifications of a boolean expression has been given. Once the expression Q has been constructed for a particular expression P, the verification of any particular quantification of P reduces to the evaluation of an assignment in Q. That is, QBF reduces to evaluation of an assignment, with time and space requirements that are linear in the size of Q. Monotone boolean formulas are an important easy case; they give a direct representation of all valid quantifications of themselves.

It has also been shown, empirically, there are cases in which the size of Q is super-polynomial. With some effort, the author is confident that a non-random hard case similar to the one presented here can be proven to be analytically hard (producing large Q expressions), and thus $QSpace > NP$. It is easy to construct such non-random cases, run the general construction method on them, and produce large Q expressions. However, simple proofs that large Q expressions can be produced are not obvious.

One rather intriguing result, already noted, is that Q essentially gives a list of all "minimally" invalid quantifications. For each of the m clauses of Q, not all variables in that clause can be simultaneously universal, even if the rest of the n variables are existential! That is a very strong statement, and shows that Q is quite a powerful form. Q is the minimal form (modulo a possible factor of average clause size) for explicitly stating that each of those m quantifactions is invalid. The fact that Q is sufficent to verify the validity or invalidity of all 2^n quantifications is rather a bonus. There is much more to be said about the expression Q, including its usefulness in circuit testing and verification.

There are some questions to be resolved. First, how close is Q in cnf to the actual minimal expression for Q? That is, is there a better way to represent Q? Since P contains all the information for "deriving" Q, is it fair to say that

Q in monotone conjunctive normal form is the correct method for representing all valid quantifications? Second, while assignment evaluation Q is sufficient for checking a QBF, since Q contains all universal constraints derivable from P, it could be that a particular quantification could be verified with less work than checking every minimal universal constraint derivable from P. It would be enlightening to come up with such examples, however "work" is measured. For example, if you want to know whether P is a tautology, there is no need to check all universal constraints derivable from P. However, when a quantification is complex, when it has many alternations of existential and universal quantifiers, then checking it by evaluating an assignment in Q does look appealing, even when Q is large. If such assignment evaluations are "the best we can do", then the size of Q constructively implies $PSpace > NP$.

Beyond the simplistic "size of $|Q|$" question, there are some other valuable questions to ask. In particular, these two questions are important: Lexicographically, what is the greatest clause in Q beginning with variable k? How large is the smallest proof, for any given clause, that all of its variables cannot be simultaneously universal?

In some sense, the general construction runs through all proofs simultaneously, walking over the Qtree from left to right, outputting the descriptive clauses as they occur. Is there a shorter way to prove any particular clause of Q?

Q describes all valid quantifications in a very direct manner. P contains, indirectly, all the information needed to deduce the valid quantifications. The empirical fact that Q can be exponentially larger than P is clear. A proper understanding of the differences between the directness of Q and the indirectness of P could lead to a clear separation of $PSpace$ from NP. Another question is, for a given Q, of the many boolean expressions that map to it, what is the smallest one?

There are other complexity questions raised, and possibly answered, by the size of Q. As a circuit, with bounded fan-in, Q has at most linear depth. That is, each clause can be evaluated with a log depth circuit; then, with m clauses, the whole circuit requires $log(m)$ depth. When m is exponential, QBFs need $O(n)$ depth circuits to decide them; when m is polynomial in n, then QBFs can be decided with $O(log(n))$ depth circuits.

The size of Q, as a simple measurement, appears to cleanly separate instances which are known to be easy from those that appear to be difficult. In the future, it will be interesting to find classes of boolean formulas which have $n^{k-1} < |Q| < n^k$, for increasing k, possibly with correlation to the polynomial hierarchy. The author expects instances with tree-width bounded by $klog(n)$ to have such behavior, but again, much further study is needed.

$QSpace$ needs a rigorous mathematical definition. The example from propositional logic is pretty clear, and the completeness of a variety of propositional logic problems tends to imply some sort of completeness for $QSpace$, but this has not yet been done. For example, NxN Go is $PSpace$-complete; the term "valid quantifications" as applied to Go needs formalization to obtain the corresponding $QSpace$ problem. Since $PSpace$-complete problems are easily translated to

QBF, it makes sense that constructing Q has some analog in each *PSpace*-complete problem, and not just with QBFs; however, in this paper, there is no such presentation.

The construction of the expression Q is given as the canonical problem in *QSpace*, but no completeness results or definitions using turing machines are provided. An assumption is made, based on the observed symmetries evident in the boolean and polynomial hierarchies, that completeness results of some kind are forthcoming. The current plan is to use Cook's Theorem as a model, generalize it to show that QBF is *PSpace*-complete, and generalize that to show that constructing Q is *QSpace*-complete. However, that is alot of work, and quite bluntly, it may be out of the author's reach, if it's possible at all. The basic patterns and symmetries evident in the boolean hierarchy give some hope that the project is feasible.

In the present paper, the idea is that, given Any method for producing Q, the size of Q gives a lower bound on the time needed to produce Q. Since the Q expression can be super-polynomial, that places *QSpace* outside of NP. To place *PSpace* outside of NP, it must be shown that there are some quantifications of hard instances that must essentially check a significant fraction of the clauses of Q. In other words, while the expression Q is sufficient for checking any quantification, it has not been shown that most of Q is necessary. Clearly, Q is useful for checking long series of different quantifications, especially an exponential number of different quantifications of the same expression P.

The general method for producing Q creates a monotone cnf. The dedekind sequence tells how many such monotone cnfs there are. Analysis of which monotone functions have small representation would very likely lead to easily solvable classes of *PSpace* instances.

The ideas of *QSpace* were used to, in a couple weeks, transform a satisfiability checking program into a qbf checking program. Bugs were quite easy to locate and squash. Even small (20 variable) examples can contain many alternations, and significant complexity.

In conclusion, the *QSpace* class provides a very promising general approach to higher order boolean problems. Studying the size of Q may correspond well with previous work on lower bounds for circuit size. The fact that *PSpace* sentences reduce to expression evaluation, and the expressions can get large, appears to be a promising route to showing $PSpace > NP$, and, of course, $PSpace > P$. At the very least, the approach illuminates some of the daunting higher levels of the hierarchy, and demonstrates that the reasoning about those higher levels need not be overly complex, regardless of their computational cost.

References

[GJ 84] Garey and Johnson. *Computers and Intractability* Second Edition, 1984.

[Tarjan 79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* 8(3):121-123, 15 March 1979.

[Bodl 97] Hans L. Bodlaender. Treewidth: Algorithmic Techniques and Results. *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science* MFCS'97, Lecture Notes in Computer Science, volume 1295,Springer-Verlag, pages 19-36.