

# Exploiting Structure in Quantified Formulas <sup>1</sup>

Richard E. Stearns and Harry B. Hunt III  
University at Albany - SUNY  
Department of Computer Science  
Albany, NY 12222  
E-mail: {res,hunt}@cs.albany.edu

Version: January 30, 2002

We study the computational problem “find the value of the quantified formula obtained by quantifying the variables in a sum of terms.” The “sum” can be based on any commutative monoid, the “quantifiers” need only satisfy two simple conditions, and the variables can have any finite domain. This problem is a generalization of the problem “given a sum-of-products of terms, find the value of the sum” studied in [34]. A data structure called a “structure tree” is defined which displays information about “subproblems” that can be solved independently during the process of evaluating the formula. Some formulas have “good” structure trees which enable certain generic algorithms to evaluate the formulas in significantly less time than by brute force evaluation. By “generic algorithm”, we mean an algorithm constructed from uninterpreted function symbols, quantifier symbols, and monoid operations. The algebraic nature of the model facilitates a formal treatment of “local reductions” based on the “local replacement” of terms. Such local reductions “preserve formula structure” in the sense that structure trees with nice properties transform into structure trees with similar properties. These local reductions can also be used to transform hierarchical specified problems with useful structure into hierarchically specified problems having similar structure..

*Key Words:* quantified formulas, generic algorithms, satisfiability problems, constraint satisfaction problems, non-serial dynamic programming, computational complexity, channelwidth, structure trees, treewidth, tree decompositions

## 1. INTRODUCTION

In the formal theory of computation, problem instances are modeled as strings and computational problems are modeled as language recognition problems. That is, for a given language  $L$ , the associated computational problem is to answer questions of the form “does a given string  $w$  belong

---

<sup>1</sup>supported in part by NSF Grant CCR 94-06611

to the language  $L$ ?” The traditional way of establishing complexity connections among languages is with resource-bounded reductions, especially polynomial-time or log-space reductions. Although the language recognition model and resource-bounded reductions play a central role in the theory of computation, they are limited by their very generality. Within this model, problem instances have neither semantics nor structure, and reductions may be unrelated to the meaning of the problem. Even linear-time reductions can relate problems which are computationally dissimilar. For example, one problem may have a polynomial approximation scheme and the other not. One problem may be easy when instances are specified hierarchically and the other problem hard.

In actual practice, problem instances are more usefully described in the style of Garey and Johnson [16] as graphs, relations, formulas, etc. When so described, one is able to discuss structured problem instances such as bounded-treewidth graphs, planar graphs, bounded-bandwidth formulas, etc. One is also able to discuss concepts of a “local reduction” based on local replacements.

We believe there is a need for models which fit between the very general language recognition model and the study of problems as individual problems. Such models should:

1. be specific enough to explain why certain algorithmic techniques apply;
2. explain results already known for a variety of individual problems or classes of problems;
3. enable computational properties of problems encountered in the future to be inferred directly from the problem descriptions; and
4. extend to succinct specification methods such as hierarchically specifications.

We expect that these more general models will be algebraic in nature, just as the the more general models in modern mathematics are algebraic in nature. By an “algebraic model”, we mean a model with algebraic operations and function symbols which specialize to particular computational problems when the operators and functions are particularized. Algorithms which apply to the uninterpreted abstract problem serve as “generic algorithms” which specialize to algorithms for particular problems when the operations and functions are particularized. Reductions described algebraically are usually “local” in nature, tend to preserve computational properties, and are analogous to the algebraic concept of a homomorphism.

The literature already has a number of instances where algebraic methods are used to model classes of problems. For the “constraint satisfaction problem” extensively studied in the AI literature, very general algebraic views are presented in [6] and [20] and many of the papers they reference.

Although constraint satisfaction can be viewed as a special case of our model, much of the theoretical focus of this work in the AI community has been on finding conditions on semantics under which the problem becomes amenable to polynomial time algorithmic techniques. Another way of modeling problems is through the use of second order formulas as discussed [4, 10, 11]. Although the papers [4, 11] look at structure in much the same way as we do here, their algorithms are fundamentally different from ours as explained further below. Other examples include the closed semi-ring model of path problems [1] and the matroid model for problems solvable by greedy algorithms [21].

In our earlier work [34], we looked at problems that can be described as “find a certain sum of products of terms” where the sum is taken over all assignments to a set variables.<sup>2</sup> In particular, we showed how the sum could be more efficiently computed when the set of terms had limited “weighted depth” or “channelwidth”. (These concepts will be defined later in this paper.) We say these concepts are “structure-based” because they depend only on which variables appear in which terms and do not depend on the interpretations of the operators and terms. Thus the concepts can be thought of as syntax-based rather than semantics-based.

Satisfiability and constraint satisfaction can be modeled as a sum-of-products where  $\wedge$  is the product operation and  $\vee$  is the sum operation. This sum-of-products has the value TRUE if and only if the conjunction (i.e. product) of terms is satisfiable.<sup>3</sup> Non-serial optimization can be modeled as a sum-of-products where the semi-ring product is ordinary arithmetic addition and the semi-ring sum operation is minimization. The value of the sum-of-products is then the smallest obtainable sum of terms. Although we say that satisfiability and optimization problems are modeled as sum-of-product problems, the actual objective of these problems is often to “find an assignment which gives the best value” rather than “find the best value.” In the general case, a sum-of-products is not associated with any particular assignment. For example, there is no assignment associated with the the problem “find the number of satisfying assignments”. This problem is modeled as a sum-of-products by treating terms as zero-one valued (instead of Boolean valued) and using the plus and times of ordinary arithmetic.

When the semi-ring addition satisfies  $(a + b = a) \vee (a + b = b)$  for all  $a$  and  $b$ , there is a natural association between values and assignments. In this case, any sum of values is equal to one (or more) of these values and any of the particular assignments which generate that value can be viewed as the assignment associated with the sum. In Section 9 of [34], it is shown

---

<sup>2</sup>All variables must be finite domain variables, the sum and product operations must be from a commutative semi-ring, and each term must map assignments to its variables to elements of the semi-ring.

<sup>3</sup>Our sum-of-products model can thus be viewed as a generalization of satisfiability which is why we call the problem a “generalized satisfiability problem” in [34]. It is a further generalization of Schaefer’s “generalized satisfiability” in [33].

how the associated assignment can be computed along with the value with very minor additional cost. Thus the difference between finding the value and finding the associated assignment is not computationally significant.

In this paper, we extend our work in [34] and develop a generic model of quantified formulas where generalized quantifiers are applied to a sum of monoid-valued terms. The computational problem is to compute the value of such a formula. The quantifiers that can be modeled include the familiar  $\exists$  and  $\forall$  for Boolean-valued monoids,  $\min$  and  $\max$  for numerical-valued monoids, the “stochastic quantifier” of [27], and many others. Within this more general framework, our “sum-of-products” formulas from [34] are viewed as quantified formulas involving just one kind of quantifier. The semi-ring product operator is now seen as the monoid operator and the semi-ring sum operator is now seen as a means of defining the quantifier.<sup>4</sup> There is no need to impose an ordering on the quantifiers in this one-quantifier case because the ordering does not affect the value.

We associate a quantified formula (non-uniquely) with a data structure we call a “structure tree”. The information in the structure tree can be interpreted as information about “subproblem independence” (in the sense described below) where the information at each node describes a subproblem. This structure can be exploited by certain algorithms for computing the formula’s value. Because the structure tree is based only on the formula’s syntax and is independent of the interpretation of the operators and terms, the computational implications of the structure tree apply across the entire spectrum of problems that can be modeled by quantified formulas.

The algorithms presented for exploiting structure trees are themselves independent of formula semantics. They are expressed generically as a sequence of uninterpreted actions such as performing a monoid operation or evaluating a term. Thus the algorithms apply to any problem expressible as a set of formulas. We use the number of operations performed as the algorithm’s “generic complexity”. Of course, the actual time complexity depends also on the cost of performing the algebraic operations. In the most common applications, the algebraic operations are relatively simple and the generic complexity accurately reflects the time complexity. In any case, the costs of particular operations for particular sets of formulas is not a topic of this paper.

The algorithms involve making trial assignments to various variables and then returning or remembering the results of certain sub-calculations. The sub-calculations involve evaluating formulas obtained from the original by replacing certain variables in the original problem with constants that have been assigned to them. When this is done, the original problem may fragment into pieces so that no two pieces have unassigned variables in common. It is these pieces which we refer to (informally) as “independent subproblems”. By evaluating these independent subproblems separately

---

<sup>4</sup>See later discussion of the  $\sigma$ -quantifier.

and combining their values, there can be a considerable savings in time costs over “brute force” methods. The set of subproblems that occur as the algorithms unfold is highly dependent on the order in which variables are assigned values.

The structural concepts in this paper are direct generalizations of the concepts of the same name in [34]. The difference between the structure trees of [34] and the structure trees here is that a restriction must be imposed here to take into account the ordering of the quantifiers. The restriction must be strong enough to guarantee that the generic algorithms will give the right answer. Otherwise, we want the weakest restriction possible because, the more structure trees a formula has, the more opportunity there is to find a good structure. To this end, we base our restriction on a relationship among variables we call the “influence relation.” This relation characterizes the quantifier re-orderings which (for all interpretations) do not change the formula’s value. This restriction says intuitively that the placement of information in the structure tree must be compatible with some quantifier order equivalent to the given order. Since there are interpretations where it would be a mistake to be compatible with a different ordering, our condition is the weakest (and thus the most permissive) possible. The restriction is based solely on the formula’s syntax and the structure tree concept remains free of any interpretation of the operators and terms.

The term “problem” is commonly used as in [16] to designate a set of “problem instances” for which one wants an algorithm to solve. Because formulas have a value and structure independent of what problems the formulas happen to belong to, we are able to view formulas as “problem instances” without having any particular set of instances in mind. The concept of a “problem” is not formally defined and methods of defining problem sets are not a topic of this paper. We think of formulas as instances of the meta-problem <sup>5</sup> “what is the value of a given formula?”. Because every set of formulas is a special case of this meta-problem, the techniques developed for solving formulas in general will apply to any set of formulas and thus to any problem expressible as a set of formulas.

There are close relationships between the structure trees of one-quantifier formulas and the tree decompositions of graphs and between the channel-width of formulas and the treewidth of graphs. (Tree decompositions and treewidth are defined in [2] and have been applied to solving graph problems where the graphs are restricted to graphs having bounded-treewidth. For examples, see [3, 4, 7, 24, 31].) When the translation of graph problems into formulas is very direct, a tree decomposition of a graph is essentially an unrooted structure tree for the formula and the treewidth of the graph (plus one <sup>6</sup>) is the formula treewidth. By “very direct”, we mean that the

<sup>5</sup>We call it a “meta-problem” because the problem domain is only loosely defined.

<sup>6</sup>A tree has treewidth 0 whereas the analogous formula (one variable for each graph node and one term for each graph edge) has channelwidth 1.

formula terms correspond to graph hyper-edges and variables correspond to vertices. This includes all problems in the class called ECC in [7]. In other circumstances (such as LCC from [7]) there is a useful but less direct transformation of tree decompositions into structure trees. Bounded bandwidth instances [25], bounded clique-width instances [12, 17], and  $k$ -outer-planar graphs [5] have bounded treewidth and hence have a similar relationship to structure trees.

Although much of the treewidth literature is motivated by bounded treewidth (which implies polynomial time algorithms) there are implications for problems which are not bounded by constant treewidth. For example, planar graphs have  $O(\sqrt{n})$  treewidth and this non-constant bound implies that many planar graph problems can be solved in time  $2^{O(\sqrt{n})}$  by methods similar to [22, 14, 29, 36]. Good structure trees (when they exist) can be found using either bottom-up techniques found in [2] or [8] or top-down separator techniques found in [23] (see Theorem 7.2(4) of [34]).

In the reverse direction, one can make a hyper-graph representing the structure of a formula (a hyper-edge for each term, a node for each variable) and tree decompositions of the hyper-graph correspond directly to unrooted structure trees and theorems about tree decompositions may be applied to structure trees. In the multi-quantifier case, the influence relation restricts the set of structure trees, and we do not have a purely graph-theoretic description of a structure tree.

The concepts of quantifier and graph structure have been linked in [4, 11] where monadic second-order logic is used to model questions about graphs as model checking problems. It is shown that questions are easier to answer for graph instances having favorable treewidth. However, the approach in [4, 11] is substantially different from ours. In [4, 11], a single formula describes a question to be asked of all graph instances. In our work, the problem instances themselves are formulas. In [4, 11], there is no concept of “formula structure”, and easiness is attributed to the structure of the graph. In our work, easiness is attributed to formula structure. In [4, 11], bounded-treewidth graph problems are easy for reasons other than sub-problem independence and are solved by methods which are not specializations of our generic procedures. The algorithms in [4, 11] are more than exponential in treewidth (no problem if the treewidth is bounded) and depend on the quantifier alternations being of fixed size. Our methods are only exponential in channelwidth and have no limitations on alternations.

An up-to-date survey on model checking problems can be found in [18]. This survey includes discussion of bounded treewidth methods. In general, both a formula and a model must be supplied as problem input, and easiness occurs when formula alternations are bounded and models have bounded treewidth. An exception is conjunctive queries where formula treewidth can be exploited to make certain problems easier. However, conjunctive queries have just one kind of quantifier and hence no alternations. Thus our work and the model checking work are substantially different.

We use formulas as problem input whereas model checking input is a combination of formula and model (sometimes just models). We get easier problem instances without restricting the number of alternations whereas easier model checking problems require that alternations be bounded. Finally, the algorithms for exploiting structure are different although they all are forms of dynamic programming.

The applications of our model go far beyond the algebra and graph applications suggested by the above discussion. Many other problems can also be encoded into quantified formulas in such a way that the value of the formula is the answer to the original question. Using just sums-of-products, one can (as discussed in [34]) write formulas which describe many complete problems in each of the classes NP, CoNP, #P,  $D^P$ , OPT-P, MAX SNP, and MAX  $\Pi_1$ .

Under our more general multi-quantifier model, one can use the two quantifier symbols  $\exists$  and  $\forall$  to describe quantified CNF and DNF formulas. Using the two quantifier symbols max and min, one can describe games where the players take turns assigning values to variables. Using max or min with the stochastic quantifier of [27], one can describe “games against nature”. It is now possible to describe complete problems in PSPACE and #PSPACE. The appendix gives some examples of the problems mentioned above and shows in detail how they can be described using quantified formulas.

Sections 2, 3, 4, and 5 provide the basic theory. Quantifiers are defined in Section 2, formulas in Section 3, and structure trees in Section 4. The structure tree concept involves the “influence relation”, also introduced in Section 4, that takes into account how quantifier lists might be re-ordered without changing the formula value. The generic algorithms are presented in Section 5 and it is seen that their generic complexity is a function of quantities known as “weighted depth” and “channelwidth” which are associated with structure trees and are defined in Section 4. The reason for the dependence only on weighted depth and channelwidth is that the structure allows quantifiers to be reordered and moved down onto subexpressions so as to lessen the depth of quantifier nesting and to provide opportunities for memoization and non-serial dynamic programming.

In Section 6, we look at the influence relation more closely. We find that the concept is robust in that it is invariant under permitted quantifier re-orderings. We also show that it was correctly defined in the sense that no weaker concept is appropriate for all monoids. Weakness is desirable here since it allows for the most structure trees and thus enhances the possibility of a having good structure tree.

A quantified formula with free variables describes a function on the free variables. One can sometimes use such a formula to specify the function represented by some function symbol in another formula. In such a case, terms using the function symbol can be replaced by their defining formula and the formula thereby transformed. We call this transformation a “local

replacement”. In Section 7, we show that this kind of local transformation is “structure preserving” in that a good structure tree for the original formula translates into a good structure tree for the transformed formula. These results have several implications:

1. When reductions are specified with such local transformations, structured instances are mapped into instances with similar structure.
2. The concept of “representability” as given by Schaefer [33] extends to quantified formulas.
3. The results on hierarchical representations as given in [34] extend to quantified formulas. Furthermore, local reductions of quantified hierarchical descriptions preserve the structure of both the descriptions and the expanded objects.

In Section 8, we find that the concept of an influence relation can be weakened when one quantifier satisfies an equation analogous to the familiar  $\forall_x(f(x) \wedge g(x)) = (\forall_x f(x)) \wedge (\forall_x g(x))$ . Obviously, this is an important special case because of the importance of Boolean monoids in modeling decision problems. A “best” definition of influence for this special case is given.

In Section 9, we discuss how to find good structure trees (if any) for a given formula. Included is a method of computing influence relations in  $O(n^3)$  time.

In Section 10, we consider the relationship between weighted depth and channelwidth and find that, unlike the single quantifier case (Section 6 of [34]) a formula with small channelwidth can have large weighted depth.

Finally, in Section 11, we consider the computational utility of unquantified channelwidth (that is channelwidth unconstrained by influence) in the quantified case. We find that formulas with small unquantified channelwidth can be used to describe NP-complete problems and thus (assuming  $P \neq NP$ ) are not helpful in the general case. This shows that the influence relation is an essential ingredient of the multi-quantifier theory and that more is required than a graphical analysis of which variables occur in which clauses.

## 2. QUANTIFIERS

Suppose  $D$  and  $S$  are sets and  $m$  is a function that maps  $S^D$  into  $S$ . If  $x$  is a variable with domain  $D$  and  $f(x)$  is a function from  $D$  to  $S$ , we use the notation  $m_x f(x)$  to represent the value  $m(f(x))$ . The function  $m$  behaves like a quantifier in that it can be used to map functions of several variables into functions of one less variable as in  $h(y) = m_x g(x, y)$ . Specifically, for any given value of  $y$ ,  $g(x, y)$  is a function of  $x$  alone and  $h(y)$  is defined to be the result of applying  $m$  to that function.

We are often interested in quantifiers which apply to more than one domain. For example, we want to treat  $\exists$  as a quantifier for any finite domain and seldom want to distinguish between  $\exists$  for some three-valued domain and  $\exists$  for some four-valued domain. If name  $m$  is associated with several quantifier functions, one for each domain in a set  $\mathcal{D}$ , then  $m_x$  is understood to be the quantifier function named  $m$  associated with the domain of  $x$ . More formally, if we have a function  $m^D : S^D \rightarrow S$  for each domain  $D$  in a set of domains  $\mathcal{D}$ , we use the notation  $m_x f(x)$  to represent the value  $m_x^{D_x}(f(x))$  where  $D_x \in \mathcal{D}$  is the domain of variable  $x$ .

DEFINITION 1. Let  $\mathcal{M} = (S, +, 0)$  be a commutative monoid and let  $\mathcal{D}$  be a set of finite domains. A **quantifier symbol**  $m$  for  $\mathcal{M}$  and  $\mathcal{D}$  is a set of functions  $m^D : S^D \rightarrow S$  for  $D$  in  $\mathcal{D}$  such that

1.  $m_x(f(x) + c) = (m_x f(x)) + c$  for all functions  $f : D_x \rightarrow S$  and all  $c \in S$ ;
2.  $m_x m_y g(x, y) = m_y m_x g(x, y)$  for all functions  $g : D_x \times D_y \rightarrow S$ ;

where  $D_x$  and  $D_y$  are the domains of variables  $x$  and  $y$ . We refer to the combination  $m_x$  of quantifier symbol and variable as an  $\mathcal{M}$ -**quantifier**.

We define several familiar quantifier symbols in this framework:

DEFINITION 2. Define the symbols  $\exists$ ,  $\forall$ ,  $\max$ ,  $\min$ ,  $\sigma$ ,  $\mathfrak{R}$ , and  $\tau$  as follows:

1. For the monoids  $\mathcal{B}_\wedge = (\{\text{TRUE}, \text{FALSE}\}, \wedge, \text{TRUE})$  and  $\mathcal{B}_\vee = (\{\text{TRUE}, \text{FALSE}\}, \vee, \text{FALSE})$  and any finite domain  $D$ , define  $\exists_x f(x) = \bigvee_{d \in D} f(d)$  and  $\forall_x f(x) = \bigwedge_{d \in D} f(d)$ .
2. For ordered commutative monoid  $\mathcal{M} = (S, +, 0, \leq)$ <sup>7</sup> and non-empty finite domain  $D$ , define  $\max_x f(x) = \max\{f(d) \mid d \in D\}$  and  $\min_x f(x) = \min\{f(d) \mid d \in D\}$ .
3. Let  $\mathcal{R} = (S, +, \cdot, 0, 1)$  be a commutative semi-ring. For the monoid  $(S, \cdot, 1)$  and finite domain  $D$ , define the  $\sigma$ -**quantifier** by  $\sigma_x f(x) = \sum_{d \in D} f(d)$ .
4. Let  $\mathcal{F} = (S, +, \cdot, 0, 1)$  be a commutative field containing the integers. For the monoid  $(S, +, 0)$  and finite domain  $D$ , define the **stochastic quantifier**  $\mathfrak{R}$  by  $\mathfrak{R}_x f(x) = (\sum_{d \in D} f(d) / |D|)$ .
5. For idempotent commutative monoid  $\mathcal{M} = (S, +, 0)$ <sup>8</sup> and finite domain  $D$ , define the  $\tau$ -**quantifier** by  $\tau_x f(x) = \sum_{d \in D} f(d)$ .

<sup>7</sup>In an ordered monoid,  $a \leq c$  and  $b \leq d$  imply  $a + b \leq c + d$ .

<sup>8</sup>In an idempotent monoid,  $a + a = a$  for all  $a \in S$ .

We claim that the above definition defines quantifier symbols:

PROPOSITION 1.  $\exists, \forall, \max, \min, \sigma, \mathfrak{R}$ , and  $\tau$  from Definition 2 are quantifier symbols.

*Proof.* These symbols obviously satisfy Definition 1. ■

Quantifier symbols  $\exists, \forall, \max$ , and  $\min$  as defined above have their standard meanings. The formulas from [34], defined as sums of products, are in effect expressions of the form  $\prod_{p \in P} p$  with all variables quantified with  $\sigma$  and the value (as defined in [34]) of the formula is just the value (as defined in Definition 6) of this expression. Stochastic quantifiers were defined in [27] (read  $\mathfrak{R}_x$  as “for random  $x$ ”). The definition of  $\mathfrak{R}$  here extends in the obvious way to other probability distributions on the domain. Specifically, if  $p_d \in S$  for  $d \in D$  and  $\sum_{d \in D} p_d = 1$ , then  $m$  is a quantifier where  $m_x f(x) = \sum_{d \in D} (p_d \cdot f(d))$ . The  $\tau$ -quantifier concept is new to this paper and is discussed further in Section 8. It has certain interesting properties not associated with the other quantifiers because the  $\tau$  quantifier satisfies the equation  $\tau_x(f(x) + g(x)) = \tau_x f(x) + \tau_x g(x)$ .

Most of the functions associated with the quantifier symbols in Definition 2 can be expressed by a formula of the form

$$m_x f(x) = \bigoplus_{d \in D_x} f(d)$$

where  $\oplus$  is some kind of commutative associative operator on the set  $S$  from the commutative monoid  $\mathcal{M} = (S, +, 0)$  under consideration. Because of Definition 1(1),  $+$  and  $\oplus$  must satisfy the distributive law  $(c+a) \oplus (c+b) = c + (a \oplus b)$ . Quantifier symbols defined in this manner have a meaning for all finite domains. Usually the  $\oplus$  is different than the monoid  $+$ ; but in the case of the  $\tau$  quantifier they are the same.

PROPOSITION 2. *Quantifier symbols  $\exists, \forall, \max, \min, \sigma$ , and  $\tau$  are related as follows:*

1. for the monoids  $\mathcal{B}_\wedge$  and  $\mathcal{B}_\vee$  under the ordering  $\text{FALSE} < \text{TRUE}$ ,  $\min$  and  $\forall$  are the same and  $\max$  and  $\exists$  are the same;
2. for the monoids  $\mathcal{B}_\wedge$  and  $\mathcal{B}_\vee$  under the ordering  $\text{TRUE} < \text{FALSE}$ ,  $\max$  and  $\forall$  are the same and  $\min$  and  $\exists$  are the same;
3. if ordered commutative monoid  $\mathcal{M} = (S, +, 0, \leq)$  has a largest element  $\infty$ , then  $\max$  is the  $\sigma$ -quantifier for the commutative semi-ring  $(S, \max, +, \infty, 0)$ ;
4. if ordered commutative monoid  $\mathcal{M} = (S, +, 0, \leq)$  has a smallest element  $-\infty$ , then  $\min$  is the  $\sigma$ -quantifier for the commutative semi-ring  $(S, \min, +, -\infty, 0)$ ;

5. the  $\sigma$ -quantifier for semi-ring  $(\{\text{TRUE}, \text{FALSE}\}, \vee, \wedge, \text{FALSE}, \text{TRUE})$  is  $\exists$  and the  $\sigma$ -quantifier for semi-ring  $(\{\text{TRUE}, \text{FALSE}\}, \wedge, \vee, \text{TRUE}, \text{FALSE})$  is  $\forall$ ;
6. the  $\tau$ -quantifier for  $\mathcal{B}_\wedge$  is  $\forall$  and the  $\tau$ -quantifier for  $\mathcal{B}_\vee$  is  $\exists$ .

*Proof.* Obvious. ■

Definition 1 excludes some quantifier concepts that might be legitimately considered as quantifiers in other contexts. One such example is the Boolean valued “there exists a unique” defined by  $U_x f(x) \iff (|\{x|f(x)\}| = 1)$ . This “quantifier symbol” fails to satisfy Condition 1 for either monoid  $\mathcal{B}_\wedge$  or  $\mathcal{B}_\vee$ . It also fails to satisfy Condition 2 for domains with more than two elements. The same remark applies to other quantifiers based on cardinality such as those in [26]. Even  $\exists$  and  $\forall$  fail Condition 1 for the monoid of Boolean values under exclusive-or. These excluded quantifiers are of no interest in the context of our theory because they cannot be moved around in their quantifier lists. When Condition 1 fails, even the concept of “independent subproblems” breaks down. It should be noted, however, that “quantifiers” satisfying Condition 1 but not Condition 2 can be accommodated in our theory by treating different occurrences of the quantifier symbol as different symbols.

```

DECLARE ARRAY  $T[D_x]$  OF  $S$ 
FOR  $x \leftarrow$  ALL  $d$  IN  $D_x$ 
   $T(x) \leftarrow f(x)$ 
END FOR
 $z_x \leftarrow m(T)$ 

```

**FIG. 1** General Pseudo-code for  $m_x f(x)$

```

INITIALIZE  $z_x$ 
FOR  $x \leftarrow$  ALL  $d$  IN  $D_x$ 
   $z_x \leftarrow z_x \oplus f(x)$ 
END FOR

```

**FIG. 2** Special Pseudo-code for  $m_x f(x)$

We close this section with some discussion about computing  $m_x f(x)$ . Figure 1 gives pseudo-code which applies to all quantifiers and Figure 2 gives pseudo-code which applies to all quantifiers obtained from a  $\oplus$  operator. In both cases, the program ends with variable  $z_x$  having the value  $m_x f(x)$ .

We call the loop in Figure 1 the “standard loop for  $m$ ”. To interpret the final step  $z_x \leftarrow m(T)$ , remember that a quantifier is really a function on functions from  $D_x$  to  $S$  (Definition 1) and that array  $T$  specifies such a function. If  $S$  is finite, then each assignment and the final  $m(T)$  each take only constant time. If  $S$  is infinite, then the actual costs must take into account the representation of  $S$  and the subset of  $S$  that might be returned by  $f(x)$ . Remember, however, that our interest is in a generic assessment of the complexity and that we limit our analysis to counting operations. In this case, we observe that the standard loop performs  $2 \cdot |D_x| + 1$  assignments and one call on the quantifier function.

The pseudo-code of Figure 2 is simpler in that it doesn’t require the array  $T$ . However, the burden of keeping the array  $T$  is inconsequential in comparison with the other effects of large domains. In our discussions of space complexity, we take the liberty of ignoring the contributions of these arrays. Technically, this can be justified by assuming all variables are from some fixed domain or by assuming all quantifiers can be computed as in Figure 2.

### 3. QUANTIFIED FORMULAS

Here we provide definitions enabling us to discuss formulas and their values. First we provide some notation for describing assignments:

**DEFINITION 3.** Given a set of variables  $V$ , an **assignment on  $V$**  is a pairing in which each variable  $x$  from  $V$  is paired with a value in the domain of  $x$ . The set of assignments on  $V$  will be designated by the notation  $\Gamma(V)$ .  $\Gamma(\emptyset)$  contains one assignment, namely the empty set of pairs. For any assignment  $\gamma$ , we denote the variables in  $\gamma$  by  $VAR(\gamma)$  (i.e.  $VAR(\gamma) = V$  if and only if  $\gamma \in \Gamma(V)$ ). If  $\gamma_1$  and  $\gamma_2$  are assignments such that  $VAR(\gamma_1) \cap VAR(\gamma_2) = \emptyset$ , we let  $\gamma_1 + \gamma_2$  be the assignment in  $\Gamma(VAR(\gamma_1) \cup VAR(\gamma_2))$  formed by taking the union of the two assignments.

Next we introduce notation for writing terms:

**DEFINITION 4.** A **base symbol  $f$**  is a symbol which has an associated integer  $k$  called the **arity** of  $f$  and an associated vector of  $k$  domains  $D_1 \cdots D_k$ . A **term** for  $f$  is a string of the form  $f(x_1, \dots, x_k)$  where the  $x_i$  are variables or constants and the domain of  $x_i$  is  $D_i$ . If  $f$  is the name of a function which maps  $D_1 \times \cdots \times D_k$  into a set  $S$ , any term for  $f$  is also called an  **$S$ -term**. If  $S$  is the set of elements from a monoid  $\mathcal{M}$ , an  $S$ -term will also be called an  **$\mathcal{M}$ -term**.

For any term  $p = f(x_1, \dots, x_k)$ ,  $|p|$  is defined to be  $k + 1$  and is called the **size** of  $p$ .  $VAR(p)$  is defined to be the set of variables on the list  $x_1, \dots, x_k$ . If  $\gamma$  is an assignment,  $p[\gamma]$  is defined to be the term obtained from  $p$  by replacing each  $x_i$  in  $VAR(\gamma) \cap VAR(p)$  by its assigned value. If  $P$  is a set of terms, define  $VAR(P) = \cup_{p \in P} VAR(p)$ .

In practice, we allow any notation for terms where the meaning is clear. Thus the clause  $(x \wedge \bar{y} \wedge z)$  is considered a  $\{\text{TRUE}, \text{FALSE}\}$ -term even though not expressed in the functional notation  $f(x, y, z)$

For monoid  $\mathcal{M} = (S, +, 0)$ , one can construct quantified expressions in the usual way using quantifiers, terms, the operator  $+$ , and matching parentheses. We have a special notation for denoting formulas in prenex form with a kernel that is the sum of terms.

**DEFINITION 5.** Let  $\mathcal{M} = (S, +, 0)$  be a commutative monoid. A **quantifier list**  $Q$  for  $\mathcal{M}$  is a list of  $\mathcal{M}$ -quantifiers such that no variable appears on the list more than once. Let  $VAR(Q)$  denote the set of variables on list  $Q$  and  $|Q|$  denote  $|VAR(Q)|$ . If  $x$  and  $y$  are in  $VAR(Q)$ , we write  $x <_Q y$  if and only if  $x$  occurs before  $y$  on list  $Q$ . If  $V$  is a set of variables,  $Q_V$  denotes the list  $Q$  with the variables not in  $V$  deleted (i.e.  $Q$  restricted to  $V$ ).

Example:  $Q = \max_w \min_x \min_y \max_z$  is a quantifier list,  $VAR(Q) = \{w, x, y, z\}$ ,  $Q_{\{x, y\}} = \min_x \min_y$ , and  $w <_Q y$ .

**DEFINITION 6.** Let  $\mathcal{M} = (S, +, 0)$  be a commutative monoid, let  $Q$  be a quantifier list for  $\mathcal{M}$ , and let  $P$  be a set of  $\mathcal{M}$ -terms. Then we let the pair  $F = (Q, P)$  denote the expression  $Q(\sum_{p \in P} p)$  and we call  $F$  a **quantified  $\mathcal{M}$ -formula**. If  $W$  is a set of variables such that  $W \cap VAR(Q) = \emptyset$  and  $W \supset VAR(P) - VAR(Q)$  and if  $\gamma \in \Gamma[W]$ , then the notation  $(Q, P)[\gamma]$  denotes the value of the quantified expression  $Q(\sum_{p \in P} p[\gamma])$ .

We let  $FREE(F)$  denote the set  $VAR(P) - VAR(Q)$ , the set of **free variables** in  $F$ . If  $FREE(F)$  is empty, then  $F$  is called **fully quantified** and we define  $VALUE(F)$  to be the value of the denoted expression.

Example: If  $Q = \max_y \min_z$  and  $P = \{f(z, y), g(y, z)\}$ , then  $F = (Q, P)$  denotes the quantified expression  $\max_y \min_z (f(z, y) + g(y, z))$ . Since  $F$  is fully quantified, this formula represents a single value denoted by  $VALUE(F)$ . In logic, fully quantified formulas are usually called “sentences”.

For a monoid, the result of summing an empty set of values is defined to be the monoid identity 0. Thus Definition 6 makes sense even if  $P$  is empty. In fact, the only use of the monoid identity in this paper is that the theorems and definitions make sense for empty sums. It would be awkward but doable to develop this theory without using 0.

We will refer to “quantified formulas” instead of “quantified  $\mathcal{M}$ -formulas” if the monoid is understood or if the formula is uninterpreted.

Note that if  $F = (Q, P)$  is a fully quantified formula, we can also write  $VALUE(F) = (Q, P)[\gamma_0]$  where  $\gamma_0$  is the one assignment on the empty set  $VAR(P) - VAR(Q)$  of free variables.

One way to compute the value of a fully quantified  $\mathcal{M}$ -formula  $(Q, P)$  is to nest standard loops from Figure 1, one loop per variable, together

with the obvious inner loop ranging over  $p \in P$ . For a formula with two quantified variables, the pseudo-code is shown in Figure 3.

```

DECLARE ARRAYS  $T_1[D_x]$  AND  $T_2[D_y]$  OF  $S$ 
FOR  $x \leftarrow$  ALL  $d$  IN  $D_x$ 
  FOR  $y \leftarrow$  ALL  $d$  IN  $D_y$ 
     $v \leftarrow 0$ 
    FOR  $p \in P$ 
       $v \leftarrow v + p(x, y)$ 
    END FOR
     $T_2[y] \leftarrow v$ 
  END FOR
   $T_1[x] \leftarrow \overline{m}(T_2)$ 
END FOR
 $z \leftarrow m(T_1)$ 

```

**FIG. 3** Pseudo-code for  $m_x \overline{m}_y (\sum_{p \in P} p)$ .

Notice that the pseudo-code has four kinds of operations, namely predicate evaluations,  $+$  operations, calls on the quantifier functions, and assignments. We describe the generic time complexity of our generic algorithms by counting the number of times each of these operations is performed. In the case of nested standard loops, this generic complexity is described by the following:

**PROPOSITION 3.** *If an expression of the form  $Qf(\dots)$  is evaluated using nested standard loops, then*

1. *the number of times each  $p$  in  $P$  is evaluated is  $|\Gamma[\text{VAR}(Q)]|$ ,*
2. *the number of  $+$  operations is  $|P| \cdot |\Gamma[\text{VAR}(Q)]|$ ,*
3. *the number of calls on the quantifier function for a given variable is  $O(|\Gamma[\text{VAR}(Q)]|)$ ,*
4. *the number of assignment operations is  $O(|\Gamma[\text{VAR}(Q)]|(|Q| + |P|))$ .*

*Proof.* Obvious. ■

This method is “brute force” in the sense that it evaluates the sum of the terms for all assignments to the variables. The lemma implies that the brute force method takes exponential time even if the terms are easy to evaluate and the operations easy to perform. We will see that some formulas can be solved much faster by exploiting the “subproblem independence” that is displayed in a “structure tree” as defined in the next section. The speed-up can be attributed to the fact that structure trees display groups

of quantified variables which can be interchanged and moved down onto subexpressions. In preparation for these results, we finish this section with some key observations about moving and interchanging quantifiers. Moving or exchanging quantifiers is OK as long as the original and modified expressions are “equivalent” in the following sense:

**DEFINITION 7.** Two expressions are **equivalent** if and only if they have the same free variables and, for any assignment  $\gamma$  to the free variables, the resulting expressions have the same value.

**LEMMA 1.** *Let  $\mathcal{M} = (S, +, 0)$  be a commutative monoid and let  $E_1$  and  $E_2$  be expressions constructed from quantifiers,  $\mathcal{M}$ -terms, the operator  $+$ , and parentheses. Let  $x$  be a variable which is not quantified in  $E_1$  and does not occur at all in  $E_2$  and let  $m_x$  be an  $\mathcal{M}$ -quantifier. Then the expressions  $m_x(E_1) + E_2$  and  $m_x(E_1 + E_2)$  are equivalent.*

*Proof.* Immediate from Condition 1 of Definition 1. ■

This lemma leads to the following result for interchanging adjacent quantifiers.

**THEOREM 1.** *Let  $\mathcal{M} = (S, +, 0)$  be a commutative monoid, let  $(Q, P)$  be a quantified  $\mathcal{M}$ -formula, let  $x$  and  $y$  be two variables not in  $\text{VAR}(Q)$ , and let  $m$  and  $\bar{m}$  be quantifier symbols (not necessarily distinct). Then*

$$(m_x \bar{m}_y Q, P) \text{ and } (\bar{m}_y m_x Q, P)$$

denote equivalent expressions if either

1.  $m = \bar{m}$  or
2.  $P$  can be partitioned into two sets  $P_1$  and  $P_2$  such that  $x \notin \text{VAR}(P_2)$ ,  $y \notin \text{VAR}(P_1)$ , and  $\text{VAR}(Q) \cap \text{VAR}(P_1) \cap \text{VAR}(P_2) = \emptyset$ .

*Proof.* Condition 1 is immediate from Definition 1(2). To prove Condition 2, consider the expression  $m_x \bar{m}_y Q(E_1 + E_2)$  where  $E_1 = \sum_{p \in P_1} p$  and  $E_2 = \sum_{p \in P_2} p$ . Each quantifier can be moved down onto one of the subexpressions  $E_1$  or  $E_2$  using Lemma 1 until finally  $m_x$  is moved down to  $E_1$  and  $\bar{m}_y$  moved down to  $E_2$ . Then the the quantifiers can be moved back up to obtain the ordering  $\bar{m}_y m_x Q$ . ■

The above proof says that, for any assignment  $\gamma$  to the free variables, the value of both formulas can be obtained by solving independent subproblems  $(m_x Q_1, P_1)[\gamma]$  and  $(\bar{m}_y Q_2, P_2)[\gamma]$  (where  $\text{VAR}(Q_1) \cup \text{VAR}(Q_2) = Q$  and  $\text{VAR}(Q_1) \cap \text{VAR}(Q_2) = \emptyset$ ) and adding the results together. We call these “subproblems” because the formulas have only free variables in common and thus become “independent” after the free variables have been assigned values. The potential of subproblem independence for speed-up can be seen from the fact that solving the subproblems by brute force is exponential only in the sizes of  $Q_1$  and  $Q_2$  rather than in the size of  $Q$ .

## 4. STRUCTURE TREES

Given a fully quantified  $\mathcal{M}$ -formula  $F = (Q, P)$ , we would like to compute  $VALUE(F)$  faster than by the exhaustive method previously discussed. Given that formulas can directly describe NP- and PSPACE-complete problems, it would be remarkable (for complexity theory reasons) if we could do this in general but we can do it better if we can associate the formula with a suitable “structure tree” as defined below. This tree can be exploited in several ways, as given in Section 5, to compute the value. In this section, we present the structure tree concept and certain algebraic equations which explain the correctness of the faster algorithms given in Section 5.

In [34], a structure tree concept for unquantified problems was developed. There, subproblem independence could be based entirely on how variables were shared by terms. Here, the matter is more complicated because of the quantifiers. The following concept will enable a structure tree concept which takes the quantifiers into account.

**DEFINITION 8.** Let  $F = (Q, P)$  be a quantified formula and let  $m_x$  and  $\overline{m}_y$  be quantifiers in  $Q$ . We write  $x \prec y$  or  $x$  **influences**  $y$  if and only if  $m \neq \overline{m}$  and there exists a sequence of variables  $z_1 \dots z_k$  from  $VAR(Q)$  such that

1.  $z_1 = x$  and  $z_k = y$ ;
2. for each  $i$ ,  $1 \leq i < k$ , there is a term  $t_i$  in  $P$  such that  $\{z_i, z_{i+1}\} \subset VAR(t_i)$ ;
3.  $z_1 <_Q z_i$  for  $i$ ,  $1 < i \leq k$ ;
4. there does not exist a  $j$ ,  $1 < j < k$ , such that the quantifier symbol for  $z_j$  is not  $\overline{m}$  and  $z_j <_Q z_i$  for all  $i$ ,  $j < i \leq k$ .

The sequence  $z_1 \dots z_k$  is called an **influence sequence from  $x$  to  $y$** .

Relation  $\prec$  is a partial ordering because it is contained in  $<_Q$ . The purpose of  $\prec$  is to describe when quantifiers can be reordered. More specifically, Theorem 10 shows that two adjacent quantifiers  $m_x$  and  $\overline{m}_y$  in quantifier list  $Q$  can be interchanged without changing the value of  $(Q, P)$  whenever  $x$  and  $y$  do not satisfy  $x \prec y$  or  $y \prec x$ . Theorem 11 says that, without considering semantics, no smaller  $\prec$  has this property.

Condition 4 says in effect that “no suffix of an influence sequence is an influence sequence.” This is implied by the following proposition:

**PROPOSITION 4.** *Let  $F = (Q, P)$  be a quantified formula and let  $z_1 \dots z_k$  be a sequence of variables from  $VAR(Q)$  such that  $z_1$  and  $z_k$  have different quantifier symbols and such that Conditions 2 and 3 of Definition 8 hold. Then there is a unique  $i$ ,  $1 \leq i < k$ , such that subsequence  $z_i \dots z_k$  is an influence sequence.*

*Proof.* There cannot be distinct  $i$  and  $j$  such that both  $z_i \dots z_k$  and  $z_j \dots z_k$  are influence sequences because (assuming  $i < j$ ) Condition 4 for  $z_i \dots z_k$  asserts that such a  $j$  does not exist.

If  $z_1 \dots z_k$  is not an influence sequence, it must be because Condition 4 fails. The negation of Condition 4 implies there must be a maximal  $j$  such that  $z_j$  has a quantifier symbol different from  $z_k$  and  $z_j <_Q z_i$  for all  $i, j < i \leq k$ . Clearly  $z_j \dots z_k$  is an influence sequence. ■

The influence relation is a purely combinatorial concept in that it applies to any quantified formula and can be computed without knowing the semantics of the quantifiers and terms. This is why the statement of the definition says “quantified formula” instead of “quantified  $\mathcal{M}$ -formula”.

Note that the influence relation is empty if all the quantifiers have the same quantifier symbol. This reflects the fact that quantifiers with the same quantifier symbol can be interchanged arbitrarily.

When we use the terms “ancestor” and “descendant”, we consider any tree node to be an ancestor and a descendant of itself.

**DEFINITION 9.** Let  $F = (Q, P)$  be a quantified formula. A **structure tree  $S$  for  $F$**  is a triple  $(T, \alpha, \beta)$  where

1.  $T$  is a rooted tree with node set  $N$ ,
2.  $\alpha : VAR(Q) \rightarrow N$  gives the **variable association**,
3.  $\beta : P \rightarrow N$  gives the **term association**,
4. for all  $y$  in  $VAR(Q)$  and  $p$  in  $P$ , if  $y \in VAR(p)$  then  $\alpha(y)$  is an ancestor of  $\beta(p)$  in  $T$ ,
5. if  $x$  and  $y$  are in  $VAR(Q)$  and  $x \prec y$ , then  $\alpha(x)$  is an ancestor of  $\alpha(y)$ .
6. If  $x$  in  $FREE(F)$ , then  $\alpha(x)$  is the root of  $T$ .

This is essentially the definition of a structure tree for formulas as given in [34] with Conditions 5 and 6 added. Condition 5 can be interpreted as asserting that any information upon which a variable really depends must be placed higher in the tree. This condition is vacuous if all the quantifier symbols are the same because then the influence relation is empty. Condition 6 takes effect only when the quantified formula has free variables. The structure trees of [34] are in effect structure trees for one quantifier symbol formulas with no free variables.

There are several concepts pertaining to the nodes of the structure tree that are needed later. We present these in the next definition.

**DEFINITION 10.** Let  $F = (Q, P)$  be a quantified formula and  $S = (T, \alpha, \beta)$  be a structure tree for  $F$  with node set  $N$ . For each node  $n$ , define the following:

1.  $A(n) = \{y \in VAR(Q) \mid \alpha(y) = n\}$ , the variables **associated** with  $n$ .
2.  $B(n) = \{p \in P \mid \beta(p) = n\}$ , the terms **associated** with  $n$ .
3.  $AD(n)$  is the union of the  $A(n')$  such that  $n'$  is a descendant of  $n$ .
4.  $BD(n)$  is the union of the  $B(n')$  such that  $n'$  is a descendent of  $n$ .
5.  $y$  in  $VAR(Q)$  is a **branch variable** at node  $n$  if and only if  $\alpha(y)$  is an ancestor of  $n$ . Let  $BV(n)$  be the set of branch variables at  $n$ .
6.  $y$  in  $VAR(Q)$  is a **channel variable** at node  $n$  if and only if  $y$  is a branch variable and either (i)  $n = \alpha(y)$  or (ii) there is a  $p \in P$  such that  $y \in VAR(p)$  and  $\beta(p)$  is a descendant of  $n$ . Let  $CV(n)$  be the set of channel variables at  $n$ .

In (6), Case i is implied by Case ii whenever  $y$  appears in some term. One consequence of Case i is that every variable is a channel variable of at least one node.

Intuitively, if  $y$  is a variable and  $p$  a term such that  $y \in VAR(p)$ , the tree nodes  $n$  which connect  $\alpha(y)$  to  $\beta(p)$  must be used to “channel” the value assigned to  $y$  to the term  $p$  which uses the value. Thus we say  $v$  is a “channel variable” of  $n$ . Similarly, one can think of  $CV(n)$  as the set of variables  $v$  which have node  $n$  in their scope.

The algorithms described in the next section take a structure tree as part of their input and the complexities of these algorithms are characterized in terms of the parameters “weighted depth” and “channelwidth” defined as follows:

**DEFINITION 11.** If  $S$  is a structure tree with node set  $N$ , we define  $WD(S)$ , the **weighted depth** of  $S$ , to be the maximum of  $\{|BV(n)| \mid n \in N\}$ . We define  $CW(S)$ , the **channelwidth** of  $S$ , to be the maximum of  $\{|CV(n)| \mid n \in N\}$ .

If  $F$  is a formula, the minimum weighted depth over all structure trees for  $F$  is called the **weighted depth** of  $F$ . The minimum channelwidth over all structure trees for  $F$  is called the **channelwidth** of  $F$ .

There are various identities that hold among the concepts introduced in Definition 10. Two needed later are as follows:

**PROPOSITION 5.** *If  $S$  is a structure tree for formula  $(Q, P)$  and node  $n'$  is a child of node  $n$  in  $S$ , then*

1.  $BV(n) = BV(n') - A(n')$
2.  $CV(n) \supset CV(n') - A(n')$

*Proof.* These are straightforward consequences of Definition 10. ■

We note that the concepts in Definitions 10 and 11 are identical to those for the structure trees of unquantified formulas in [34]. They are purely combinatorial in that they apply to any quantified formula and not just  $\mathcal{M}$ -formulas. The remaining results in this section relate the structure of  $\mathcal{M}$ -formulas to their values. The results are central to the correctness of our generic algorithms.

**THEOREM 2.** *Let  $S = (T, \alpha, \beta)$  be a structure tree for quantified  $\mathcal{M}$ -formula  $F = (Q, P)$  and let  $n$  be any node of  $T$ . Then the expression denoted by the  $\mathcal{M}$ -formulas  $(Q_{AD(n)}, BD(n))$  and  $(Q_{A(n)}Q_{AD(n)-A(n)}, BD(n))$  are equivalent.*

*Proof.* We want to show that  $(Q_{A(n)}Q_{AD(n)-A(n)}, BD(n))$  can be systematically transformed into  $(Q_{AD(n)}, BD(n))$  where each formula in the sequence denotes an expression equivalent to that denoted by its predecessor. This transformation is achieved by repeatedly interchanging any two adjacent quantifiers  $\overline{m}_y$  followed by  $m_x$  where  $x \in AD(n) - A(n)$ ,  $y \in A(n)$ , and  $m_x <_Q \overline{m}_y$ . Each quantifier list  $Q'$  obtained during these interchanges will have the following two properties:

1. for all  $v_1 \in AD(n) - A(n)$  and  $v_2 \in AD(n)$ ,  $v_1 <_{Q'} v_2$  implies  $v_1 <_Q v_2$ .
2. for all  $v_1 \in A(n)$  and  $v_2 \in AD(n)$ ,  $v_2 <_{Q'} v_1$  implies  $v_2 <_Q v_1$ .

(Proof: list  $Q_{A(n)}Q_{AD(n)-A(n)}$  has these properties and the interchanges preserve them.) Once all such interchanges have been performed, the result is clearly list  $Q_{AD(n)}$ .

If  $Q' = Q_1 m_x \overline{m}_y Q_2$ , we need only show that  $(m_x \overline{m}_y Q_2, BD(n))$  and  $(\overline{m}_y m_x Q_2, BD(n))$  denote equivalent expressions. If  $\overline{m} = m$ , then the expressions are equivalent by Theorem 1(1). Now suppose the quantifiers  $m$  and  $\overline{m}$  are different. We consider two cases.

Case 1: There exists a sequence of variables  $z_1 \dots z_k$  such that  $x = z_1$ ,  $y = z_k$ , and for each  $i$ ,  $1 \leq i < k$ , there is a term  $t_i$  in  $BD(n)$  such that  $VAR(t_i) \supset \{z_i, z_{i+1}\}$  and  $x <_{Q'} z_i$  for  $1 < i < k$  (and by Property 1  $x <_Q z_i$ ). We also have  $x <_Q z_i$  for  $i = k$  since  $z_k = y$  and  $x <_Q y$  is assumed. Because  $z_1 \dots z_k$  satisfies the hypothesis of Proposition 4, there is an  $i$  such that  $z_i \dots z_k$  is an influence relation. Thus  $z_i \prec z_k (= y)$  even though  $\alpha(y) = n$  and  $\alpha(z_i)$  in  $AD(n) - A(n)$  is below  $n$ . This violates Definition 9(5). Thus Case 1 is impossible.

Case 2: The sequence in Case 1 does not exist. Then  $P$  can be partitioned into two sets  $P_x$  and  $P_y$  where  $P_x$  contains the terms with variables connected to  $x$  by such sequences and  $P_y$  contains the remaining terms. This partition satisfies Theorem 1(2) and thus the interchange is valid. ■

**THEOREM 3.** *Let  $S = (T, \alpha, \beta)$  be a structure tree for quantified  $\mathcal{M}$ -formula  $F = (Q, P)$ , let  $n$  be any node of  $T$ , and let  $n_1, \dots, n_k$  be the children of  $n$ . Then the expressions*

$$Q_{AD(n)}\left(\sum_{p \in BD(n)} p\right)$$

and

$$Q_{A(n)}\left[\sum_{p \in B(n)} p + \sum_{i=1}^k Q_{AD(n_i)}\left(\sum_{p \in BD(n_i)} p\right)\right]$$

are equivalent.

*Proof.* From Definition 10 it follows that  $AD(n) = A(n) \cup (\bigcup_1^k AD(n_i))$  and  $BD(n) = B(n) \cup (\bigcup_1^k BD(n_i))$ . Thus the two expressions have the same terms and the same quantified variables and thus the same free variables. If  $A(n) = AD(n)$ , then the  $AD(n_i)$  and hence the  $Q_{AD(n_i)}$  are empty and the expressions are immediately the same. If  $A(n) \neq AD(n)$ , there must be a first quantifier  $m_x$  on list  $Q_{AD(n)}$  which is not on  $Q_{A(n)}$ . Quantifier  $m_x$  must be on some  $Q_{AD(n_i)}$  and must be first on this list (any preceding quantifier would also precede  $m_x$  on  $Q_{AD(n)}$  contradicting the choice of  $m_x$ ). By Lemma 1,  $m_x$  can be moved out to the end of the list  $Q_{A(n)}$ . Repeating this with the second quantifier and so forth until all the quantifiers are outside, we can write the expression as

$$Q_{A(n)}Q_{AD(n)-A(n)}\left(\sum_{p \in BD(n)} p\right)$$

This is the expression denoted by formula  $(Q_{A(n)}Q_{AD(n)-A(n)}, BD(n))$  and Theorem 2 thus says this expression is equivalent to the expression denoted by  $(Q_{AD(n)}, BD(n))$ , namely  $Q_{AD(n)}(\sum_{p \in P} p)$ . ■

Theorem 3 can also be expressed as follows:

**PROPOSITION 6.** *Let  $S = (T, \alpha, \beta)$  be a structure tree for quantified  $\mathcal{M}$ -formula  $F = (Q, P)$ , let  $n$  be any node of  $T$ , and let  $n_1, \dots, n_k$  be the children of  $n$ . Then the expression denoted by formula  $(Q_{AD(n)}, BD(n))$  is equivalent to the expression described by*

$$Q_{A(n)}\left[\sum_{p \in B(n)} p + \sum_{i=1}^k (Q_{AD(n_i)}, BD(n_i))\right]$$

This formulation suggests that formulas can be evaluated by recursive procedures which assign values to structure tree nodes. This is the topic of the next section.

Theorem 3 also implies a method of interpreting the structure tree as the description of an expression in which quantifiers have been rearranged and moved down onto subformulas:

**THEOREM 4.** *Let  $S = (T, \alpha, \beta)$  be a structure tree for quantified  $\mathcal{M}$ -formula  $F = (Q, P)$ . The expression denoted by  $F$  is equivalent to the expression  $E$  described by the following grammar having one nonterminal  $E_n$  for each node of  $T$ , having starting nonterminal  $E_r$  where  $r$  is the root of  $T$ , and having the following production for each node  $n$ :*

$$E_n \rightarrow Q_{A(n)} \left[ \sum_{p \in B(n)} p + \sum_1^k E_{n_k} \right]$$

where nodes  $n_1 \cdots n_k$  are the children of node  $n$ .

*Proof.* It can be seen inspecting the two expressions in Theorem 3 that each nonterminal  $E_n$  generates an expression equivalent to the expression denoted  $(AD(n), BD(n))$ . ■

**PROPOSITION 7.** *The quantifiers in expression  $E$  of Theorem 4 are nested to a depth equal to the weighted depth of the structure tree.*

*Proof.* Obvious from the construction in the proof of Theorem 4. ■

If the formula  $F$  in Theorem 3 is fully quantified, then the expression derived from the structure tree can be converted using standard loops into a program to find the value of  $F$ . By Proposition 7, these loops are nested only to the weighted depth of the structure tree and thus the number of operations performed by this program is exponential only in weighted depth. This is a non-recursive description of Plan 1 in the next section. The running time may be improved to be exponential only in channelwidth (by employing memoization or dynamic programming) to eliminate redundant calculations. This is the essence of Plan 2 in the next section. Both plans are can be much faster than brute force plans which are exponential in  $|Q|$ .

Notice that the expression  $E$  from Theorem 3 and hence the expression's value depends both on the quantifier order and the structure tree. If  $|A(n)| \leq 1$  for all tree nodes  $n$ , the expression  $E$  can be constructed from the structure tree alone and there is thus a single value associated with that tree. Later we will need the fact that every formula has at least one such tree:

**THEOREM 5.** *Every fully quantified formula  $(Q, F)$  has a structure tree with  $|A(n)| = 1$  for all tree nodes  $n$ .*

*Proof.* Suppose  $Q$  is  $m_{x_1}^1 \dots m_{x_k}^k$  for some  $k$ . Let  $T$  be the tree with nodes  $n_1 \dots n_k$  where the parent of  $n_{i+1}$  is  $n_i$  for  $1 \leq i < k$ . Let  $\alpha(x_i) = n_i$  for  $1 \leq i \leq k$  and  $\beta(p) = n_k$  for all  $p$  in  $P$ . It is easily verified that  $(T, \alpha, \beta)$  is a structure tree for  $(Q, P)$ . ■

The structure tree from the proof is just a representation of brute force evaluation.

## 5. COMPUTING THE VALUE

In this section, we consider two plans for exploiting a given structure tree to find the value of a  $\mathcal{M}$ -formula. Both are based on Proposition 6. One plan allows the value to be computed in time exponential only in the weighted depth of the structure tree. The space used by this plan is essentially linear. The other plan allows the value to be found in time exponential only in the channelwidth. This can be a considerable improvement over the time of Plan 1 since channelwidth can be much smaller than weighted depth (See Section 10). However, Plan 2 has a big disadvantage in that it uses space exponential in the channelwidth.

**Plan 1:** Given a fully quantified  $\mathcal{M}$ -formula  $(Q, P)$  and a structure tree for the formula, the value of the formula can be found by recursively computing the various  $(Q_{AD(n)}, BD(n))[\gamma]$  where  $n$  is a node of the structure tree and  $\gamma$  is an assignment from  $\Gamma[BV(n) - A(n)]$ . (Note that  $BV(n) - A(n)$  is empty if  $n$  is the root and otherwise, by Proposition 5, is the set of branch variables of the parent of  $n$ .) The computation is done using the following ideas:

1. To compute  $(Q_{AD(n)}, BD(n))[\gamma]$  for some node  $n$  of the structure tree and some  $\gamma \in \Gamma[BV(n) - A(n)]$ , use Proposition 6 and standard loops to obtain this value. Inside the innermost loop, an assignment  $\gamma' \in \Gamma(A(n))$  is under consideration, and the following items are computed and summed:
  - $\sum_{p \in B(n)} p[\gamma + \gamma']$
  - $(Q_{AD(n')}, BD(n'))[\gamma + \gamma']$  for all children  $n'$  of  $n$ . (The requirement that  $\gamma + \gamma'$  be in  $\Gamma[BV(n') - A(n')]$  is met because of Proposition 5(1).)
2. To start the recursion, find  $(Q_{AD(r)}, BD(r))[\gamma]$  where  $r$  is the root of the structure tree and  $\gamma$  is the one assignment in  $\Gamma[BV(r) - A(r)] = \Gamma[\emptyset]$ . (Observe that  $AD(r) = VAR(Q)$  and  $BD(r) = P$  and thus this call returns  $VALUE(P, Q)$ )
3. Use global variables, one for each variable of  $Q$ , to keep the  $\gamma$  and  $\gamma'$  required in (1).

To bound the complexity of this procedure, we observe the following:

**LEMMA 2.** *Let  $F = (Q, P)$  be a fully quantified  $\mathcal{M}$ -formula and let  $S = (T, \alpha, \beta)$  be a structure tree for  $F$ . Then, when an algorithm following Plan 1 above is used to find  $VALUE(F)$ ,*

1. *for each node  $n$  of  $T$  and assignment  $\gamma$  in  $\Gamma(BV(n) - A(n))$ , the value of  $(Q_{BV(n)}, BD(n))[\gamma]$  is computed exactly once,*
2. *each term  $p \in P$  is evaluated  $|\Gamma(BV(\beta(p)))|$  times,*

3. each variable  $x$  in  $\text{VAR}(Q)$  is assigned a value at most  $|\Gamma(BV(\alpha(x)))|$  times.

*Proof.* Let  $n$  and  $\gamma$  be as in Part 1. If  $n$  is the root,  $BV(n) - A(n)$  is empty and  $\gamma$  must be the one assignment in  $\Gamma(BV(n) - A(n))$ . This one assignment corresponds to the initial procedure call. If  $n$  has a parent  $\bar{n}$ ,  $\gamma = \gamma_1 + \gamma_2$  where  $\gamma_1 \in \Gamma(BV(n) - A(n))$  and  $\gamma_2 \in \Gamma(A(n))$ . The corresponding call occurs when  $(Q_{BV(\bar{n})-A(\bar{n})}, BD(\bar{n}))[\gamma_1]$  is being evaluated and the variables in  $A(\bar{n})$  have been set to  $\gamma_2$ . Thus Part 1 holds.

A given term  $p \in P$  is evaluated only when some  $(Q_{BV(\beta(p))-A(\beta(p))}, BD(\beta(p)))[\gamma]$  is being evaluated and some assignment  $\gamma'$  to  $A(\beta(p))$  has been made. But  $\gamma + \gamma'$  is an assignment to  $BV(\beta(n))$  and so  $p$  is evaluated once for each such assignment. Thus part 2 holds.

Part 3 follows because the variable  $x$  is assigned a value only when node  $\alpha(x)$  is under consideration and  $x$  is changed while the loop for  $x$  is executed. This happens at most once for each assignment to the variables in  $BV(\alpha(x))$  (less if  $x$  is not the last quantified variable on list  $Q_{A(\alpha(x))}$ ). ■

There is nothing in the definition of a structure tree to prevent the number of nodes from being much greater than the size of the formula itself. Since our plans involve inspecting nodes of the tree, we cannot get a meaningful complexity bound until we show that a large number of nodes cannot be helpful.

**DEFINITION 12.** Let  $(T, \alpha, \beta)$  be a structure tree for quantified formula  $F = (Q, P)$  and let  $n$  be a node of  $T$ . Node  $n$  is called **surplus** if and only if  $n$  is not the root and  $A(n) = \emptyset$ .

**PROPOSITION 8.** Let  $(T, \alpha, \beta)$  be a structure tree for quantified formula  $F = (Q, P)$ . The following transformation will remove a surplus node  $n$  from  $T$  without changing the branch variables or the channel variables of the remaining nodes:

1. change the parent of the children of  $n$  to be the parent of  $n$ ,
2. for  $p$  in  $B(n)$ , change  $\beta(p)$  to the parent of  $n$ ,
3. remove  $n$  from the node set.

*Proof.* Easily verified. ■

Since surplus nodes are easily removed from a structure tree without changing the essential features (the branch and channel variables) we can restrict our attention to running the generic algorithms on structure trees which have no surplus nodes. In this case, we have a nice bound on the number of nodes:

**PROPOSITION 9.** Let  $S = (T, \alpha, \beta)$  be a structure tree for quantified formula  $F = (Q, P)$ . If  $S$  has no surplus nodes, then the number of nodes in  $T$  is at most  $|Q| + 1$ .

*Proof.* Since every non-root node  $n$  of  $T$  has  $A(n) \neq \emptyset$  and since  $\alpha$  (by definition) takes all free variables to the root, the number of non-roots is bounded by  $|Q|$ . ■

Now we can state the full implications of Lemma 2:

**THEOREM 6.** *Given  $F$  and  $S$  as in Lemma 2 where the weighted depth of the tree is  $WD$  and where  $S$  has no surplus nodes and given a bound  $D$  on the size of the variable domains, then during the execution of the algorithm,*

1. *the number of term evaluations is  $O(|P| \cdot D^{WD})$ ,*
2. *the number of + operations is  $O((|P| + |Q|) \cdot D^{WD})$ ,*
3. *the number of calls on quantifier functions is  $O(|Q| \cdot D^{WD})$ ,*
4. *the number of assignment operations is  $O((|P| + |Q|) \cdot D^{WD})$ ,*

*Proof.* For any node  $n$  of  $T$ ,  $|BV(n)| \leq WD$  by definition and  $|\Gamma(BV(n))| \leq D^{WD}$ . Part 1 of the theorem is then immediate from Part 2 of the lemma.

The plus operation occurs immediately after a term is evaluated or a node is visited. A particular term  $p$  is evaluated only when some assignment to the branch variables of  $\beta(p)$  is under consideration and each node is visited only when some assignment to the parent is under consideration. Part 2 follows.

The quantifier function for some variable  $x$  is computed once for each assignment to the variables before  $x$  in  $BV(\alpha(x))$  and Part 3 follows.

Assignments occur after each plus operation, each call on a quantifier function, each time a loop variable is changed, and each time a loop is started. The total occurrences of each of these cases is within the stated bound and so Part 4 holds. ■

The theorem implies that the time of the algorithm will be  $O(|F| \cdot D^{WD})$  whenever the operations, assignments, and term evaluations can be done quickly. The space is linear in  $|Q|$  whenever values can be stored in constant space and domain sizes are bounded.

There is some inefficiency built into Plan 1. For every node  $n$  of the structure tree, the algorithm ends up computing the value of  $F_n = (AD(n), BD(n))$  for all assignments  $\gamma$  in  $\Gamma(BV(n) - A(n))$ . However, this value depends only on  $\gamma$  restricted to the set of free variables in  $F_n$  and here this is the set  $CV(n) - A(n)$ . This means that the same value is recomputed  $|\Gamma(BV(n) - CV(n))|$  times. The time of the algorithm can be improved by maintaining a table  $T_n$  to store each of the various  $F_n[\gamma]$  the first time it is computed. This value can then be looked-up instead of recomputed whenever it is needed again.

**Plan 2:** Given a fully quantified  $\mathcal{M}$ -formula  $F = (Q, P)$ , the value of the formula can be found by systematically constructing tables, one for

each structure tree node  $n$ , where table  $T_n$  has an entry for each  $\gamma$  in  $\Gamma[CV(n) - A(n)]$ . The entry is used to store the value  $(Q_{AD(n)}, BD(n))[\gamma]$  once it has been computed. This is done using the following ideas:

1. To compute  $(Q_{AD(n)}, BD(n))[\gamma]$  for some node  $n$  of the structure tree and some  $\gamma \in \Gamma(CV(n) - A(n))$ , use Proposition 6 and standard loops. Inside the innermost loop, an assignment  $\gamma' \in \Gamma(A(n))$  is under consideration, and the following items are summed:
  - the result of computing  $\sum_{p \in BD(n)} p[\gamma + \gamma']$
  - for each child  $n'$  of  $n$ , the value of  $(Q_{AD(n')}, BD(n'))[\gamma + \gamma']$  which is table entry  $T_{n'}(\gamma_0)$  where  $\gamma_0$  is  $\gamma + \gamma'$  restricted to variables in  $CV(n') - A(n')$ . (Proposition 5(2) insures that the variables in  $\gamma + \gamma'$  can be restricted to  $CV(n') - A(n')$ .)
2. Let  $r$  be the root of the structure tree. The table for  $T_r$  has one entry because  $CV(r) - A(r)$  is empty. Upon completion, this entry contains  $VALUE(F)$ .
3. The table values can be found with a top-down approach of computing an entry when it is first needed (a process sometimes called “memoization”) or with a bottom-up approach of constructing the table for a given structure tree node only after the tables for the node’s children have been constructed (a process usually called “dynamic programming.”)

LEMMA 3. *Let  $F = (Q, P)$  be a fully quantified  $\mathcal{M}$ -formula and let  $S = (T, \alpha, \beta)$  be a structure tree for  $F$ . Then, when an algorithm following Plan 2 above is used to find  $VALUE(F)$ ,*

1. *for each node  $n$  of  $T$  and assignment  $\gamma$  in  $\Gamma(CV(n) - A(n))$ , the value of  $(Q_{CV(n)}, BD(n))[\gamma]$  is computed exactly once,*
2. *each term  $p \in P$  is evaluated  $|\Gamma(CV(\beta(p)))|$  times,*
3. *each variable  $x$  in  $VAR(Q)$  is assigned a value at most  $|\Gamma(CV(\alpha(x)))|$  times,*
4. *for each non-root node  $n$  of  $T$ , the table  $T_n$  is accessed  $|\Gamma(CV(n'))|$  where  $n'$  is the parent of  $n$ .*

*Proof.* Similar to the proof of Lemma 2. ■

THEOREM 7. *Given  $F$  and  $S$  as in Lemma 3 where the channelwidth of the tree is  $CW$  and where  $S$  has no surplus nodes and given a bound  $D$  on the size of the variable domains, then during the execution of the algorithm,*

1. *the number of term evaluations is  $O(|P| \cdot D^{CW})$ ,*

2. the number of + operations is  $O((|P| + |Q|) \cdot D^{CW})$ ,
3. the number of calls on quantifier functions is  $O(|Q| \cdot D^{CW})$ ,
4. the number of assignment operations is  $O((|P| + |Q|) \cdot D^{CW})$ ,
5. the number of table look-ups is  $O(|Q| \cdot D^{CW})$ .

*Proof.* Similar to the proof of Theorem 6. ■

The theorem implies that the time of the algorithm will be  $O(|F| \cdot D^{CW})$  whenever the operations, assignments, and term evaluations can be done quickly. The tables computed by the algorithm have  $O(D^{CW})$  entries.

## 6. QUANTIFIER ORDER AND INFLUENCE

Let  $F = (Q, P)$  and  $F' = (Q', P)$  be quantified formulas where  $Q'$  is a permutation of  $Q$ . We are interested in the following question:

Under what circumstances can we say that  $F$  and  $F'$  have the same value under any interpretation?

In this section, we give the following answer:

Formulas  $F$  and  $F'$  have the same value (and the same influence relation and the same structure trees) if  $Q'$  is consistent with the influence relation of  $F$ .

By “consistent”, we mean the following:

DEFINITION 13. Let  $Q$  be a quantifier list and let  $<$  be a partial ordering on  $VAR(Q)$ . We say that  $Q$  is **consistent with**  $<$  if and only if  $x < y$  implies that  $x$  comes before  $y$  in  $Q$ .

We also want to show that our answer is the best possible in that, if  $Q'$  is not consistent with the influence relation of  $F$ , then there is some interpretation of the terms and quantifiers for which the values are different. We show this in Theorem 11 with a very natural interpretation involving the quantifiers  $\min$  and  $\max$  and the monoid  $\mathcal{I}$  of integers under addition.

It is possible that, for particular quantifiers, the answer is not the best possible and that some other definition of “influence” works out better. This is true for the  $\tau$ -quantifier (Definition 2(5)), an important special case because of Proposition 2(6). This case is given special consideration in Section 8.

LEMMA 4. *If  $P$  is a set of terms and  $Q$  and  $Q'$  are two quantifier lists with the same variables such that formulas  $(Q, P)$  and  $(Q', P)$  have different influence relations, then there is an influence sequence  $z_1 \dots z_k$  for  $(Q, P)$  such that  $z_k \dots z_1$  is an influence sequence for  $(Q', P)$ .*

*Proof.* Consider the shortest sequence  $z_1 \dots z_k$  which is an influence sequence for one formula but not for the other. We can assume without loss of generality that  $z_1 \dots z_k$  is an influence sequence for  $(Q, P)$  but not for  $(Q', P)$ . We let  $m$  be the quantifier symbol for  $z_1$  and  $\overline{m}$  the quantifier symbol for  $z_k$ . We want to show that  $z_k \dots z_1$  is an influence sequence for  $(Q', P)$ . Let  $z_\ell$  be the variable ( $1 \leq \ell \leq k$ ) such that  $z_\ell <_{Q'} z_i$  for all  $i$ ,  $1 \leq i \leq k$  and  $i \neq \ell$ .

Consider first the case where the quantifier symbol for  $z_\ell$  is not  $\overline{m}$ . Then Proposition 4 says there is an  $i$  such that  $z_i \dots z_k$  is an influence sequence for  $(Q', P)$ . We cannot have  $i = 1$  because it is assumed  $z_1 \dots z_k$  is not an influence sequence for  $(Q', P)$ . We cannot have  $i > 1$  because then  $z_i \dots z_k$ , being shorter than  $z_1 \dots z_k$ , would also be an influence sequence for  $(Q, P)$  contradicting the uniqueness asserted by Proposition 4.

Now consider the case where the quantifier symbol for  $z_\ell$  is  $\overline{m}$ . Proposition 4 then says there is an  $i$ ,  $\ell \geq i > 1$ , such that  $z_i \dots z_1$  is an influence relation for  $(Q', P)$ . If  $i < k$ ,  $z_i \dots z_k$  must also be an influence sequence for  $(Q, P)$ . But this is impossible because we cannot have both  $z_1 <_Q z_i$  and the relationship  $z_i <_Q z_1$  required by Definition 8(3).

The one remaining possibility is that  $i = k$ . This means  $z_k \dots z_1$  is an influence relation for  $(Q', P)$  which is what we wanted to show. ■

**THEOREM 8.** *If  $P$  is a set of predicates and  $Q$  and  $Q'$  are two quantifier lists with the same quantifiers such that formulas  $(Q, P)$  and  $(Q', P)$  have different influence relations  $\prec$  and  $\prec'$ , then there are two variables  $x$  and  $y$  such that  $x \prec y$  and  $y \prec' x$ .*

*Proof.* Immediate from Lemma 4. ■

**THEOREM 9.** *If  $\prec$  is an influence relation for formula  $(Q, P)$  and  $Q'$  is a permutation of  $Q$ , formula  $(Q', P)$  also has influence relation  $\prec$  if and only if  $Q'$  is consistent with  $\prec$ .*

*Proof.* Suppose  $(Q', P)$  has a different influence relation  $\prec'$ . Then Theorem 8 says there are two variables  $x$  and  $y$  such that  $x \prec y$  and  $y \prec' x$ . But  $y \prec' x$  implies by Definition 8 that the quantifier for  $y$  comes before the quantifier for  $x$  in  $Q'$  and so  $Q'$  is not consistent with  $\prec$ .

Now suppose that  $Q'$  is not consistent with  $\prec$ . This means that, for some  $x$  and  $y$ ,  $x \prec y$  but the quantifier for  $y$  comes before the quantifier for  $x$  in  $Q'$ . By Definition 8(3),  $x$  cannot influence  $y$  in  $(Q', P)$  and so  $(Q', P)$  has a different influence relation. ■

**THEOREM 10.** *If quantifier lists  $Q$  and  $Q'$  have the same quantifiers and formulas  $(Q, P)$  and  $(Q', P)$  have the same influence relation, then the two formulas have the same structure trees and the same value.*

*Proof.* The definition of a structure tree depends only on which variables belong to which terms and on the influence relation. These relationships are the same for both formulas so they must have the same structure trees.

The structure tree for  $(Q, P)$  given by Theorem 5 must therefore be a structure tree for  $(Q', P)$ . Since  $|A(n)| = 1$  for each tree node  $n$ , the expression  $E$  from Theorem 4 is the same for both formulas and so both formulas have the same value. ■

We now provide a converse for Theorem 10. The result uses the monoid  $\mathcal{I}$  of integers under addition and the quantifier symbols max and min. A formula constructed from this monoid and these quantifier symbols can be interpreted as a zero-sum two-person game. In this game, a maximizing player gets to choose an assignment to all variables quantified by max and a minimizing player gets to choose an assignment to variables quantified by min. The variables are assigned in the order they appear on the quantifier list and each player is aware of all the previous choices when it is his turn to assign a particular variable. After all variables are assigned, the minimizing player pays the maximizing player the sum of the terms under the given assignment.

In our proof, we make use of the facts from game theory (which we will not prove) that the value of the formula is the game theoretic value of the game and that this value is no smaller than any value guaranteed by some strategy of the maximizing player and no larger than any value guaranteed by some strategy of the minimizing player.

The reader may find it helpful to study the example given after the proof.

**THEOREM 11.** *Let  $P$  be a set of terms such that each term has a distinct base symbol. Let  $Q$  and  $Q'$  be quantifier lists such that  $Q$  and  $Q'$  are permutations of each other. If quantified formulas  $(Q, P)$  and  $(Q', P)$  have different influence relations, then there is an association of base symbols with integer valued functions and quantifier symbols with max and min such that quantified  $\mathcal{I}$ -formulas  $(Q, P)$  and  $(Q', P)$  have different values.*

*Proof.* From Lemma 4, we know there is an influence sequence  $z_1 \dots z_k$  for  $(Q, P)$  such that  $z_k \dots z_1$  is an influence sequence for  $(Q', P)$ . We associate the quantifier symbol of  $z_1$  with max and the symbol of  $z_k$  with min. From definition 8, there are terms  $t_i$  in  $P$  for  $1 \leq i < k$  such that  $z_i$  and  $z_{i+1}$  are in  $VAR(t_i)$ . To keep notation simple, we assume each that  $t_i$  has exactly two variables and can be written  $f_i(z_i, z_{i+1})$ . We will also assume these are the only terms in  $P$ , that all variables have the domain  $\{0, 1\}$ , and that there are only two quantifier symbols. At the end, we explain how these assumptions can be easily dropped.

If  $z_{i+1}$  has quantifier symbol min, we define  $f_i(x, y) = 0$  if  $x = y$  and  $f_i(x, y) = 1$  otherwise. If  $z_{i+1}$  has quantifier symbol max, we define  $f_i(x, y) = 1$  if  $x = y$  and  $f_i(x, y) = 0$  otherwise. Let  $v_0$  be the sum of all the terms when all variables have the value zero. By symmetry of the construction, this is also the value when the variables have the value one. We will show that the value of quantified  $\mathcal{I}$ -formula  $(Q, P)$  is  $v_0$  and the value of quantified  $\mathcal{I}$ -formula  $(Q', P)$  is at least  $v_0 + 1$ .

Considering  $(Q, P)$  as a game, we will show that the maximizing player can insure a result of at least  $v_0$  by assigning the same value to all variables with quantifier symbol max. Without loss of generality, consider what happens when all are assigned zero. If the minimizer also assigns all variables zero, the sum of the terms is  $v_0$  by definition. Now consider a maximal subsequence of variables  $z_i \dots z_j$  such that  $z_\ell$  is assigned one for  $i \leq \ell \leq j$ . Because the  $z_\ell$  have been assigned one, we know they belong to the minimizer. Because  $z_1$  belongs to the maximizer, we know that  $i > 1$  and that there is a variable  $z_{i-1}$ . The term  $f_{i-1}(z_{i-1}, z_i)$  thus returns value 1 ( $z_i = 1, z_{i-1} = 0, z_i$  belongs to min). Terms  $f_\ell(z_\ell, z_{\ell+1})$  for  $i \leq \ell < j$  return 0 and  $f_j(z_j, z_{j+1})$  (which exists only if  $j < k$ ) returns 0. Altogether, these terms add up to the same as before (if  $j < k$ ) or one more than before (if  $j = k$ ). Thus the minimizer can only lose by making some variable one.

Now consider the minimizer strategy which sets all variables equal to the value assigned to  $z_1$ . This strategy is possible because Definition 8(3) says  $z_1$  precedes the other  $z_i$ . An argument similar to the previous paragraph implies that this strategy assures that the minimizer pays no more than  $v_0$ . Thus the result  $v_0$  can be guaranteed by both players and thus the value of  $(Q, P)$  is exactly  $v_0$ .

In the game  $(Q', P)$ , the maximizing player can set all her variables to be the opposite of  $z_k$ , now possible because  $z_k$  is set before all others. The minimizing player now has some variables set differently than  $z_1$  including  $z_k$  and, by the above argument, the result must be one greater than  $v_0$ . Thus  $(Q, P)$  and  $(Q', P)$  have different values.

If a term  $t_i$  has more than the two arguments, simply define the function to be determined (as above) just by the arguments associated with  $z_i$  and  $z_{i+1}$  and to be unaffected by the settings of the other variables. If several of the terms are actually the same term, associate that term with the function equal to the sum of the corresponding functions used in the proof. If  $P$  contains a term other than the  $f_i$ , associate that term with a constant function. If  $Q$  has more than two quantifier symbols, make the other quantifier symbols (that is, other than the quantifier symbols for  $z_1$  and  $z_k$ ) equal to quantifier symbols which behave like (but have different names than) max or min. These changes do not affect the results of the strategies considered above. ■

**Example:** Let  $Q = \max_w \min_x \max_y \min_z$  and let  $Q' = \min_x \max_w \max_y \min_z$ . Let  $P = \{f(w, z), g(x, y, z)\}$ . The influence relations for  $(Q, P)$  and  $(Q', P)$  are the same except that  $w \prec x$  but  $x \prec' w$ . The influence sequence from  $w$  to  $x$  in  $(Q, P)$  is  $w, z, y, x$  and the reverse sequence  $x, y, z, w$  is the influence sequence from  $x$  to  $w$  in  $(Q', P)$ . As suggested by the proof, we let  $f(w, z) = 0$  if  $w = x$  and  $f(w, z) = 1$  otherwise. We let  $g(x, y, z) = g_1(z, y) + g_2(y, x)$  where  $g_1(z, y) = 1$  if  $z = y$  and  $g_1(z, y) = 0$  otherwise and where  $g_2(y, x) = 0$  if  $y = z$  and  $g_2(y, x) = 1$  otherwise. This makes the terms into  $\mathcal{I}$ -terms and it can be verified that  $VALUE(Q, P) = 1$  but

$VALUE(Q', P) = 2$ .

It should be noted that the formulas  $(Q, P)$  and  $(Q', P)$  above cannot be made different if we replace  $\max$  with  $\exists$ ,  $\min$  with  $\forall$  and attempt to make the terms map into the monoid  $\mathcal{B}_\wedge = (\{\text{TRUE}, \text{FALSE}\}, \wedge, \text{TRUE})$ . The order of  $w$  and  $x$  is irrelevant in this case because

$$\exists w \forall x \exists y \forall z (f(w, z) \wedge g(x, y, z)) = \forall x \exists w \exists y \forall z (f(w, z) \wedge g(x, y, z))$$

for all Boolean valued functions  $f$  and  $g$ . (Because  $\wedge$  commutes with  $\forall$ , both expressions are equal to  $(\exists w \forall z f(w, z)) \wedge (\forall x \exists y \forall z (g(x, y, z)))$ .) There is a concept of influence which is more suitable for quantified Boolean formulas. This is studied in Section 8.

An open question suggested by Theorem 11 is this: what other monoids and quantifiers also satisfy the theorem? Our proof makes use of the fact that the monoid is on an infinite set. Are there any monoids on a finite set which also satisfy the theorem? If we are allowed to pick a monoid after the formulas are given, then a finite monoid can be picked as described in the following corollary:

**COROLLARY 1.** *Let  $P$  be a set of terms such that each term has a distinct base symbol. Let  $Q$  and  $Q'$  be quantifier lists such that  $Q$  and  $Q'$  are permutations of each other. If quantified formulas  $(Q, P)$  and  $(Q', P)$  have different influence relations, then there is a finite monoid  $\mathcal{M} = (S, \oplus)$  and an association of base symbols with  $S$ -valued functions and quantifier symbols with  $\max$  and  $\min$  such that quantified  $\mathcal{M}$ -formulas  $(Q, P)$  and  $(Q', P)$  have different values.*

*Proof.* Let  $z_0, \dots, z_k$  be as in the proof of Theorem 11, let  $S$  be the set of integers from 0 to  $k$ , and let  $\oplus$  be defined by  $a \oplus b = \min\{a + b, k\}$ . Clearly  $\mathcal{M} = (S, \oplus, 0)$  is a commutative monoid and  $\max$  and  $\min$  defined in the usual way are quantifiers for  $\mathcal{M}$ . The proof of Theorem 11 goes through as given using  $\mathcal{M}$  instead of  $\mathcal{I}$ . ■

## 7. LOCAL REPLACEMENT, LOCAL REDUCTIONS, AND HIERARCHICAL SPECIFICATIONS

A quantified formula with free variables represents a function from the domains of the free variables into the formula's monoid, as indicated in Definition 6. Such a formula can be used to define the function associated with the base symbol of a term (Definition 4). In this section, we show how such defining formulas can be used as macros to replace terms in a formula by their definitions. This is done by replacing free variables (i.e., formal parameters) of the definition with the corresponding arguments of the term (i.e. the actual parameters) and by appending the quantifier list from the defining formula to the quantifiers of the formula being expanded.<sup>9</sup>

<sup>9</sup>We always assume without loss of generality that the variables of the definition and the formula are disjoint.

We refer to the replacement of a term by its definition as a “local replacement.” We refer to the local replacement of all terms as a “local reduction”. In finite model theory [15], defining formulas of the type described here are known as “syntactical interpretations” and the local reductions used here are examples of “logical reductions”. Although we are giving local reductions a very brief treatment here, we have found evidence in our other work that having a formal way of treating local reductions is very useful and that, when taking an algebraic view of problem specifications, local reductions can be viewed as a natural algebraic extension of homomorphisms. We also discuss briefly the application of local replacement to hierarchical specifications.

In this section, it is shown that

1. a formula obtained by local replacement has the same value as the original formula (Lemma 5),
2. a “good” structure tree for a formula and a “good” structure tree for a replacement rule can be combined to get a “good” structure tree for the transformed formula (Theorem 12) and
3. local reductions preserve structure (Corollary 2).

LEMMA 5. *Let  $\mathcal{M} = (S, +, 0)$  be a commutative monoid, let  $F_1 = (Q_1, P_1)$  be a fully quantified  $\mathcal{M}$ -formula, and let  $t = f(x_1, \dots, x_k)$  for some  $k$  be a term in  $P$ . Let  $F_2 = (Q_2, P_2)$  be a quantified  $\mathcal{M}$ -formula with free variables  $z_1, \dots, z_k$  such that  $f(z_1, \dots, z_k)[\gamma] = \text{VALUE}(F_2[\gamma])$  for all  $\gamma$  in  $\Gamma[\text{FREE}(F_2)]$  and such that  $\text{VAR}(F_1) \cap \text{VAR}(F_2) = \emptyset$ . Let  $F_3 = (Q_3, P_3)$  where  $Q_3 = Q_1 Q_2$  and let  $P_3 = (P_1 - \{t\}) \cup P_2[\gamma_0]$  where  $\gamma_0$  is the assignment<sup>10</sup> which replaces each  $z_i$  by  $x_i$ . Then  $\text{VALUE}(F_1) = \text{VALUE}(F_3)$ .*

*Proof.* Because  $\text{VAR}(Q_2) \cap \text{VAR}(P_1) = \emptyset$  and because of Lemma 1, the values of both formulas are equal to the value of the expression  $Q_1(\sum_{p_1 \in (P_1 - \{t\})} p_1 + Q_2(\sum_{p_2 \in P_2} p_2[\gamma_0]))$ . ■

THEOREM 12. *Let  $F_1$ ,  $F_2$ , and  $F_3$  be as in Lemma 5. Let  $CW_i$  and  $WD_i$  be the channelwidth and weighted depth of  $F_i$  for  $i = 1, 2, 3$ . Then*

1.  $CW_3 \leq \max\{CW_1, CW_2\}$ ,
2.  $WD_3 \leq WD_1 + WD_2$ .

*Proof.* Let  $S_1 = (T_1, \alpha_1, \beta_1)$  and  $S_2 = (T_2, \alpha_2, \beta_2)$  be structure trees for  $F_1$  and  $F_2$  respectively. To get a structure tree  $S_3 = (T_3, \alpha_3, \beta_3)$  for  $F_3$ , combine  $T_1$  and  $T_2$  into tree  $T_3$  by making the root of  $T_2$  be a child of  $\beta_1(t)$  and define  $\alpha_3$  and  $\beta_3$  have the same values as  $\alpha_1$ ,  $\alpha_2$ ,  $\beta_1$ , and  $\beta_2$ .

<sup>10</sup>The assignment notation is extended here in the obvious way to allow the replacement of variables by variables.

That is,  $\alpha_3(x) = \alpha_1(x)$  for  $x \in VAR(Q_1)$ ,  $\alpha_3(x) = \alpha_2(x)$  for  $x \in VAR(Q_2)$ ,  $\beta_3(p) = \beta_1(p)$  for  $p \in P_1$ , and  $\beta_3(p[\gamma_0]) = \beta_2(p)$  for  $p \in P_2$ .

We must verify that  $S_3$  satisfies Definition 9. Conditions 1, 2, and 3 are true by construction. Condition 6 is vacuously true (and would be true even if  $F_1$  did have free variables.) It remains to verify Conditions 4 and 5.

Let  $p$  be a term in  $P_3$  and let  $y$  be a variable in  $VAR(p)$ . If  $p \in P_1$  and  $y \in VAR(Q_1)$ , then  $\alpha_3(y)$  is above  $\beta_3(p)$  because the same is true in  $S_1$ . If  $p \in P_2[\gamma_0]$  and  $y \in VAR(Q_2)$ , then  $\alpha_3(y)$  is above  $\beta_3(p[\gamma_0])$  because the same is true in  $S_2$ . The only other case is that  $p \in P_2[\gamma_0]$  and  $y \in VAR(Q_1)$ . In this case,  $\alpha_1(y)$  is above  $\beta_1(t)$  which is above the root of  $T_2$  which is above  $\beta_3(p)$ .

Finally, we must show that Condition 5 holds. Suppose that  $x \prec y$  in  $F_3$  and let  $z_1 \dots z_k$  be as in Definition 8. We need to prove that  $\alpha_3(x)$  is an ancestor of  $\alpha_3(y)$ . There are four cases to consider:

Case  $x$  and  $y$  in  $VAR(Q_2)$ : Since  $Q_2$  comes after  $Q_1$  in  $Q_3$ , all the  $z_i$  must be in  $VAR(Q_2)$  (Definition 8(3)) and the terms connecting the  $z_i$  must be from  $P_2[\gamma_0]$ . Therefore  $x \prec y$  in  $S_2$  and  $\alpha_3(x) = \alpha_2(x)$  is an ancestor of  $\alpha_3(y) = \alpha_2(y)$ .

Case  $x$  and  $y$  in  $VAR(Q_1)$ : In this case, some of the  $z_i$  may be from  $VAR(Q_2)$ . We claim that deleting these  $z_i$  from the sequence results in an influence sequence for  $F_1$ .

If  $z_i$  is from  $VAR(Q_1)$  and either  $z_{i-1}$  or  $z_{i+1}$  is from  $VAR(Q_2)$ , then  $z_i$  must be a variable in some term from  $P_2[\gamma_0]$  and hence  $z_i$  in  $VAR(t)$ . Thus if  $z_i$  and  $z_j$  become adjacent after removing variables from  $VAR(Q_2)$ ,  $\{z_i, z_j\} \subset VAR(t)$  and Definition 8(2) is satisfied for the modified sequence. The other conditions are easily verified and so  $x \prec y$  in  $F_1$ . Thus  $\alpha_3(x) = \alpha_1(x)$  is an ancestor of  $\alpha_3(y) = \alpha_1(y)$ .

Case  $x$  in  $VAR(Q_1)$  and  $y$  in  $VAR(Q_2)$ : There must be a largest  $i$  such that  $z_i$  is in  $VAR(Q_1)$ . As in the previous case,  $z_i$  is in  $VAR(t)$ . If  $i = 1$ , then  $\alpha_3(x) = \alpha_1(x) = \alpha_1(z_1)$  is an ancestor of  $\beta_1(t)$  which is an ancestor of all  $p[\gamma_0]$  in  $P[\gamma_0]$ . If  $i \neq 1$ , then  $z_i$  must have the same quantifier as  $y$  (otherwise Definition 8(4) is violated) and it is easily verified that  $z_1 \dots z_i$  is an influence sequence for  $S_3$ . From the previous case, we know that  $x \prec z_i$  in  $F_1$  and hence  $\alpha_3(x) = \alpha_1(x)$  is an ancestor of  $\alpha_1(z_i)$  which is an ancestor of  $\beta_1(t)$  which is an ancestor of  $\alpha_3(y)$ .

Case  $x$  in  $VAR(Q_2)$  and  $y$  in  $VAR(Q_1)$ : This implies  $y$  comes before  $x$  in  $F_3$  (Definition 8(3)) which contradicts the construction  $Q_3 = Q_1Q_2$ .

Having now shown that  $S_3$  is a structure tree for  $F_3$ , we must prove the claims for channelwidth and weighted depth. The channel variables for nodes from  $S_1$  remain the same and the channel variables for nodes from  $S_2$  are just the old channel variables with the free variables replaced with the corresponding variable from  $\gamma_0$ . Thus  $CW_3 \leq \max(CW_1, CW_2)$ . The result for weighted depth is even more immediate. ■

The theorem implies that “local reductions preserve structure” in the

following sense:

**COROLLARY 2.** *Let  $H = \{f_1, \dots, f_k\}$  be a finite set of  $k$  finite domain functions and let  $F_1, \dots, F_k$  be quantified formulas with free variables such that formula  $F_i$  defines  $f_i$ ,  $1 \leq i \leq k$ . There exist constants  $c$  and  $w$  such that, if  $CW$  and  $WD$  are the channelwidth and weighted depth of some formula  $F$  with terms constructed from functions in  $H$ , the the formula  $F'$  obtained from  $F$  using the  $F_i$  for local replacement has channelwidth at most  $\max\{CW, c\}$  and weighted depth at most  $WD + w$ .*

*Proof.* Let  $c$  be the maximum channelwidth of the  $F_i$ . The result on channel width is immediate from the Theorem 12. Let  $w$  be the maximum weighted depth of the  $F_i$ . Because the application of the  $F_i$  are not nested, the proof of Theorem 12 implies the result for weighted depth. ■

In Section 8 of [34], hierarchically specified formulas were defined by nested (but not recursive) replacement rules. Such a description is capable of describing a formula whose size is exponential in the size of the description. Any such formula can be evaluated with a number of operations that is exponential only in the size of the description. In [34], structure trees are defined for descriptions. Lemma 5 and Theorem 12 above are generalizations of key ideas from [34]. The main result (Theorem 8.8 in [34]) then generalizes, namely the value of a hierarchically specified formula can be found in time exponential only in the channelwidth of the description. Any local reduction for formulas can also be used in the obvious way to reduce one hierarchical specification to another. Corollary 2 “lifts” to hierarchically specified formulas as follows:

**COROLLARY 3.** *The result of Corollary 2 holds when the  $F_i$  are used to transform a hierarchical specification of  $F$  into a hierarchical specification of  $F'$ .*

*Proof.* This follows at once from Corollary 2 and the definitions of hierarchical specifications from [34] extended to quantified formulas. ■

Thus local reductions map both ordinary formulas and hierarchically specified formulas into formulas of similar structures which can be used to solve the instances in comparable times. In contrast, a resource bounded reduction generally neither maps a formula into a formula with similar structure nor extends to a reduction between hierarchical specifications.

## 8. THE $\tau$ -QUANTIFIER SYMBOL

The influence concept is used to place restrictions on which variables may be above others in a structure tree. It is imperative that this concept place enough restrictions so that Theorem 3 is true. Otherwise, the generic algorithms would fail. Beyond this requirement, restrictions are undesirable since each structure tree is a potential opportunity for speeding up

the evaluation of a formula. In general, the influence concept cannot be improved upon because of Theorem 11. However, if something is known about the semantics of a quantifier symbol, it may be possible to define a weaker influence relation which is still strong enough to prove Theorem 3 and other main theorems. This happens if one of the quantifier symbols is the “ $\tau$ -quantifier symbol” defined in Definition 2. This is an important case because  $\forall$  and  $\exists$  sometimes behave as  $\tau$ -quantifier symbols (see Proposition 2(6).)

Here we sketch how the results and proofs can be changed to allow for  $\tau$ -quantifier symbols. Most changes are consequences of the following:

**PROPOSITION 10.** *Let  $\mathcal{M} = (S, +, 0)$  be an idempotent commutative monoid,  $x$  a variable, and  $F(x)$  and  $G(x)$  functions to  $S$ . Then  $\tau_x(F(x) + G(x)) = (\tau_x F(x)) + (\tau_x G(x))$ .*

*Proof.* Obvious. ■

For  $\mathcal{B}_\wedge$  and  $\mathcal{B}_\vee$ , Proposition 10 yields the familiar  $\forall x(F(x) \wedge G(x)) = (\forall x F(x)) \wedge (\forall x G(x))$  and  $\exists x(F(x) \vee G(x)) = (\exists x F(x)) \vee (\exists x G(x))$  respectively. In a sense,  $F(x)$  and  $G(x)$  are “independent subproblems” even though they share a variable. After evaluating the subproblems independently, we are able to combine the computed values. The proposition enables a stronger version of Theorem 1:

**PROPOSITION 11.** *If the monoid in Theorem 1 is idempotent, then condition 2 of the theorem can be changed to*

2.  $P$  can be partitioned into two sets  $P_1$  and  $P_2$  such that  $x \notin P_2, y \notin P_1$ , and all variables in  $\text{VAR}(Q) \cap \text{VAR}(P_1) \cap \text{VAR}(P_2)$  have quantifier symbol  $\tau$ .

*Proof.* Same as the proof of Theorem 1 except variables with quantifier symbol  $\tau$  are moved down onto both subexpressions using Proposition 10.

■

To explain how to exploit the extra swapping implied by Proposition 11, we define a new kind of influence relation:

**DEFINITION 14.** Let  $F = (Q, P)$  be a quantified formula for an idempotent commutative monoid. Let  $m_x$  and  $\overline{m}_y$  be as in Definition 8. We write  $x \prec_\tau y$  if  $x$  and  $y$  satisfy Definition 8 with the following additional condition:

5. variables  $z_i$  for  $1 < i < k$  do not have quantifier symbol  $\tau$ .

Relation  $\prec_\tau$  is called  **$\tau$ -influence**. A structure tree based on Definition 10 using the  $\tau$ -influence relation  $\prec_\tau$  instead of  $\prec$  is called a  **$\tau$ -structure tree for  $F$** .

Note that  $x \prec_\tau y$  implies  $x \prec y$  so a structure tree for a quantified formula is always a  $\tau$ -structure tree. The following theorem insures that the generic algorithms for the special case under consideration will always give the correct answer.

**THEOREM 13.** *Let  $\mathcal{M} = (S, +, 0)$  be a commutative idempotent monoid. Then Theorem 2 holds for  $\tau$ -structure trees.*

*Proof.* The proof begins like the proof of Theorem 2 until the two cases are considered. Then the cases are slightly modified:

Case 1: Same as Case 1 from Theorem 2 with the extra condition that the variables  $z_i$  for  $1 < i < k$  have quantifier symbol different from  $\tau$ . The argument is the same except it is now seen that  $z_i \dots z_k$  is a  $\tau$ -influence sequence and so  $z_i \prec_\tau y$  and Case 1 is again impossible.

Case 2: When Case 1 fails,  $P$  can be partitioned into sets  $P_x$  and  $P_y$  as in the proof of Theorem 2. This time, a term from  $P_x$  and a term from  $P_y$  may have common variable with quantifier symbol  $\tau$ . This partition satisfies Proposition 11(2) and thus the interchange is valid. ■

Next we want to show that the nice properties of  $\prec$  from Section 6 also hold for  $\prec_\tau$ :

**THEOREM 14.** *Lemmas 4 and Theorems 8, 9 and 10 all hold for  $\tau$ -influence and  $\tau$ -structure trees.*

*Proof.* All the arguments about influence sequence go through with the observation that if  $z_1 \dots z_k$  satisfies Definition 14(5), then so does any subsequence. ■

Now we present a result for  $\tau$ -influence analogous to Theorem 11. Here we use the monoid  $\mathcal{B}_\wedge = (\{\text{TRUE}, \text{FALSE}\}, \wedge, \text{TRUE})$  and quantifier symbols  $\forall$  and  $\exists$  with  $\forall$  being the  $\tau$ -quantifier symbol (see Proposition 2(5)). A formula constructed from this monoid and these quantifier symbols can be interpreted as a zero-sum two-person game where the  $\forall$  and  $\exists$  players make assignments to their respective variables following the order of quantification. The  $\exists$  player wins if and only if the resulting assignment makes all the terms true.<sup>11</sup> As in the proof of Theorem 11, we make use of certain game theory facts without proof.

**THEOREM 15.** *Let  $P$  be a set of terms such that each term has a distinct base symbol. Let  $Q$  and  $Q'$  be quantifier lists such that  $Q$  and  $Q'$  are permutations of each other and each variable is quantified with one of two quantifier symbols. Suppose further that one of the quantifier symbols is designated as a  $\tau$  quantifier symbol. If quantified formulas  $(Q, P)$  and  $(Q', P)$  have different  $\tau$ -influence relations, then there is an association of base symbols with Boolean valued functions and quantifier symbols with  $\forall$*

<sup>11</sup>Player  $\exists$  can be regarded as either max or min depending on which ordering we impose on  $\{\text{TRUE}, \text{FALSE}\}$  (see Proposition 2(1 and 2)).

and  $\exists$  with  $\forall$  being the  $\tau$ -quantifier symbol such that quantified  $\mathcal{B}_\wedge$ -formulas  $(Q, P)$  and  $(Q', P)$  have different values.

*Proof.* From Lemma 4 and Theorem 14, we know there is a  $\tau$ -influence sequence  $z_1 \dots z_k$  for  $(Q, P)$  such that  $z_k \dots z_1$  is a  $\tau$ -influence relation for  $(Q', P)$ . Without loss of generality, assume  $z_1$  has quantifier symbol  $\forall$  and  $z_k$  has quantifier symbol  $\exists$ . For the same reason as in the proof of Theorem 11, we can assume that  $P$  consists of terms  $f_i(z_i, z_{i+1})$  for  $1 \leq i < k$ .

We let all the  $f_i$  be the function  $f$  defined by  $f(x, y) = \text{TRUE}$  if  $x = y$  and  $\text{FALSE}$  otherwise. The conjunction of these terms will be  $\text{TRUE}$  if and only if all the  $z_i$  are assigned the same value.

In  $(Q, P)$ , the  $\forall$  or  $\tau$  player first chooses the assignment for  $z_1$  and then the  $\exists$  player (in control of all other  $z_i$  by Definition 14(5)) sets all the other  $z_i$  to be equal to  $z_1$ . Thus the value of the formula is  $\text{TRUE}$ .

In  $(Q', P)$ , the  $\exists$  player must first choose the assignment for  $z_k$  and the  $\forall$  player can make the conjunction  $\text{FALSE}$  by choosing  $z_1$  to be different from  $z_k$ . Thus the value of the formula is  $\text{FALSE}$  and the two formulas have different values. ■

## 9. FINDING GOOD STRUCTURE TREES

In this section we discuss briefly how to find good structure trees. First we discuss how the influence relation can be computed in  $O(n^3)$  time. To do this, we need a preliminary binary relation  $B$ .

**DEFINITION 15.** Given a formulas  $F = (Q, P)$ , binary relation  $B$  on  $\text{VAR}(Q)$  is defined inductively as follows:

1.  $xBy$  for all  $x$  and  $y$  such that  $x <_Q y$  and  $\{x, y\} \subset \text{VAR}(p)$  for some  $p \in P$ ;
2. if  $xBz$ ,  $yBz$ , and  $x <_Q y$ , then  $xBy$ .

Relationship  $xBy$  can be read as “ $x$  borders  $y$ ” meaning that  $x$  is assigned before  $y$  and  $x$  belongs to the subproblem containing  $y$  that results when all variables before  $y$  in  $Q$  have been assigned. In other words, assigned variable  $x$  is still connected to  $y$  via unassigned variables when all variables before  $y$  have been assigned. The transitive closure of  $R$  satisfies Conditions 1, 2, and 3 of Definition 8 and thus contains  $\prec$ . We extract  $\prec$  from  $R$  with rules similar to but more restrictive than transitive closure.

**DEFINITION 16.** Given a formula  $F = (Q, P)$ , binary relation  $\prec$  on  $\text{VAR}(Q)$  is defined inductively as follows:

1.  $x \prec y$  for all  $x$  and  $y$  such that  $xBy$  and  $x$  and  $y$  have different quantifier symbols,
2. if  $x \prec z$ ,  $zBy$ , and  $z$  and  $y$  have the same quantifier symbol, then  $x \prec y$ .

PROPOSITION 12. *The relations  $\prec$  given in Definitions 8 and 16 are the same.*

*Proof.* Proof is straightforward. ■

The above inductive definitions allow influence to be computed in  $O(n^3)$  time using three nested loops in a way similar to Warshall's Algorithm (while taking quantifier order into account). One first computes  $B$  this way and then computes  $\prec$  similarly. We omit further details.

Several algorithms for finding good structure trees in the unquantified case involve enumerating and testing certain variable orderings hoping to find an ordering that gives good weighted depth or channelwidth. These algorithms include the basic top-down approach discussed in [34](p. 461) and the basic bottom up approach used in [31]. When these basic procedures are given a bound  $k$  on weighted depth or on channelwidth (or equivalently on the treewidth of the underlying graph), they run in time exponential only in  $k$  (which means polynomial time when  $k$  is fixed). These algorithms can be modified in a straightforward way to consider only orderings that respect the influence relation precomputed as above. It is possible that this idea also applies to the more sophisticated algorithms more recently developed (see [8] for some references).

## 10. RELATIONSHIP BETWEEN WEIGHTED DEPTH AND CHANNELWIDTH

The channelwidth of any structure tree cannot be more than the weighted depth because every channel variable is a branch variable. We show that the weighted depth can be equal to the number of variables even when the channelwidth is 2. This is in sharp contrast to the one quantifier case [34] where the weighted depth of a formula is no more than channelwidth times the log of the number of variables.

THEOREM 16. *For all  $n$ , there exists a quantified formula  $F = (Q, P)$  where  $WD(F)$  and  $|F|$  are  $\Theta(n)$  and  $CW(F) = 2$ .*

*Proof.* Let  $Q = m_{x_0}^0 \cdots m_{x_n}^n$  where  $m^i = m$  if  $i$  is odd and  $m^i = \bar{m}$  if  $i$  is even and  $m \neq \bar{m}$ . Let  $P = \{f_i(x_{i-1}, x_i) \mid 1 \leq i \leq n\}$ . Let  $F = (Q, P)$ . Observe that  $x_{i-1} \prec x_i$  for  $1 < i \leq k$  by the length two influence sequence  $x_{i-1}x_i$ .

We construct a structure tree  $(T, \alpha, \beta)$  for  $F$  as follows: Let the nodes of  $T$  be  $v_i$  for  $0 \leq i \leq n$  and let  $v_0$  be the root. Let the edges be  $(v_{i-1}, v_i)$  for  $1 \leq i \leq n$ . Let  $\alpha(x_i) = v_i$  for  $0 \leq i \leq n$  and  $\beta(f_i(x_{i-1}, x_i)) = v_i$ . Clearly  $CV(v_i) = \{x_{i-1}, x_i\}$  for  $1 \leq i \leq n$  and  $CV(v_0) = \{x_0\}$  so  $CW(F) = 2$ .

Now consider an arbitrary structure tree  $(T, \alpha, \beta)$  for  $F$ . For  $0 \leq i \leq n$ , let  $v_i = \alpha(x_i)$ . (The  $v_i$  are not necessarily distinct.) Consider any  $j$ ,  $1 \leq j \leq n$ . Nodes  $v_{j-1}$  and  $v_j$  are both ancestors of  $\beta(f_j(x_{j-1}, x_j))$  by

Definition 9(4) and are thus both are on the same branch of  $T$ . Furthermore,  $v_{j-1}$  must be an ancestor of  $v_j$  by Definition 9(5). It follows that all the  $v_i$  are ancestors of  $v_n$ . Thus all the  $v_i$  are on the same branch of  $T$  and  $WD(T) = n$ . ■

## 11. UNQUANTIFIED CHANNELWIDTH

Can anything about the complexity of a quantified formula be inferred from its unquantified channelwidth? (By “unquantified channelwidth, we mean the channelwidth of structure trees defined without Condition 5 of Definition 9.) Here we give strong evidence that, in the general case, the answer is “no”.

Consider the following problem:

INSTANCE: Fully quantified formula  $F = (Q, P)$  for the monoid of integers and an integer  $K$  where the variables in  $Q$  have domains  $\{0, 1\}$  and the terms in have the form  $f_w(x, y)$  where  $w$  is an integer and  $f_w(x, y)$  has value  $w$  if  $x = y$  and value 0 otherwise.

QUESTION: Is  $VALUE(F) \leq K$ ?

The next result tells us that this problem is hard, even when the channelwidth of the unquantified formula and the number of alternations are both bounded.

**THEOREM 17.** *The above problem is NP-HARD, even when restricted to formulas  $F$  where the unquantified formula  $(VAR(Q), P)$  has channelwidth and weighted depth 2.*

*Proof.* Let  $w_1 \cdots w_n$  be an instance of PARTITION. We map this to a quantified formula with variables  $x_1, \dots, x_n$  and  $y$  as follows: Let  $Q = \min_{x_1} \cdots \min_{x_n} \max_y$ , let  $P = \{f_{w_1}(x_1, y), \dots, f_{w_n}(x_n, y)\}$ , let  $F = (P, Q)$ , and let  $K = (\sum w_i)/2$ .

To see that this map is a reduction, it is easiest to interpret  $F$  as a game. First the minimizing player assigns each  $w_i$  to one of two sets  $S_0$  or  $S_1$  by setting the corresponding variable  $x_i$  to either 0 or 1. Then the maximizing player sets variable  $y$ . Her payoff will be the sum of the weights in  $S_0$  if she picks 0 and the sum of weights in  $S_1$  if she picks 1. The optimal strategy for the minimizing player is to divide the weights as evenly as possible. If the weights can be partitioned into two equal sets, then the value of the game is  $K$  and the answer to the question is “yes”. If they cannot be so partitioned, one set must sum to an amount greater than  $K$  and the answer is “no”.

Next we construct a structure tree  $(T, \alpha, \beta)$  for the unquantified formula  $(VAR(Q), P)$ . Let the nodes of  $T$  be  $v_0, \dots, v_n$  where  $v_0$  is the root and the other nodes are leaves. Let  $\alpha(x_i)$  and  $\beta(f_i(x_i, y))$  both equal  $v_i$  for  $1 \leq i \leq n$  and let  $\alpha(y) = v_0$ . This is obviously an unquantified structure tree of channelwidth and weighted depth 2. ■

In contrast, the formulas involved have the highest possible (quantified) channelwidth, as the next result indicates:

**PROPOSITION 13.** *All structure trees for the formula  $F = (Q, P)$  from the proof of Theorem 17 have weighted depth and channelwidth  $|\text{VAR}(Q)|$ .*

*Proof.* Observe that  $x_i \prec y$  for all  $1 \leq i \leq k$ . Given any structure tree  $(T, \alpha, \beta)$  for  $F = (Q, P)$  and given any  $i$ ,  $1 \leq i \leq n$ ,  $\alpha(x_i)$  must be above  $\alpha(y)$  (Definition 9(4)) and  $\alpha(y)$  is above  $\beta(f_i(x_i, y))$  by Definition 9(4). Therefore all  $x_i$  are channel variables at  $\alpha(y)$  by Definition 10(6). Thus set  $CV(\alpha(y))$  contains all variables and  $WD = CW = n + 1$ . ■

The above does not rule out all possible uses of unquantified structure in quantified situations. In fact, the literature does have examples involving quantifiers where bounded treewidth is sufficient to produce polynomial algorithms. (Treewidth, a graph theory concept, is tightly coupled to the channelwidth of unquantified formulas.) This is done in a very general way in [4, 11].

The above also does not rule out solving special cases by methods unrelated to subproblem independence. A contributing factor to the hardness proof of Theorem 17 was the fact that an infinite set of functions was available. If we limit the available functions to a finite set of  $k$ , the formulas can be evaluated in polynomial time using the dynamic programming concepts used to show that PARTITION is not strongly NP-complete.

## APPENDIX A: SAMPLE PROBLEMS

Here we illustrate techniques for representing problems as sets of formulas using some familiar problems as examples. In what follows, we use the terminology “Boolean valued term” to refer to an expression on finite domain variables which describes a function from variable assignments to  $\{\text{TRUE}, \text{FALSE}\}$ . In many applications, the terms have the form “ $f(x, y, z)$ ” where  $f$  is from some set  $S$  of finite arity functions. Often  $S$  is finite. When the functions are Boolean-valued, functions from  $S$  can also be regarded as representing relations, namely the set of assignments which make the function TRUE. For some applications, we reinterpret such terms as being  $\{0,1\}$ -valued where TRUE is interpreted as the number 1 and FALSE as the number 0. In all one quantifier applications, the quantified variables can be listed in any order. As discussed in the introduction, the one quantifier examples are also sums-of-products and generalized satisfiability problems (GSPs) as defined in [34].

### A.1. Applications of the Boolean Monoid $\mathcal{B}_\wedge$

Using  $\mathcal{B}_\wedge$  (see Definition 1(1)), one can represent any satisfiability problem and constraint satisfaction problem by quantifying all variables with

$\exists$  and constructing terms from some appropriate set of Boolean-valued function symbols. More specifically, one can represent the satisfiability problem for CNF boolean formulas (sometimes called **SAT**) and the problems **SAT**( $S$ ) of [33] where  $S$  is a finite set of finite arity Boolean-valued functions.

Noteworthy special cases are bounded bandwidth problems as in [25] because they have bounded treewidth and the planar problems of [22], [16] or [19] or the  $\delta$ -near planar problems of [30] because they have treewidth bounded by  $2^{O(\sqrt{n})}$ .<sup>12</sup>

Using the two quantifier symbols  $\exists$  and  $\forall$ , one can define quantified CNF formulas (sometimes called **QSAT**) or **QSAT**( $S$ ) as defined by Schaefer [33]. Formulas of this type can be thought of as defining two-player games where one player controls the variables quantified with  $\exists$  and wants the conjunction of terms to be **TRUE** and the other player controls the variables quantified with  $\forall$  and wants the conjunction to be **FALSE**. The rules for the influence relations complicate the application of bounded bandwidth or planarity to such quantified formulas.

## A.2. Applications of Ordinary Addition

Using a monoid of numerical values<sup>13</sup> under ordinary addition, one can quantify all variables with **min** and model all non-serial optimization problems as given in [32]. One can also maximize the sum instead of minimizing it.

A special case is **MAX Satisfiability**. (Given a set of Boolean valued terms, what is the maximum number of terms that can be simultaneously satisfied?) To model this, reinterpret the terms as  $\{1,0\}$ -valued and use the monoid of natural numbers under addition and the quantifier **max**. All general satisfiability problems have a corresponding **max** problem. See the discussion of **MAX SNP** in [28].

When the two quantifiers **max** and **min** are both used, the value of the formula is the value of the two-person game where one player controls the variables quantified with **max** and wants the sum of terms to be large and the other player controls the variables quantified with **min** and wants the sum to be small. By adding the stochastic quantifier of [27] (see Definition 2(4)) one adds a third player (known as nature) who assigns values randomly.

The two person game specializes to the problem known as **MAX Quantified SAT** (see **MAX QSAT** in [9, 13]) when the terms are reinterpreted Boolean-valued terms. Because of the minmax theorem for zero-sum games,

<sup>12</sup>This assumes that terms have bounded arity. The structure tree can be found by a recursive application of the Planar Separator Theorem from [23].

<sup>13</sup>The numerical values must of course be from some computationally manageable set such as the integers or rationals.

this problem could (from the perspective here) also be called MIN Quantified Satisfiability.

### A.3. Constrained Optimization

By “constrained optimization”, we mean minimize (or maximize) the sum of numerical terms subject to the restriction that a set of Boolean-valued terms are satisfied. The minimization problem can be converted to **non-serial optimization** by putting  $\infty$  into a numerical-valued monoid of ordinary addition. Each constraint is reinterpreted as having value zero when the constraint is satisfied and the value  $\infty$  when the constraint is unsatisfied. Using the one quantifier symbol  $\min$ , the sum takes on the value of the minimum sum when the constraints are satisfiable and  $\infty$  when they are not satisfiable. As discussed in [34], one can treat the numerical and Boolean terms as a single set and exploit the structure with the generic methods, but the structures of the numerical and Boolean terms considered separately are not useful (assuming  $P \neq NP$ ). It is not clear how one would define or model constrained multi-quantifier problems.

### A.4. Applications of Ordinary Multiplication

Monoids of non-negative numerical values under multiplication support the  $\sigma$  quantifier associated with ordinary addition and the quantifiers  $\min$  and  $\max$ . The  $\sigma$  quantifier by itself can be used to count the number of satisfying assignments to a set of Boolean valued terms. By interpreting the Boolean terms as one-zero terms, the product of terms is one if all the terms are **TRUE** and is zero if any of the terms is **FALSE**. Thus by quantifying the variables with the  $\sigma$  quantifier, the formula gives the number of assignments. These problems are called  $\#$ -problems. See [35, 16] for discussion of  $\#\text{SAT}$  and the concept of  $\#P$ -hardness.

The combination of the  $\sigma$  and  $\min$  quantifiers describes a game-like situation where an agent controlling the  $\min$  variables wants to minimize the number of possible ways that the agent controlling  $\sigma$  can make all the terms satisfied.

## REFERENCES

- [1] Aho A.V., Hopcroft, J.E., and Ullman, J.D. (1974), “The Design and Analysis of Computer Algorithms”, Addison-Wesley
- [2] Arnborg A., Corneil, D.G., Proskurowski, A. (1987), Complexity of finding embeddings in a k-tree, *SIAM J. Alg. and Discr. Methods* **8**, 277–284.

- [3] Arnborg, S., and Proskurowski, A. (1989), Linear time algorithms for NP-hard problems restricted to partial k-trees, *Discrete Applied Mathematics* **23**, 11–24.
- [4] Arnborg, S., Lagergren, J., and Seese, D. (1991), Easy problems for tree-decomposable graphs, *J. of Alg.* **12**, 308–340.
- [5] Baker, B.S. (1997) Approximation algorithms for NP-complete problems on planar graphs, *J ACM* **40**, 153–180.
- [6] Bistarelli, S., Montanari, U., and Rossi, F. (1997), Semiring-based constraint satisfaction and optimization, *J ACM* **44**, 201–236.
- [7] Bodlaender, H.L. (1988) Dynamic programming on graphs with bounded tree width, *Proceedings of ICALP*, LNCS 317, 105–118.
- [8] Bodlaender, H.L. (1996), A linear-time algorithm for finding tree-compositions of small treewidth, *SIAM Journal on Computing* **25**, 1305–1317
- [9] Condon, A., Feigenbaum, J., Lund, C., and Shor, P. (1995), Probabilistically checkable debate systems and approximation algorithms for PSPACE-hard functions, *Chicago Journal of Theoretical Computer Science*, **4**, A preliminary version of the paper appears in (1993) *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, 305–313.
- [10] Courcelle, B. (1990), The monadic second order logic of graphs I: recognizable sets of finite graphs, *Inf. Comput.* **85**, 12–75
- [11] Courcelle, B. (1992), The monadic second order logic of graphs III: tree-decompositions, minors, and complexity issues, *Informatique Theorique et Applications* **26**, 257–286.
- [12] Courcelle, B., Makowsky, J.A., and Rotics, U. (2000), Linear time solvable optimization problems on graphs of bounded clique width, *Theory of Computing Systems* **33(2)**, 125–150.
- [13] Condon, A., Feigenbaum, J., Lund, C., and Shor, P. (1997), Random debaters and the hardness of approximating stochastic functions, *SIAM Journal on Computing*, **26(2)**, 369–400. A preliminary version of the paper appears in (1994) *Proc. 9th IEEE Annual Conference on Structure in Complexity Theory*, 280–293.
- [14] Dyer, M.E., and A.M. Frieze, A.M. (1986), Planar 3DM is NP-complete, *J. of Alg.* **7**, 174–184.
- [15] Ebbinghaus, H.D., and Flum, J. (1999), “Finite Model Theory”, Springer-Verlag, 2nd Edition

- [16] Garey, M.R. and Johnson, D.S. (1979), “Computers and Intractability: A Guide to the Theory of NP-Completeness”, Freeman, San Francisco, CA.
- [17] Gottlob, G., and Pichler, R. (2001), Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width, (F. Orejas, P.G. Sprakis, and J. van Leeuwen Eds.), *in* “Proceedings of the 28th International Colloquium on Automata, Languages, and Programming”, volume 2076 of *Lecture Notes in Computer Science*, Springer-Verlag, 708–719.
- [18] Grohe, M. (2001) Generalized model-checking for first-order logic, (H. Reichel and A. Ferreira, Eds.), *in* “Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science”, Volume 2010 of *Lecture Notes in Computer Science*, Springer-Verlag, 12–26.
- [19] Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., and Stearns, R.E. (1998), The complexity of planar counting problems, *SIAM Journal on Computing* **27**, 1142–1167.
- [20] Jeavons, P., Cohen, D., and Gyssens, M. (1997), Closure properties of constraints, *JACM* **44**, 527–549.
- [21] Korte, B., and Lovasz, L. (1981) Mathematical structures underlying greedy algorithms, (F. Gecseg Ed.), “Fundamentals of Computation Theory”, Number 117 in *Lecture Notes in Computer Science*, Springer-Verlag, 205-209.
- [22] Lichtenstein, D. (1982), Planar formulae and their uses, *SIAM Journal on Computing* **11**, 329–343.
- [23] Lipton, R.L., and Tarjan, R.E. (1980), Applications of a planar separator theorem, *SIAM Journal on Computing* **9**, 615-629.
- [24] Makowsky, J.A. (2001) Colored Tutte polynomials and Kauffman brackets for graphs of bounded treewidth, *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 487–495.
- [25] Monien, B., and Sudborough, I.H. (1981) Bounding the bandwidth of NP-complete problems, *Lecture Notes in Computer Science* **100**, Springer-Verlag, Berlin, 279–292.
- [26] Mostowski, A. (1957), On a generalization of quantifiers, *Fundamenta Mathematicæ* **44**, 12–36.
- [27] Papadimitriou, C.H. (1985), Games against nature, *JCSS* **31**, 288–301.

- [28] Papadimitriou, C.H., and Yannakakis, M. (1991), Optimization, approximation, and complexity classes, *J. Comput. System Sci* **43**, 425–440.
- [29] Ravi, S.S., and Hunt III, H.B. (1987), Application of planar separator theorem to counting problems, *IPL* **25** 317–321.
- [30] Radhakrishnan, V., Hunt III, H.B., and Stearns, R.E (1993), Efficient Algorithms for  $\delta$ -Near Planar Graph and Algebraic Problems, (Pandos Pardalos Editor), “Complexity in Numerical Optimization”, World Scientific, 323–350.
- [31] Robertson, N. and Seymour, P.D. (1986), Graph minors II, algorithmical aspects of tree-width, *J. of Alg.*, **7**, 309–322.
- [32] Rosenthal, A. (1982), Dynamic programming is optimal for non-serial optimization problems, *SIAM Journal on Computing* **11**, 47–59.
- [33] Schaefer, T.J. (1978), The complexity of satisfiability problems, *Proceedings 10th Annual ACM Symposium on Theory of Computing*, 216–226.
- [34] Stearns, R.E., and Hunt III, H.B (1996), An Algebraic Model for Combinatorial Problems, *SIAM Journal on Computing* **25**, 448–476.
- [35] Valiant, L.G. (1979), The complexity of enumeration and reliability problems, *SIAM Journal on Computing* **8**, 410–421.
- [36] White, A.T. (1984), “Graphs, Groups and Surfaces,” McGraw Hill.