Algorithmica (1992) 7: 555–581

Algorithmica

Automatic Generation of Linear-Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families[1]

Richard B. Borie,[2] R. Gary Parker,[3] and Craig A. Tovey[3]

Abstract. This paper describes a predicate calculus in which graph problems can be expressed. Any problem possessing such an expression can be solved in linear time on any recursively constructed graph, once its decomposition tree is known. Moreover, the linear-time algorithm can be generated *automatically* from the expression, because all our theorems are proved constructively. The calculus is founded upon a short list of particularly primitive predicates, which in turn are combined by fundamental logical operations. This framework is rich enough to include the vast majority of known linear-time solvable problems.

We have obtained these results independently of similar results by Courcelle [11], [12], through utilization of the framework of Bern *et al.* [6]. We believe our formalism is more practical for programmers who would implement the automatic generation machinery, and more readily understood by many theorists.

Key Words. Linear-time algorithms, Recursive graph class, Automatic algorithm generation.

1. Introduction. Numerous research efforts have focused on the development of efficient algorithms for $\mathcal{NP}$-hard problems when instances are restricted to graphs belonging to certain recursively constructed families. Among these are trees, series-parallel graphs, Halin graphs, partial k-trees, and bandwidth-k graphs. The common feature of these families is that any sufficiently large member can be composed from smaller members of the same family, joined at special vertices called *terminals*. Fast algorithms on these graphs are typically based on dynamic programming, so that a solution to a large member can be determined directly from solutions to the smaller members which constitute it, by a recurrence relation specific to the problem. The number of terminals is restricted to a fixed k, so this recurrence relation can be evaluated efficiently, which in turn leads to an efficient algorithm (assuming a *decomposition tree* for any graph in the recursive family can be found quickly).

[1] R. B. Borie was supported by a National Science Foundation Graduate Fellowship. C. A. Tovey was supported by a Presidential Young Investigator Award from the National Science Foundation (ECS-8451032) and a matching grant from the Digital Equipment Corporation.
[2] School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA. Current address: Department of Computer Science, Box 870290, University of Alabama, Tuscaloosa, AL 35487-0290, USA.
[3] School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.

Received August 14, 1989; revised March 13, 1990. Communicated by Greg N. Frederickson.

Usually, the recurrence relation is found by intuitive methods, with complicated problems requiring much attention to details. In this paper we show how to generate automatically the recurrence from a problem description in a particular predicate calculus. To obtain a linear-time algorithm for a problem on recursively constructed graphs, we simply state the problem in our calculus, and the rest follows *automatically*.

Takamizawa *et al.* [22] appear to have initiated the activity of taking a general approach toward efficient algorithms for recursively constructed graphs, working with series-parallel graphs. Richey [18] developed other algorithms on the same family. Arnborg, Corneil, and Proskurowski [1], [2], [4] have generalized to partial k-trees, while Wimer, Hedetniemi, and Laskar [23], [24] list a large number of problems and graph families for which such problems are linear-time solvable.

Bern *et al.* [6] give a general approach for constructing linear-time algorithms for problems on recursively constructed graphs. Their method is based on a *homomorphism* that maps graph–subgraph pairs to a finite number of equivalence classes, and they call a subgraph property *regular* if such a finite homomorphism exists for the property.

More recently, Mahajan and Peters [16] show that certain properties (independence, domination, irredundance, etc.) that exhibit a type of *locality* have computable homomorphism operations, thereby leading to linear-time algorithms for problems on recursively constructed graphs. Bodlaender [7] solves problems that satisfy a composition of certain locality conditions, where instances are restricted to graphs with bounded treewidth; then Bodlaender [8] shows that such graphs include most of the popular examples of recursively constructed graphs. Scheffler and Seese [20] have independently developed still another notion of local properties, all of which lead to linear-time algorithms for recognition and optimization problems on recursively constructed graph families.

In work done independently of that reported in this paper, Courcelle [11], [12] develops a systematic method for describing graph properties using a predicate calculus, monadic second-order logic, that is similar to the calculus presented in this paper, such that the recognition problems corresponding to all expressible properties can be solved in linear time on graphs with bounded treewidth. Arnborg *et al.* [3] extend Courcelle's result to include optimization and enumeration problems. Thus these authors and ourselves reach similar conclusions, but by altogether different means. Their results are developed in the realm of formal language theory, whereas ours utilize the algorithmic framework of Bern *et al.*[6]. Our development is entirely self-contained and, we believe, much simpler. In addition, our calculus is extensible, which allows the solution of more problems including chromatic index and graph b-partitioning.

In the development of the aforementioned predicate calculus our proofs embody a method for automatically generating homomorphism classes and recurrence relations to encode the relevant information about a graph problem, thus mechanically creating a dynamic programming algorithm. Moreover, the calculus comprises a small number of easily checkable primitive predicates, combined by logical operators ($\neg$, $\wedge$, $\vee$) and quantifiers ($\forall$, $\exists$).

Many common graph properties and concepts such as adjacency, vertex degree,

connectivity, forests, cycles, matchings, minor subgraphs, and isomorphism are expressible in the calculus. This in turn allows the expressibility of most common graph problems, including vertex cover, domination, Hamiltonian cycle, Steiner subgraph, Eulerian subgraph, and chromatic number. Maximality and minimality are expressed in complete generality, so problems such as minimum–maximal matching and maximum–minimal vertex cover can be stated in the calculus as well. Not only the existence of a solution to an expressible predicate can be determined: optimum cardinality sets, optimum weighted sets, and even the number of distinct solutions can be found from valid calculus expressions.

The calculus is extensible, merely by the addition of other primitive predicates, and we give a useful extension that permits the solution to several problems, such as chromatic index, that are apparently not handled by the other work (although Bodlaender [10] has independently obtained a very similar algorithm for chromatic index using a less general method). We also give a small number of problems that are solvable in linear time on recursive families, but that do not appear to be expressible in the calculus without further extension. Further problems that are solvable in polynomial, but not linear, time on recursive graph families are discussed, as are problems that remain $\mathcal{NP}$-hard on recursive families. The latter are especially important from the perspective of the implied restrictions on further extensions to the calculus.

Before proceeding further, we wish to emphasize several important characteristics of our calculus for expressing graph problems. First, the calculus is elegantly defined in terms of a small canonical set of primitive predicates, which are combined by logical operators and quantifiers to form legal expressions.

Second, many graph problems stated in Garey and Johnson [14] can be translated into our calculus in a straightforward manner. For example, consider the vertex cover problem:

INSTANCE. Graph $G = (V, E)$, positive integer $k \leq |V|$.
QUESTION. Is there a subset $V_1 \subseteq V$ with $|V_1| \leq k$ such that, for each edge $\{u, v\} \in E$, at least one of u and v belongs to V_1?

This problem can be stated succinctly in our calculus as

$$\min|V_1|\colon (\forall e_1)(\exists v_1 \in V_1)(\mathrm{Inc}(v_1, e_1)).$$

We incorporate the use of macros to make expressions concise, so the Hamiltonian cycle problem can be stated as

$$(\exists E_1)(\mathrm{Conn}(V, E_1) \wedge \mathrm{Reg}_2(V, E_1)),$$

where Conn and Reg_m are predicates (expressed in the calculus) that test for connectedness and regularity of degree m.

Third, our calculus has considerable scope. For example, it easily solves nine of the ten problems listed by Johnson [15] for recursively constructed graphs,

including some which were listed as open or unverified. Indeed, there are only a few problems known to be solvable in linear time by dynamic programming on recursive graph families that are not known to be expressible in our calculus, so the coverage of our calculus is almost complete. In later sections of this paper we examine several such problems and conjecture that there does not exist any calculus which completely covers all such problems while still yielding an automatic algorithm generator, and hence that our results are in a sense the best possible.

2. Background. Formally, a *k-terminal graph* $G = (V, T, E)$ is a graph with an ordered set $T \subseteq V$ of distinguished vertices, known as *terminals*, such that $|T| \leq k$. Here $T = \{t_1, \ldots, t_{\tau(G)}\}$, where $\tau(G) = |T|$.

A *k-terminal graph composition operation* f is written $G = f(G_1, \ldots, G_c)$, where G and the G_j are k-terminal graphs, and c denotes the arity of the operation. f can be represented by a matrix $m(f)$ with r rows and c columns, such that $0 \leq m_{ij}(f) \leq \tau(G_j)$ for $1 \leq i \leq r$, $1 \leq j \leq c$. (Here r is the number of vertices in G that result directly from the merger of terminals under f.) The nonzero elements $m_{ij}(f)$ of the ith row indicate which terminals of each G_j are merged together, or *identified*; in particular, terminal $t_{m_{ij}(f)}$ of graph G_j is merged into the ith new vertex of G. (If an element $m_{ij}(f)$ has value 0, then no terminal from G_j is used in the creation of the ith new vertex.) The first $\tau(G)$ rows of $m(f)$ indicate the ordered set of terminals of G, and the remaining rows indicate new vertices which result by identifying terminals from the G_j but which become nonterminals of G. Therefore we let $\tau(f) = \tau(G)$, and f is completely specified as $f = (m, \tau)$.

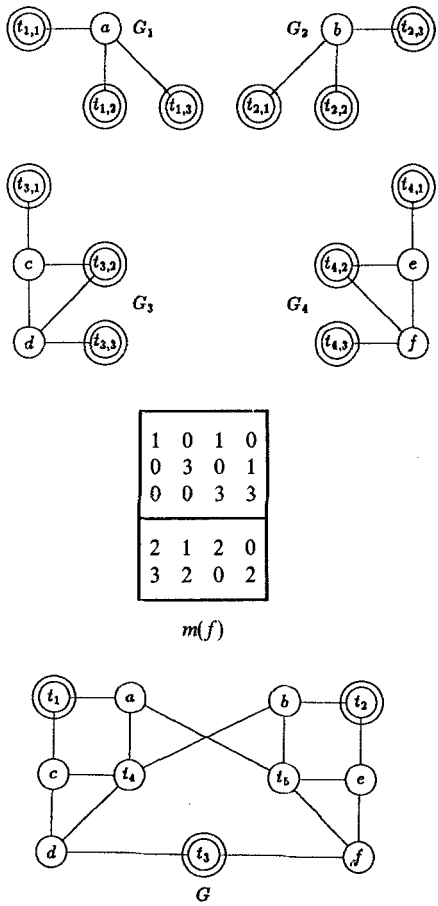
As an example, consider the 4-ary 3-terminal operation $G = f(G_1, \ldots, G_4)$, with $m(f)$ as shown and $\tau(f) = 3$, shown in Figure 1. The fourth row of $m(f)$ indicates that terminal vertices 2 of G_1, 1 of G_2, and 2 of G_3 merge to form vertex t_4 in G.

Since it is not possible to identify terminals in the same G_j, no value (other than 0) can appear more than once in a single column of $m(f)$. In addition, allowing for an arity in excess of 2 relative to f is meaningful since, in general, a c-ary operation cannot be replaced by $c - 1$ binary compositions.

We take a *base graph* to be a k-terminal graph with no nonterminal vertices. (This could be generalized to permit additional base graphs, but we have not found any cases for which this would be advantageous.) A *decomposition tree* of a k-terminal graph G is a rooted tree with vertex labels g and f such that

- $g_v = G$ if v is the root,
- f_v is a graph composition operation if v is interior,
- $g_v = f_v(g_{v_1}, \ldots, g_{v_c})$ if v is interior with children $v_1, \ldots, v_c$, and
- g_v is a base graph if v is a leaf.

Given a set of k-terminal base graphs B and a finite set of k-terminal graph composition operations R, the *k-terminal recursive family* $F = (B, R)$ is the closure of B by the operations in R. Well-known recursive families (e.g., see [23]) include trees, outerplanar graphs, series-parallel graphs, Halin graphs, bandwidth-k graphs, and partial k-trees (or, equivalently, graphs with treewidth $\leq k$). It can also be seen from Bodlaender [8] that any j-terminal recursive graph family can



1	0	1	0
0	3	0	1
0	0	3	3
2	1	2	0
3	2	0	2

$m(f)$

Fig. 1

be embedded in the class of partial k-trees for suitably large k; however, the required value of k is likely to be larger than j. Thus, because many linear-time algorithms are exponential or worse in the number of terminals, using the smaller number of terminals is clearly more desirable.

Notice that each $G \in F$ has a decomposition tree such that $g_v \in B$ if v is a leaf and each $f_v \in R$. Our aim is to determine which $\mathcal{NP}$-hard problems can be solved efficiently when the instance is restricted to a graph $G \in F$, assuming a decomposition tree of G is given as part of the instance. Note that the instance is not allowed to include any numbers, so recognition versions of problems often must either include constants in the problem description or be expressed as optimization versions instead. Also, the instance cannot contain more than one graph, so problems like subgraph isomorphism cannot be expressed in full generality.

We develop a predicate calculus in which to express problems that can be solved efficiently on a recursive family $F = (B, R)$. A problem instance includes a graph $G = (V, E) \in F$, its decomposition tree, and possibly specified subsets $V' \subseteq V$ or $E' \subseteq E$. Let variables range over obvious domains: $v_i \in V$, $e_i \in E$, $V_i \subseteq V$, and $E_i \subseteq E$.

Consider a component graph $G_j = (V_j, T_j, E_j)$ of G. The value of a vertex variable v_i with respect to G_j is either one of its terminals $t_{j1}, \ldots, t_{jk}$, an unspecified interior vertex $\text{int}_j \in V_j - T_j$, or an unspecified exterior vertex $\text{ext}_j \in V - V_j$.

The value of an edge variable e_i with respect to G_j is either an unspecified interior edge $\text{int}_j \in E_j$ or an unspecified exterior edge $\text{ext}_j \in E - E_j$. The value of a vertex set variable V_i with respect to G_j is one of the at most 2^k subsets of T_j, because it suffices to know merely which terminals are included in V_i. The value of an edge set variable E_i with respect to G_j can be treated as a constant, say $\emptyset$, because its value is not explicitly used when applying a composition operation.

Let $P_G(x_1, \ldots, x_t)$ denote a t-ary predicate, where the x_i are variables and G can be omitted if understood. Also let $F_{x_1,\ldots,x_t}$ denote $\{(G, x_1, \ldots, x_t)\colon G \in F$ and the x_i range over the appropriate domains$\}$. Note that each $f \in R$ has a corresponding operation $\circ_f$ that guarantees merely that the free variables of P are selected compatibly in the realm of $F_{x_1,\ldots,x_t}$ when f is applied in the realm of F.

As an example, consider the well-known class of biconnected series-parallel graphs, which has binary operations p (parallel) and s (series) defined by the following matrices:

1	1
2	2

$m(p)$

1	0
0	2
2	1

$m(s)$

If $G = f(G_1, G_2)$ where f is either p or s, then

$$\circ_f((G_1, x_{11}, \ldots, x_{1t}), (G_2, x_{21}, \ldots, x_{2t})) = (G, x_1, \ldots, x_t),$$

where each x_j has the same domain as x_{1j} and x_{2j} and is determined from their values according to the rules illustrated in the following tables (empty entries indicate incompatible operands):

- If x_j is a vertex variable:

$\circ_p$	t_{21}	t_{22}	int_2	ext_2
t_{11}	t_1			
t_{12}		t_2		
int_1				int
ext_1			int	ext

$\circ_s$	t_{21}	t_{22}	int_2	ext_2
t_{11}				t_1
t_{12}	int			
int_1				int
ext_1		t_2	int	ext

- If x_j is an edge variable:

$\circ_p$	int_2	ext_2
int_1		int
ext_1	int	ext

$\circ_s$	int_2	ext_2
int_1		int
ext_1	int	ext

- If x_j is a vertex set variable:

$\circ_p$	$\emptyset$	$\{t_{21}\}$	$\{t_{22}\}$	T_2
$\emptyset$	$\emptyset$			
$\{t_{11}\}$		$\{t_1\}$		
$\{t_{12}\}$			$\{t_2\}$	
T_1				T

$\circ_s$	$\emptyset$	$\{t_{21}\}$	$\{t_{22}\}$	T_2
$\emptyset$	$\emptyset$		$\{t_2\}$	
$\{t_{11}\}$	$\{t_1\}$		T	
$\{t_{12}\}$		$\emptyset$		$\{t_2\}$
T_1		$\{t_1\}$		T

- If x_j is an edge set variable:

$\circ_p$	$\emptyset$
$\emptyset$	$\emptyset$

$\circ_s$	$\emptyset$
$\emptyset$	$\emptyset$

More generally, if $G = f(G_1, \ldots, G_m)$ is any graph composition operation, then $\circ_f((G_1, x_{11}, \ldots, x_{1t}), \ldots, (G_m, x_{m1}, \ldots, x_{mt})) = (G, x_1, \ldots, x_t)$, where each x_j is determined as follows:

- If x_j is a vertex variable:
  If one or more of the x_{ij} denote all the terminals that are merged together to form some new vertex t_l of G, then x_j is either t_l (if t_l is a terminal of G) or int (if t_l loses its terminal status).
  If one of the x_{ij} is int_i and each other $x_{i'j}$ is $\text{ext}_{i'}$ (for $i' \neq i$), then x_j is int.
  If each x_{ij} is ext_i, then x_j is ext.
  Any other combination is incompatible.
- If x_j is an edge variable:
  If one of the x_{ij} is int_i and each other $x_{i'j}$ is $\text{ext}_{i'}$ (for $i' \neq i$), then x_j is int.
  If each x_{ij} is ext_i, then x_j is ext.
  Any other combination is incompatible.
- If x_j is a vertex set variable:
  If for each set of terminals that are merged together to form some new vertex t_l of G, either all of these terminals are members of their respective x_{ij} or all of them are not, then x_j contains exactly those terminals that meet the former condition.
  Any other combination is incompatible.
- If x_j is an edge set variable:
  Every combination is always compatible.

In the spirit of Bern *et al.* [6], we say that a predicate $P(x_1, \ldots, x_t)$ is *regular* if there is a finite set C and a *homomorphism* $h: F_{x_1,\ldots,x_t} \to C$ such that, for each $\circ_f$, there exists a multiplicative operation $\odot_f$ of C such that both

- properties are preserved:

$$h(G_1, x_{11}, \ldots, x_{1t}) = h(G_2, x_{21}, \ldots, x_{2t}) \to P_{G_1}(x_{11}, \ldots, x_{1t}) = P_{G_2}(x_{21}, \ldots, x_{2t})$$

and

- integrity of composition operators is maintained:

$$h(\circ_f((G_1, x_{11}, \ldots, x_{1t}), \ldots, (G_m, x_{m1}, \ldots, x_{mt}))) = \odot_f(h(G_1, x_{11}, \ldots, x_{1t}), \ldots, h(G_m, x_{m1}, \ldots, x_{mt})).$$

(Observe that the homomorphism class depends upon both the graph and the values of the free variables.) A class $c \in C$ is said to be *accepting* if $h(G, x_1, \ldots, x_t) = c$ implies that $P_G(x_1, \ldots, x_t)$ is true.

If a decomposition tree for $G \in F$ is given and P is regular, then there is a linear-time algorithm to find optimal values $x_1, \ldots, x_t$ satisfying P, using a straightforward dynamic programming approach. This follows because C is described by a finite set of homomorphism classes, which thus leads to a constant amount of work at each node of the decomposition tree. (The size of this tree is, of course, linear in the size of G.)

3. Regular Predicates. In this section we establish the regularity property with respect to a set of primitive predicates. (These primitives are later used to generate other predicates, or macros, that are also regular.) We begin by showing that the primitive predicates vertex equality, vertex-edge incidence, membership, and a congruence property for cardinality are regular.

THEOREM 1. *Each of the following predicates is regular*:

1. $v_1 = v_2$ (*vertex equality*).
2. $\mathrm{Inc}(v_1, e_1)$ (*vertex-edge incidence*).
3. $v_1 \in V_1$ (*vertex membership*).
4. $e_1 \in E_1$ (*edge membership*).
5. $|E_1| \equiv a \pmod b$ (*congruence property for edge cardinality*).
6. $|V_1| \equiv a \pmod b$ (*congruence property for vertex cardinality*).

PROOF. The first four cases are similar, having $C = \{0, 1\}$ with 1 the accepting class (such that the predicate is true); for $G = f(G_1, \ldots, G_m)$, let $\odot_f$ be m-ary logical disjunction, $\vee_m$.

1. $v_1 = v_2$. If G is a base graph, let $h(G, v_1, v_2)$ be 1 if both v_1 and v_2 are vertices of G such that $v_1 = v_2$, and 0 otherwise.

2. $\mathrm{Inc}(v_1, e_1)$. If G is a base graph, let $h(G, v_1, e_1)$ be 1 if both v_1 and e_1 are in G and v_1 is incident to e_1, and 0 otherwise.

3. $v_1 \in V_1$. For G a base graph with vertices V, let $h(G, v_1, V_1)$ be 1 if $v_1 \in V_1 \cap V$ and 0 otherwise.

4. $e_1 \in E_1$. For G a base graph with edge set E, let $h(G, e_1, E_1)$ be 1 if $e_1 \in E_1 \cap E$ and 0 otherwise.

5. $|E_1| \equiv a \pmod b$. Let $C = \{0, \ldots, b-1\}$, with accepting class 0. If G is a base graph with edge set E, let $h(G, e_1, E_1)$ be $|E_1 \cap E| \pmod b$. If $G = f(G_1, \ldots, G_m)$, let $\odot_f$ be the sum (mod b) of the $h(G_j)$.

6. $|V_1| \equiv a \pmod{b}$. This case is essentially the same as the previous one, except that $\odot_f$ compensates for multiple counts of v_i when v_i is a terminal that appears in more than one G_j. (This predicate is not needed in what follows, and is in fact later rendered redundant by an extension to the calculus based on a stronger primitive, but it is included here for completeness.) □

The next theorem shows that the primitive predicates can be employed to construct others that also satisfy the regularity property. Specifically, any predicate constructed from the primitive predicates using logical negation, conjunction, disjunction, universal and existential quantification is regular.

THEOREM 2. *The set of regular predicates is closed under* $\neg$, $\wedge$, $\vee$, $\forall$, *and* $\exists$, *where quantification is over variables which range over vertices, edges, vertex sets, and edge sets.*

PROOF. To simplify matters, transform any expression Q into prenex normal form, such that all the quantifiers are brought to the outermost nesting level. That is, let $Q(x_1, \ldots, x_t) = (q_1 y_1) \cdots (q_s y_s) P(y_1, \ldots, y_s, x_1, \ldots, x_t)$, where each q_i is either $\forall$ or $\exists$. Let the primitive properties that are embedded in P (those that contain no $\neg$, $\wedge$, or $\vee$) be $P_1, \ldots, P_r$. Since each P_i is regular, it has a corresponding homomorphism h_i that maps into a finite set S_i of homomorphism classes with cardinality n_i. Let the cross product $S = \prod_{i=1}^{r} S_i$ (with cardinality $n = \prod_{i=1}^{r} n_i$) be the set of homomorphism classes corresponding to P. Whether a class of S is accepting for particular values of the y_i can be determined by application of the operators $\neg$, $\wedge$, and $\vee$, assuming it is known whether the classes of each S_i are accepting for these y_i values. Simply replace accepting classes with *true* (1) and nonaccepting classes with *false* (0), then apply the logical operators. (That these logical operators preserve regularity is shown in more detail in [6].) Hence, for fixed y_i, the expression P is regular.

Let $Q_i = (q_i y_i) \cdots (q_s y_s) P(y_1, \ldots, y_s, x_1, \ldots, x_t)$. Since it has just been shown that $Q_{s+1} (= P)$ is regular, it suffices to show, for $s \geq i \geq 1$, that if Q_{i+1} is regular, then Q_i is regular. Let C denote the (finite) set of homomorphism classes of Q_{i+1}. There are four cases, because y_i either denotes a vertex, an edge, a vertex set, or an edge set.

- y_i is a vertex.
  For $1 \leq j \leq k$ let C_j be the set of homomorphism classes of Q_{i+1} that are possible when y_i is the jth terminal. Let I be the set of possible sets of homomorphism classes of Q_{i+1}, one set corresponding to the substitution of each interior vertex for y_i. (Although there could conceivably be very many such interior vertices, these substitutions produce only a constant number of different subsets of classes from C.) Let E be the set of homomorphism classes of Q_{i+1} when y_i is an exterior vertex, i.e., one which is not in this component of the decomposed graph.
  The set of homomorphism classes of Q_i is the cross product $C' = C_1 \times \cdots \times C_k \times I \times E$, where $|C'| \leq (|C|)^{k+1} 2^{|C|}$, so Q_i is regular. Notice that $c = (c_1, \ldots, c_k, d, e) \in C'$ is accepting iff either
  $q_i = \exists$ and $\{c_1, \ldots, c_k\} \cup d$ contains an accepting class, or
  $q_i = \forall$ and $\{c_1, \ldots, c_k\} \cup d$ contains only accepting classes.

We have yet to show how to multiply the newly constructed homomorphism classes, and to verify that the resultant mapping is a homomorphism. If $G = f(G_1, \ldots, G_m)$, then let $\odot_f((c_{11}, \ldots, c_{1k}, d_1, e_1), \ldots, (c_{m1}, \ldots, c_{mk}, d_m, e_m)) = (c_1, \ldots, c_k, d, e)$. Here each c_j is the intersection of all the c_{pq} such that t_{pq} merges into t_j together with all the e_p such that no terminal of G_p merges into t_j; d is the set that has as its members both

the intersections of all the c_{pq} such that t_{pq} merges into v together with all the e_p such that no terminal of G_p merges into v, for each vertex v that loses its terminal status when f is applied, and

the intersections of each set that is a member of each d_p together with all the $e_{p'}$ such that $p' \neq p$;

and e is the intersection of all the e_p.

- y_i is an edge.

  Let I be the set of possible sets of homomorphism classes of Q_{i+1}, one set for y_i being each interior edge. Let E be the set of homomorphism classes of Q_{i+1} when y_i is an exterior edge. Then the set of homomorphism classes of Q_i is the cross product $C' = I \times E$, where $|C'| \leq 2^{|C|}|C|$, so Q_i is regular. Here $c = (d, e) \in C'$ is accepting iff either

  $q_i = \exists$ and d contains an accepting class, or

  $q_i = \forall$ and d contains only accepting classes.

  If $G = f(G_1, \ldots, G_m)$, then let $\odot_f((d_1, e_1), \ldots, (d_m, e_m)) = (d, e)$, where d is the set that contains the intersections of each set that is a member of each d_p together with all the $e_{p'}$ such that $p' \neq p$, and e is the intesection of all the e_p.

- y_i is a vertex set.

  The set of homomorphism classes of Q_i is $C' = \prod_{A \subseteq T} C_A$, where C_A denotes the set of homomorphism classes of Q_{i+1} such that $A = y_i \cap T$. Thus $|C'| \leq (2^{|C|})^{2^k}$, and Q_i is regular. Letting c_j denote the set of homomorphism classes such that $y_i \cap T$ has a bit-vector representation (over universal set T) that corresponds to the binary representation of integer j, it follows that $c = (c_0, \ldots, c_{2^k-1}) \in C'$ is accepting iff either

  $q_i = \exists$ and some c_j contains an accepting class, or

  $q_i = \forall$ and each c_j contains only accepting classes.

  If $G = f(G_1, \ldots, G_m)$, then let $\odot_f((c_{10}, \ldots, c_{1,2^k-1}), \ldots, (c_{m0}, \ldots, c_{m,2^k-1})) = (c_0, \ldots, c_{2^k-1})$. Here each c_j is the union of all intersections of sequences of values $c_{1p_1}, \ldots, c_{mp_m}$ that are *compatible* with respect to c_j; by this we mean that there is some vertex set W such that $y_i \cap T \subseteq W \subseteq V - (T - y_i)$ and each $W \cap T_q$ is a set whose bit-vector representation (over universal set T_q) corresponds to the binary representation of p_q. (It suffices to consider only those W which consist of terminals from the component graphs $G_1, \ldots, G_m$, of which there are a bounded number, so the computation of $\odot_f$ takes constant time.)

- y_i is an edge set.

  The set of homomorphism classes of Q_i is $C' = \{C_B : B \subseteq E\}$, where C_B denotes the homomorphism class of Q_{i+1} such that $B = y_i \cap E$. So $|C'| \leq 2^{|C|}$ and Q_i is regular. Hence $c \in C'$ is accepting iff either

  $q_i = \exists$ and c contains an accepting class, or

  $q_i = \forall$ and c contains only accepting classes.

If $G = f(G_1, \ldots, G_m)$, then let $\odot_f(c_1, \ldots, c_m) = c$, where c is the intersection of all the c_j.

That the resultant mapping is indeed a homomorphism follows directly from definitions and from the meanings of $\exists$ and $\forall$. This completes the inductive step. □

The proofs of Theorems 1 and 2 indicate a method for automatically generating homomorphism classes and recurrence relations to encode the relevant details about a graph problem expressed in terms of the primitive predicates with logical operators and quantifiers. (It does not appear likely that a simple formula exists to map an arbitrary k-terminal graph to its image under the homomorphism; rather this image must be obtained by applying the sequence of transformations implied by Theorem 2. Furthermore, while it is true that closure under the operations $\vee$ and $\exists$ would follow as a corollary of closure under $\neg$, $\wedge$, and $\forall$, we believe the proof of Theorem 2 to be more concise as written.) Although the number of classes generated by application of the indicated method could conceivably be enormous, in practice a reduction technique can be employed to minimize the number of such classes, using notions of class equivalence to remove classes that are empty, redundant, or unreachable (see [6]). The theorems merely guarantee that the number of classes is constant, with respect to the size of a problem instance.

4. An Automatically Generated Algorithm. We illustrate the aforementioned automatic algorithm generator with an example involving vertex covering. The recognition version of this problem can be expressed in the calculus as

$$(\exists V_1)(\forall e_1)(\exists v_1 \in V_1)(\mathrm{Inc}(v_1, e_1)).$$

We solve this problem on the family of series-parallel graphs, by explicitly generating the classes implied by the above proofs. The only base graph of the series-parallel graphs is

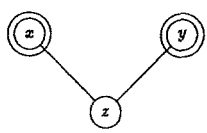
which belongs to the homomorphism class

$$\begin{aligned}&(\{(\{(01, 01, \emptyset, 00)\}, (00, 00, \emptyset, 00))\},\\ &\{(\{(11, 01, \emptyset, 00)\}, (10, 00, \emptyset, 00))\},\\ &\{(\{(01, 11, \emptyset, 00)\}, (00, 10, \emptyset, 00))\},\\ &\{(\{(11, 11, \emptyset, 00)\}, (10, 10, \emptyset, 00))\}),\end{aligned}$$

which can be seen as follows. The outermost 4-tuple contains sets corresponding to the possible values of $V_1 \cap \{x, y\}$: $\emptyset$, $\{x\}$, $\{y\}$, and $\{x, y\}$. (Recall that V_1 is the outermost quantified variable in the expression for vertex cover.) Each of these

sets has only one element, because there is no interior vertex, and hence no alternative ways to select nonterminals in V_1. This single element is a pair, whose first entry is a set corresponding to interior edges (of which there is only one), and whose second entry corresponds to an exterior edge. At the innermost layer are 4-tuples, this time corresponding to possible values of v_1: x, y, interior, and exterior. Each element contains a value of the vector $[v_1 \in V_1, \text{Inc}(v_1, e_1)]$. For instance, the first such value corresponds to $V_1 \cap \{x, y\} = \emptyset$, e_1 is interior, and $v_1 = x$; here $v_1 \in V_1$ is false and $\text{Inc}(v_1, e_1)$ is true, so the vector is 01.

Next a series operation between two base graphs with vertex sets $\{x, z\}$ and $\{z, y\}$ respectively produces the graph

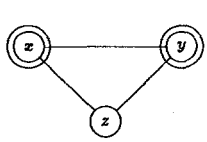


which belongs to the homomorphism class

({({(01, 00, {01}, 00), (00, 01, {01}, 00)}, (00, 00, {00}, 00)),
({(01, 00, {11}, 00), (00, 01, {11}, 00)}, (00, 00, {10}, 00))},
{({(11, 00, {01}, 00), (10, 01, {01}, 00)}, (10, 00, {00}, 00)),
({(11, 00, {11}, 00), (10, 01, {11}, 00)}, (10, 00, {10}, 00))}
{({(01, 10, {01}, 00), (00, 11, {01}, 00)}, (00, 10, {00}, 00)),
({(01, 10, {11}, 00), (00, 11, {11}, 00)}, (00, 10, {10}, 00))},
{({(11, 10, {01}, 00), (10, 11, {01}, 00)}, (10, 10, {00}, 00)),
({(11, 10, {11}, 00), (10, 11, {11}, 00)}, (10, 10, {10}, 00))}).

Here the fourth value of the outermost 4-tuple consists of the last two rows and corresponds to $V_1 \cap \{x, y\} = \{x, y\}$. In fact, these two rows correspond to $V_1 = \{x, y\}$ and $V_1 = \{x, y, z\}$, respectively. Each row consists of a pair: a set of values corresponding to the possible interior edges e_1, and a value corresponding to an exterior edge e_1. As before, each innermost 4-tuple consists of vectors corresponding to each of $v_1 = x$, $v_1 = y$, v_1 is interior, and v_1 is exterior. Therefore, for example, the third element of the final such 4-tuple on the last row corresponds to $V_1 = \{x, y, z\}$, e_1 is exterior, and v_1 is interior. The only possibility is that $v_1 = z$, so this element is a set with a single member; observe that $v_1 \in V_1$ is true and $\text{Inc}(v_1, e_1)$ is false, hence the vector 10.

Following, a parallel operation yields a graph G



which belongs to the homomorphism class

({{(01, 01, {00}, 00), (01, 00, {01}, 00), (00, 01, {01}, 00)}, (00, 00, {00}, 00)),
 {(01, 01, {10}, 00), (01, 00, {11}, 00), (00, 01, {11}, 00)}, (00, 00, {10}, 00))},
{{(11, 01, {00}, 00), (11, 00, {01}, 00), (10, 01, {01}, 00)}, (10, 00, {00}, 00)),
 {(11, 01, {10}, 00), (11, 00, {11}, 00), (10, 01, {11}, 00)}, (10, 00, {10}, 00))},
{{(01, 11, {00}, 00), (01, 10, {01}, 00), (00, 11, {01}, 00)}, (00, 10, {00}, 00)),
 {(01, 11, {10}, 00), (01, 10, {11}, 00), (00, 11, {11}, 00)}, (00, 10, {10}, 00))},
{{{(11, 11, {00}, 00), (11, 10, {01}, 00), (10, 11, {01}, 00)}, (10, 10, {00}, 00)),
 {(11, 11, {10}, 00), (11, 10, {11}, 00), (10, 11, {11}, 00)}, (10, 10, {10}, 00))}).

The structure of this class resembles that of the class which resulted from the series operation above, except that each set corresponding to interior edges e_1 now has cardinality 3, which is simply because the graph G now under consideration has three edges.

Suppose we are interested in knowing the solution to the vertex cover problem on the graph G. To determine whether the homomorphism class is an accepting class, we first remove the cases involving exterior vertices and edges. This yields

({{(01, 01, {00}), (01, 00, {01}), (00, 01, {01})},
 {(01, 01, {10}), (01, 00, {11}), (00, 01, {11})}},
{{(11, 01, {00}), (11, 00, {01}), (10, 01, {01})},
 {(11, 01, {10}), (11, 00, {11}), (10, 01, {11})}},
{{(01, 11, {00}), (01, 10, {01}), (00, 11, {01})},
 {(01, 11, {10}), (01, 10, {11}), (00, 11, {11})}},
{{(11, 11, {00}), (11, 10, {01}), (10, 11, {01})},
 {(11, 11, {10}), (11, 10, {11}), (10, 11, {11})}}).

Next each accepting class is replaced by *true* (1) and each nonaccepting class is replaced by *false* (0); for the given problem this step has no effect. Then the logical operators $\neg$, $\wedge$, and $\vee$ are applied to obtain the results of the expression $v_1 \in V_1 \wedge \text{Inc}(v_1, e_1)$,

({{(0, 0, {0})},
 {(0, 0, {0}), (0, 0, {1})}},
{{(1, 0, {0}), (0, 0, {0})},
 {(1, 0, {0}), (1, 0, {1}), (0, 0, {1})}},
{{(0, 1, {0}), (0, 0, {0})},
 {(0, 1, {0}), (0, 0, {1}), (0, 1, {1})}},
{{(1, 1, {0}), (1, 0, {0}), (0, 1, {0})},
 {(1, 1, {0}), (1, 0, {1}), (0, 1, {1})}}).

The innermost quantifier, $(\exists v_1)$, is now applied to obtain

({{0}, {0, 1}},
 {{0, 1}, {1}},
 {{0, 1}, {1}},
 {{1}}).

Finally $(\forall e_1)$ is applied to obtain $(\{0\}, \{0, 1\}, \{0, 1\}, \{1\})$, and $(\exists V_1)$ is applied to obtain (0, 1, 1, 1), which means that a vertex cover C exists for each $V_1 \cap C \subseteq T$ other than $\emptyset$.

The linearity of the algorithm is a direct result of the observations that there are a finite number of composition rules, each of which can be described by a finite matrix, and that the number of nodes in a decomposition tree is linear in the number of edges in the original graph. As explained in the following, the cardinality of a minimum vertex cover can be computed, or an optimum cover can be produced by maintaining back pointers in the usual fashion.

5. Development of the Calculus

5.1. Some Macros. In order to demonstrate how the primitives in Theorem 1 can be combined to form more complex expressions, we can create some macros that will prove useful in making expressions more concise. Following, we capture this with a pair of easy theorems. The first is more general.

THEOREM 3. *If P and Q are regular properties, then each of the following predicates is a regular property*:

$$
\begin{aligned}
P \to Q &\Leftrightarrow \neg P \vee Q.\\
P \leftrightarrow Q &\Leftrightarrow (P \to Q) \wedge (Q \to P).\\
(\exists x \in X)(P(x)) &\Leftrightarrow (\exists x)(x \in X \wedge P(x)).\\
(\forall x \in X)(P(x)) &\Leftrightarrow (\forall x)(x \in X \to P(x)).\\
Q(P(x)) &\Leftrightarrow (\exists y)((y = P(x)) \wedge Q(y)).\\
P(x_i)(1 \le i \le m) &\Leftrightarrow P(x_1) \wedge \cdots \wedge P(x_m).\\
P(x_i, x_j)(1 \le i < j \le m) &\Leftrightarrow (P(x_i, x_j)(1 \le i \le j - 1))(2 \le j \le m).
\end{aligned}
$$

Relative to well-known graph-theoretic properties, we have

THEOREM 4. *Each of the following predicates is a regular property*:

1. $e_1 = e_2 \Leftrightarrow (\forall v_1)(\mathrm{Inc}(v_1, e_1) \leftrightarrow \mathrm{Inc}(v_1, e_2))$.
2. $\mathrm{Adj}(v_1, v_2, E_1) \Leftrightarrow (\exists e_1 \in E_1)(\mathrm{Inc}(v_1, e_1) \wedge \mathrm{Inc}(v_2, e_1)) \wedge \neg v_1 = v_2$.
3. $V_1 \cup V_2 = V_3 \Leftrightarrow (\forall v_1)((v_1 \in V_1 \vee v_1 \in V_2) \leftrightarrow v_1 \in V_3)$.
4. $V_1 \backslash V_2 = V_3 \Leftrightarrow (\forall v_1)((v_1 \in V_1 \wedge \neg v_1 \in V_2) \leftrightarrow v_1 \in V_3)$.
5. $\mathrm{Part}(V_0, V_1, \ldots, V_m) \Leftrightarrow (V_1 \cup \cdots \cup V_m = V_0) \wedge (V_i \cap V_j = \emptyset)(1 \le i < j \le m)$.
6. $V_1 \subseteq V_2 \Leftrightarrow (\forall v_1)(v_1 \in V_1 \to v_1 \in V_2)$.
7. $V_1 = V_2 \Leftrightarrow V_1 \subseteq V_2 \wedge V_2 \subseteq V_1$.
8. $V_1 \subset V_2 \Leftrightarrow V_1 \subseteq V_2 \wedge \neg V_2 \subseteq V_1$.
9. $|V_1| \ge m \Leftrightarrow (\exists v_1 \in V_1, \ldots, v_m \in V_1)(\neg v_i = v_j(1 \le i < j \le m))$.
10. $\min(V_1, P(V_1)) \Leftrightarrow P(V_1) \wedge (\forall V_2)(V_2 \subset V_1 \to \neg P(V_2))$.
11. $\max(V_1, P(V_1), V_2) \Leftrightarrow P(V_1) \wedge (\forall V_3)((V_1 \subset V_3 \wedge V_3 \subseteq V_2) \to \neg P(V_3))$.
12. $E_1 = \mathrm{Ind}(V_1) \Leftrightarrow (\forall e_1)(e_1 \in E_1 \leftrightarrow (\forall v_1)(\mathrm{Inc}(v_1, e_1) \to v_1 \in V_1))$.
13. $E_1 = \mathrm{IncE}(V_1) \Leftrightarrow (\forall e_1)(e_1 \in E_1 \leftrightarrow (\exists v_1 \in V_1)(\mathrm{Inc}(v_1, e_1)))$.

14. $V_1 = \text{IncV}(E_1) \Leftrightarrow (\forall v_1)(v_1 \in V_1 \leftrightarrow (\exists e_1 \in E_1)(\text{Inc}(v_1, e_1)))$.
15. $\deg(v_1, E_1) = m \Leftrightarrow |E_1 \cap \text{IncE}(\{v_1\})| = m$.
16. $\text{Reg}_m(V_1, E_1) \Leftrightarrow (\forall v_1 \in V_1)(\deg(v_1, E_1) = m)$.
17. $\text{Isom}_H(V_1, E_1) \Leftrightarrow (\exists v_1 \in V_1, \ldots, v_{|H|} \in V_1)$
$\times (\neg v_i = v_j \wedge (\text{Adj}(v_i, v_j, E_1) \leftrightarrow \text{Adj}(i, j, H))$
$\times (1 \le i < j \le |H|)) \wedge |V_1| = |H|$.
18. $\text{Conn}(V_1, E_1) \Leftrightarrow (\forall V_2, V_3)(\text{Part}(V_1, V_2, V_3) \rightarrow (V_2 = \emptyset \vee V_3 = \emptyset$
$\vee (\exists v_2 \in V_2, v_3 \in V_3)(\text{Adj}(v_2, v_3, E_1))))$.
19. $\text{Conn}_m(V_1, E_1) \Leftrightarrow \text{Isom}_{K_m}(V_1, E_1) \vee (|V_1| \ge m + 1$
$\wedge (\forall V_2)(|V_2| \le m - 1$
$\rightarrow \text{Conn}(V_1 \backslash V_2, E_1 \backslash \text{IncE}(V_2))))$.
20. $\text{Forest}(V_1, E_1) \Leftrightarrow (\forall V_2)(\text{Conn}_2(V_2, \text{Ind}(V_2) \cap E_1) \rightarrow \neg(V_2 \subseteq V_1))$.
21. $\text{Tree}(V_1, E_1) \Leftrightarrow \text{Forest}(V_1, E_1) \wedge \text{Conn}(V_1, E_1)$.
22. $\text{Path}(V_1, E_1) \Leftrightarrow \text{Tree}(V_1, E_1) \wedge (\forall v_1 \in V_1)(\deg(v_1, E_1) \le 2)$.
23. $\text{Cycle}(V_1, E_1) \Leftrightarrow \text{Conn}(V_1, E_1) \wedge \text{Reg}_2(V_1, E_1)$.
24. $\text{Match}(E_1) \Leftrightarrow \text{Reg}_1(\text{IncV}(E_1), E_1)$.
25. $\text{Minor}_H(V_0, E_0) \Leftrightarrow (\exists V_1, \ldots, V_{|H|})(\text{Part}(V_0, V_1, \ldots, V_{|H|})$
$\wedge \text{Conn}(V_i, \text{Ind}(V_i))(1 \le i \le |H|)$
$\wedge (\text{Adj}(i, j, H) \rightarrow (\exists v_1 \in V_i, v_2 \in V_j)$
$\times (\text{Adj}(v_1, v_2, E_0)))$
$\times (1 \le i < j \le |H|))$.

Observe that the predicates in Theorem 4 form only a partial list. For ease of presentation we have omitted numerous obvious operations (e.g., $\cap$, $\le$, and the edge set analogues of macros 3–9).

5.2. Applications. We now demonstrate how the predicate calculus developed in the previous section provides the basis for linear-time solvability on recursively constructed graph families.

THEOREM 5. *If $P(x_1, \ldots, x_t)$ is regular (for fixed t), then each of the following problems can be solved in linear time on recursively constructed graph families.*

- *Recognition.*
  $(\exists x_1) \cdots (\exists x_t) P(x_1, \ldots, x_t)$.
- *Optimization.*
  If $t = 1$ and x_1 denotes a set, then $\min |x_1|: P(x_1)$ *or* $\max |x_1|: P(x_1)$.
- *Enumeration.*
  Compute the number of solutions, $|\{(x_1, \ldots, x_t): P(x_1, \ldots, x_t)\}|$.

PROOF. First observe that a decomposition tree D for a graph G in a recursive family F has leaves that are mutually edge-disjoint base graphs, so the size of D is linear in the size of G. Let r denote the root node of D.

• Recognition. By Theorem 2, if $P(x_1, \ldots, x_t)$ is regular, then $Q = (\exists x_1) \cdots (\exists x_t) P(x_1, \ldots, x_t)$ is also regular. Thus there are a finite set C and homomorphism $h: F \rightarrow C$ corresponding to Q. The set of base graphs is finite, so h can be evaluated

at each leaf of D in constant time. The set of graph composition operations is also finite, and for each one there is a corresponding finite multiplication table, so h can be evaluated at each interior node of D in constant time. Therefore h is found for r by a linear-time dynamic programming algorithm, and Q is true for G iff $h(r)$ is an accepting class of C.

- Optimization. Since $P(x_1)$ is regular, it has a corresponding finite set C and homomorphism h: $F_{x_1} \to C$. For each class $c \in C$ and node v of D, let $z_c(v)$ denote the optimum of $|x_1|$ over all values of x_1 for which $h(g(v), x_1) = c$. Computation of each z_c is trivial at leaves of D; at interior nodes, it involves summation—modified so that duplicate elements are counted only once—and either minimization or maximization over the possible sums. The solution to the optimization problem is found by taking the minimum or maximum over those $z_c(r)$ such that c is an accepting class.
- Enumeration, $P(x_1, \ldots, x_t)$ is regular, and the corresponding finite set and homomorphism are C and h: $F_{x_1,\ldots,x_t} \to C$. For each class $c \in C$ and node v of D, let $z_c(v)$ denote $|\{(x_1, \ldots, x_t)\colon P(x_1, \ldots, x_t) \wedge h(g(v), x_1, \ldots, x_t) = c\}|$. Computation of each z_c is trivial at leaves of D; at interior nodes, it involves computing the sum of products of terms whose corresponding variable assignments are compatible. The solution to the enumeration problem is found by taking the sum of all $z_c(r)$ such that c is an accepting class. □

To illustrate, recall the vertex cover problem described earlier:

$$\mathrm{VC}(V_1) = (\forall e_1)(\exists v_1 \in V_1)(\mathrm{Inc}(v_1, e_1)).$$

Theorem 5 shows that linear-time algorithms can be found for each of

- $(\exists V_1)(\mathrm{VC}(V_1))$,
- $\min|V_1|\colon \mathrm{VC}(V_1)$, and
- $\mathrm{Compute}|\{V_1\colon \mathrm{VC}(V_1)\}|$.

In the optimization case, a minimum cardinality solution can be reconstructed by maintaining back pointers, in the standard manner.

The next theorem summarizes the status of numerous popular problems.

THEOREM 6. *The recognition, optimization, and enumeration versions of each of the following problems is solvable in linear time on any recursively constructed graph family: vertex cover, independent set, dominating set, clique, maximal matching, Hamiltonian cycle, cubic subgraph, planar subgraph, Eulerian subgraph, Steiner tree, rural postman, leafset of spanning tree, k-degree-bounded spanning tree, k-partite subgraph, k-colorability, edge k-colorability, partition into k trees, partition into k forests, partition into k perfect matchings, partition into k cliques, and covering by k cliques.*

PROOF. Each of these problems can be stated by a well-formed calculus expression. To see this, consider the following examples:

Dominating set:

$$\mathrm{DS}(V_1) \Leftrightarrow V = \mathrm{IncV}(\mathrm{IncE}(V_1)).$$

Planar subgraph:

$$\mathrm{Planar}(E_1) \Leftrightarrow \neg \mathrm{Minor}_{K_5}(V, E_1) \wedge \neg \mathrm{Minor}_{K_{3,3}}(V, E_1).$$

Rural postman problem (a subset of edges E' is part of the instance):

$$\mathrm{RP}(E_1) \Leftrightarrow E' \subseteq E_1 \wedge \mathrm{Cycle}(\mathrm{IncV}(E_1), E_1).$$

k-Colorability:

$$\mathrm{Color}(V_1, \ldots, V_k) \Leftrightarrow \mathrm{Part}(V, V_1, \ldots, V_k) \wedge \mathrm{IS}(V_i)(1 \leq i \leq k). \qquad \square$$

Of course, many well known $\mathcal{NP}$-hard problems are described on graphs with weighted vertices and/or edges. For example, vertex cover, independent set, and dominating set are no less difficult in the general case when vertices are weighted. Steiner tree and rural postman normally have weighted edges, while graph b-partitioning often has both weighted vertices and edges. Our next result demonstrates that cardinality cases can be replaced by their weighted analogues.

THEOREM 7. *The cardinality of a vertex or edge set, $|x_i|$, can be replaced by the sum of the weights of its elements, $\sum_{y \in x_i} w(y)$, in either a regular predicate or an optimization function, while maintaining a linear-time algorithm.*

PROOF. It is obvious that the proof of the optimization portion of Theorem 5 can be modified to handle sums of vertex or edge weights, by altering the values of $z_c(v)$ appropriately for each leaf v.

The case in which a subexpression $|x_i| \geq m$ is replaced by $\sum_{y \in x_i} w(y) \geq m$ is more difficult. Let $x'_j = \{z: w(z) = j\}$ (for $j < m$) and $x'_m = \{z: w(z) \geq m\}$ be specified with the instance. The subexpression can be replaced by

$$\bigvee_S \bigwedge_{1 \leq j \leq m} (\exists y_1 \in x_i \cap x'_j, \ldots, y_{a_j} \in x_i \cap x'_j)(\neg y_a = y_b)(1 \leq a < b \leq a_j),$$

where $S = \{(a_1, \ldots, a_m): m \leq \sum_{1 \leq j \leq m}(ja_j) < m + \min\{j: a_j > 0\}\}$ is a finite set corresponding to possible partitions of the weight $w(x_i)$ among members of the various $x_i \cap x'_j$. $\square$

Finally, we demonstrate how easily we can find linear-time algorithms for new problems. Consider the following (contrived) problem:

INSTANCE. Graph $G = (V, E)$, weights $w(v)$ for each $v \in V$, integer m.
QUESTION. Is there a subset $V_1 \subseteq V$ with weight $w(V_1) \geq m$ such that V_1 is minimal with respect to the properties, that it is a dominating set and the graph formed by removing the vertices of V_1 and their incident edges is planar?

Given an arbitrary recursively constructed graph, our interest is in deciding whether this problem is solvable in linear time. We know from the preceding results that this question is answered affirmatively if the problem can be stated in the predicate calculus. Accordingly, we build the following representative expression:

$$\max \sum_{v_1 \in V_1} w(v_1)\colon \min(V_1, \mathrm{DS}(V_1) \wedge \mathrm{Planar}(\mathrm{Ind}(V \backslash V_1))).$$

This legal expression guarantees a linear-time dynamic programming algorithm for the given problem; its generation is assumed by the proof of Theorem 2.

6. Extensions to the Calculus. The calculus can be extended with additional variable domains and primitive regular predicates, thereby allowing us to express additional problems and generate their linear-time solutions on recursively constructed graph classes. Apparently, no other general framework for linear-time computation on recursive graph classes shares this feature, so extensibility permits the expression of problems not handled by other formalisms.

Before we demonstrate how to extend the calculus, note that we can state several graph problems in terms of a vertex set partition such that each piece of the partition must induce a certain type of graph. For some of these problems these induced graphs must be connected (consider partition into trees, cliques, or fixed connected graphs H); for other problems these graphs can be disconnected (forests, independent sets, or perfect matchings) but any collection of the graph's connected components is the same type (e.g., each connected component of a forest is a tree, and any collection of disjoint trees is a forest). This sort of problem is linear-time solvable because, given the type of graph that must be induced by each connected component and whether or not each piece of the partition is required to be connected, it is possible to determine efficiently which vertices are in the various pieces (as we show below). Problems that involve partitioning the vertex set to satisfy other types of conditions, such as where the number of edges induced by each piece is a constant c but these pieces can induce disconnected graphs, do not appear to be solvable in linear time by these techniques.

To express problems involving partitions in the calculus, we introduce two new domains, $\mathscr{A}$ and $\mathscr{B}$. $\mathscr{A}$ consists of those partitions of the vertices such that each piece of the partition induces a connected subgraph, while $\mathscr{B}$ allows disconnected induced subgraphs subject to the constraint that any refinement of a partition from $\mathscr{B}$ that preserves connected components of pieces must satisfy the same conditions as that partition. (Observe that $\mathscr{A}$ is a subdomain of $\mathscr{B}$, because no partition from $\mathscr{A}$ has any such refinement other than itself.) Any variable A_i or B_i is assumed in the following to have domain $\mathscr{A}$ or $\mathscr{B}$ respectively. Theorem 8 establishes two new primitive regular predicates.

THEOREM 8. *Each of the following predicates is regular*:

1. $V_1 \in A_1$ (*membership of vertex set in partition of connected pieces*).
2. $A_1 \propto B_1$ (*partition refinement that preserves connected components*).

PROOF. This proof is in the spirit of the proof of Theorem 1.

1. $V_1 \in A_1$, which states that vertex set V_1 is one of the pieces of partition A_1. Consider the set of homomorphism classes $C = \{\bot, \top\} \cup \{\text{partitions of subsets of } \{1, \ldots, k\}\}$, where the accepting classes are $\top$ and those partitions that are singletons (e.g., $\{\{1, 2, 3\}\}$ but not $\{\{1, 2\}, \{3\}\}$). The intent is that each partition represents one of the different ways in which the terminals that are in set V_1 are connected by paths through only those interior vertices that are in the same piece of the partition A_1, so a singleton partition means that all the terminals in V_1 are indeed in a single piece of A_1. Also, $\bot$ is a class that can never lead to an accepting class, and $\top$ is the class such that V_1 is found to lie entirely in the interior of the graph.

If $G = (V, T, E)$ is a base graph, let $h(G, V_1, A_1)$ be as follows: if $V_1 \cap T = \emptyset$, then $h(G, V_1, A_1) = \emptyset$; if $V_1 \cap T = \{t_1\}$, then $h(G, V_1, A_1) = \{\{1\}\}$; if $V_1 \cap T = \{t_2\}$, then $h(G, V_1, A_1) = \{\{2\}\}$; and otherwise (i.e., $V_1 \cap T = \{t_1, t_2\}$), if t_1 and t_2 are in the same piece of partition A_1, then $h(G, V_1, A_1) = \{\{1, 2\}\}$, else $h(G, V_1, A_1) = \bot$. If instead $G = f(G_1, \ldots, G_m)$, then $\odot_f$ is determined by the manner in which terminals merge together. We illustrate the multiplication tables for the binary 2-terminal operations p (parallel) and s (series):

$\odot_p$	$\top$	$\emptyset$	$\{\{1\}\}$	$\{\{2\}\}$	$\{\{1\}, \{2\}\}$	$\{\{1, 2\}\}$	$\bot$
$\top$	$\bot$	$\top$					$\bot$
$\emptyset$	$\top$	$\emptyset$					$\bot$
$\{\{1\}\}$			$\{\{1\}\}$				$\bot$
$\{\{2\}\}$				$\{\{2\}\}$			$\bot$
$\{\{1\}, \{2\}\}$					$\{\{1\}, \{2\}\}$	$\{\{1, 2\}\}$	$\bot$
$\{\{1, 2\}\}$					$\{\{1, 2\}\}$	$\{\{1, 2\}\}$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

$\odot_s$	$\top$	$\emptyset$	$\{\{1\}\}$	$\{\{2\}\}$	$\{\{1\}, \{2\}\}$	$\{\{1, 2\}\}$	$\bot$
$\top$	$\bot$	$\top$		$\bot$			$\bot$
$\emptyset$	$\top$	$\emptyset$		$\{\{2\}\}$			$\bot$
$\{\{1\}\}$	$\bot$	$\{\{1\}\}$		$\{\{1\}, \{2\}\}$			$\bot$
$\{\{2\}\}$			$\top$		$\bot$	$\{\{2\}\}$	$\bot$
$\{\{1\}, \{2\}\}$			$\bot$		$\bot$	$\{\{1\}, \{2\}\}$	$\bot$
$\{\{1, 2\}\}$			$\{\{1\}\}$		$\{\{1\}, \{2\}\}$	$\{\{1, 2\}\}$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

2. $A_1 \propto B_1$, which states that partition A_1 is a refinement of partition B_1 such that the pieces of A_1 are precisely the connected components of the pieces of B_1. Consider $C = \{0, 1\}$, with 1 the accepting class. If G is a base graph with vertices V, let $h(G, V_1, A_1) = 1$ if $\{V_1 \cap V: V_1 \in A_1\} = \{V_1 \cap V: V_1 \in B_1\}$ (so that an edge is induced by a piece of A_1 iff it is also induced by a piece of B_1); otherwise $h(G, V_1, A_1) = 0$. If $G = f(G_1, \ldots, G_m)$, let $\odot_f$ be $\wedge_m$ (m-ary logical conjunction).

The intent is that the primitive $A_1 \propto B_1$ is false iff some edge is induced by a piece of either A_1 or B_1 but not both. □

There does not appear to be any way to generalize either part of Theorem 8 to include arbitrary partitions of the vertices. The reason is that, when $f(G_1, \ldots, G_m)$ is applied, $\odot_f$ must receive enough information from the different $h(G_j)$ to determine which pieces of the partition from the different G_j are combined into the same piece for G. While this is obvious whenever a piece has nonempty intersection with the terminals of G_j (because each terminal is in exactly one piece), it is more difficult when this intersection is empty. However, an empty intersection with the terminals of G_j can be handled, if the partition belongs to either of the domains $\mathscr{A}$ or $\mathscr{B}$. The pieces of a partition from domain $\mathscr{A}$ must induce connected subgraphs, so no piece that is entirely interior to G_j can be combined with any piece that is entirely exterior to G_j. While the pieces of a partition from domain $\mathscr{B}$ can induce disconnected subgraphs, each piece that is entirely interior to G_j can be combined *arbitrarily* with any piece that is entirely exterior, because every refinement of a partition from $\mathscr{B}$ that preserves connected components satisfies the same conditions as that partition.

THEOREM 9. *The set of regular predicates is closed under quantification over variables from domains $\mathscr{A}$ and $\mathscr{B}$.*

PROOF. The proof of Theorem 2 is extended to handle a quantified variable y_i with domain $\mathscr{A}$ or $\mathscr{B}$. Let $\pi(T)$ denote the set of partitions of the terminal set T; note that $|\pi(T)| \leq k!$. For each $j_1 \in \pi(T)$ and each refinement j_2 of j_1, let $C_{(j_1, j_2)}$ denote the set of homomorphism classes of Q_{i+1} such that $\{V_1 \cap T: V_1 \in y_i\} = j_1$ and two terminals are connected by a path through interior vertices of a piece of partition j_1 iff they are in the same piece of j_2. Then the set C' of homomorphism classes of Q_i is contained in the cross product $\prod_{(j_1, j_2): j_1 \in \pi(T) j_2 \in \pi(T), j_2 \propto j_1} C_{(j_1, j_2)}$, so $|C'| \leq (2^{|C|})^{(k!)^2}$ and Q_i is regular.

Here $c = (\ldots, c_{(j_1, j_2)}, \ldots) \in C'$ is accepting iff either

- $q_i = \exists$ and some $c_{(j_1, j_2)}$ contains an accepting class, or
- $q_i = \forall$ and every $c_{(j_1, j_2)}$ contains only accepting classes.

To multiply homomorphism classes, suppose $G = f(G_1, \ldots, G_m)$. Define $\odot_f((\ldots, c_{(j_{11}, j_{12})}, \ldots), \ldots, (\ldots, c_{(j_{m1}, j_{m2})}, \ldots)) = (\ldots, c_{(j_1, j_2)}, \ldots)$. Then each $c_{(j_1, j_2)}$ is the union of all intersections of sequences of values $c_{(j_{11}, j_{12})}, \ldots, c_{(j_{m1}, j_{m2})}$ that are *compatible* with respect to $c_{(j_1, j_2)}$; by this we mean that there exist graphs $G_{p_1}, \ldots, G_{p_q}$ and vertices $v_{p_0}, \ldots, v_{p_q}$ such that the terminals of each G_{p_r} that merge into $v_{p_{r-1}}$ and v_{p_r} (for $r = 1, \ldots, q$) are in the same piece of the $j_{p_r s}$th partition of T_{p_r} iff v_{p_0} and v_{p_q} are in the same piece of the j_sth partition of T, for $s = 1, 2$ (also for domain $\mathscr{A}$ only, there is an additional requirement for compatibility that no two vertices in the same piece of some j_{i1} but in different pieces of j_{i2} can be stranded with no means of ever joining the same connected component). □

We illustrate Theorem 9 with multiplication tables for compatibility of the

parallel and series operations. For conciseness, denote $\{\{1, 2\}\}$ by S (same components) and $\{\{1\}, \{2\}\}$ by D (different components). First for the domain $\mathscr{A}$:

$\circ_p$	(S, S)	(S, D)	(D, D)
(S, S)	(S, S)	(S, S)	
(S, D)	(S, S)	(S, D)	
(D, D)			(D, D)

$\circ_s$	(S, S)	(S, D)	(D, D)
(S, S)	(S, S)	(S, D)	(D, D)
(S, D)	(S, D)		
(D, D)	(D, D)		(S, D), (D, D)

Next for the domain $\mathscr{B}$:

$\circ_p$	(S, S)	(S, D)	(D, D)
(S, S)	(S, S)	(S, S)	
(S, D)	(S, S)	(S, D)	
(D, D)			(D, D)

$\circ_s$	(S, S)	(S, D)	(D, D)
(S, S)	(S, S)	(S, D)	(D, D)
(S, D)	(S, D)	(S, D)	(D, D)
(D, D)	(D, D)	(D, D)	(S, D), (D, D)

THEOREM 10. *Theorem 5 can be extended to include recognition, optimization, and enumeration problems when some free variables of a predicate* $P(x_1, \ldots, x_t)$ *have domains* $\mathscr{A}$ *or* $\mathscr{B}$.

PROOF.

- Recognition. $(\exists x_1)\cdots(\exists x_t)P(x_1, \ldots, x_t)$ is regular, so this part is true by a straightforward generalization.
- Optimization. If x_1 denotes a partition from A_1 or B_1, then $\min|x_1|$: $P(x_1)$ or $\max|x_1|$: $P(x_1)$ can be solved as follows. For either a min or max problem on A_1 or a max problem on B_1, interior nodes of the decomposition tree count, using addition operations, the number of connected components that are absorbed entirely into the interior of a graph, and also keep track of the number of connected components that contain terminals. For a min problem on B_1 the situation is different because each piece of the partition could consist of several connected components; therefore interior nodes of the decomposition tree replace the addition operations for counting the number of connected components by maximum operations, to allow arbitrary connected components from separate portions of the graph to combine into the same piece of the partition.
- Enumeration. Computing the number of solutions, $|\{(x_1, \ldots, x_t)$: $P(x_1, \ldots, x_t)\}|$, still involves computing sums of products of terms whose corresponding variable values are compatible. Of course, the computations for $\mathscr{A}$ and $\mathscr{B}$ produce different results, because partitions from $\mathscr{B}$ can allow several connected components in each piece, whereas partitions from $\mathscr{A}$ can only have one. □

Theorem 11 defines a macro that tests whether a vertex set V_1 induces one of the connected components of some piece of a partition B_1. This macro is then used to express several of the problems in Theorem 12.

THEOREM 11. *The following predicate is regular*:

$$V_1 \hat{\in} B_1 \Leftrightarrow (\exists A_1)(V_1 \in A_1 \wedge A_1 \propto B_1).$$

THEOREM 12. *The recognition, optimization, and enumeration versions of each of the following problems is solvable in linear time on any recursively constructed graph family*:

Parition into trees:

$$\text{PartT}(A_1) \Leftrightarrow (\forall V_1 \in A_1)(\text{Tree}(V_1, \text{Ind}(V_1))).$$

Partition into cliques:

$$\text{PartC}(A_1) \Leftrightarrow (\forall V_1 \in A_1)(\text{Clique}(V_1)).$$

Generalized H-matching for a connected graph H:

$$\text{GMatch}_H(A_1) \Leftrightarrow (\forall V_1 \in A_1)(\text{Isom}_H(V_1, \text{Ind}(V_1))).$$

Partition into forests:

$$\text{PartF}(B_1) \Leftrightarrow (\forall V_1)(V_1 \hat{\in} B_1 \rightarrow \text{Forest}(V_1, \text{Ind}(V_1))).$$

Chromatic number:

$$\text{Color}(B_1) \Leftrightarrow (\forall V_1)(V_1 \hat{\in} B_1 \rightarrow \text{IS}(V_1, \text{Ind}(V_1))).$$

Partition into perfect matchings:

$$\text{PartPM}(B_1) \Leftrightarrow (\forall V_1)(V_1 \hat{\in} B_1 \rightarrow \text{Reg}_1(V_1, \text{Ind}(V_1))).$$

Graph b-partitioning:

$$\text{GPart}_b(E_1) \Leftrightarrow (\exists A_1)((\forall V_1 \in A_1)(|V_1| \leq b \wedge \text{Conn}(V_1, \text{Ind}(V_1))) \wedge (\forall e_1)(e_1 \in E_1 \leftrightarrow \neg(\exists V_1 \in A_1)(e_1 \in \text{Ind}(V_1)))).$$

Theorem 12 shows that the optimization version of partition into cliques is solvable in linear time on each of series-parallel graphs, partial k-trees, and bandwidth-k graphs. We show in the next section that the optimization version of edge chromatic number (chromatic index) is solvable in polynomial time on partial k-trees and bandwidth-k graphs, which will complete the demonstration of polynomiality for nine out of the ten graph problems listed by Johnson [15] for recursively constructed families such as series-parallel graphs, Halin graphs, partial k-trees, and bandwidth-k trees. (The remaining problem, graph isomorphism, takes an instance containing two graphs, and hence does not fit our model.)

7. Limits to the Approach. Even though the calculus is extensible, it is not simple to discover new primitives that increase the number of significant expressible problems, subject to the constraint that each expressible problem must be regular. There exist a few problems solvable in linear time on recursively constructed

families that are not thought to be expressible in the extended calculus of the previous section. There also exist some closely related problems that are not thought to be solvable in linear time on these families, some of which are known to be solvable in polynomial time by dynamic programming and some of which remain $\mathcal{NP}$-hard.

One additional problem that could be included by a trivial modification of the extension in the previous section is m-Multichromatic number, coloring each vertex with m colors so that no adjacent vertices have any color in common. Merely define domains $\mathscr{A}_m$ and $\mathscr{B}_m$ of sets of vertex sets such that no vertex is in more that m sets, but that otherwise satisfy the conditions of $\mathscr{A}$ and $\mathscr{B}$ respectively (which are now simply $\mathscr{A}_1$ and $\mathscr{B}_1$).

Covering by cliques is another problem that is solvable in linear time on recursive families, and for which the connectedness condition holds. A dynamic programming algorithm merely needs a homomorphism class for each possible combination of

- which subsets of the k terminals are contained in any single clique,
- which of these cliques also contains an interior vertex (and hence cannot contain any exterior vertices), and
- which edges between terminals already exist.

There are less than $2^{2^k} \times 2 \times 2^{k^2}$ combinations. Observe that there is no bound on the number of cliques in which any particular vertex can appear, so this problem is not included by the extension to the $\mathscr{A}_m$ domains. Rather, it is the fact that no interior vertex can be in the same clique as any exterior vertex that allows a bounded number of classes; hence the only graph covering problems for which regularity is maintained in this way are covering by cliques and special cases, such as covering by triangles.

Many similar covering problems such as covering by matchings (edge chromatic number), by perfect matchings, by a fixed connected H, by cycles, by Hamiltonian cycles, by f-factors, by f-regular subgraphs, and even by *hypocliques* (cliques with one missing edge) can be solved in polynomial time, but do not appear to be solvable in linear time. As an example, consider edge chromatic number. A homomorphism class must maintain information about which edge colors are already incident to each terminal, and with t colors this implies 2^{kt} classes. By realizing that the minimum information a class needs to contain is not *which* colors, but rather *how many* colors are incident to exactly each subset of the terminals, this number can be reduced only to $\Theta(t^{2^k})$. Vizing's theorem states that the minimum cardinality edge coloring is equal either to the maximum vertex degree Δ or to $\Delta + 1$, and there is no degree bound even for simple recursive families like trees. Hence the best dynamic programming algorithm for edge chromatic number appears to take $\Omega(n^{2^k})$ time.

Covering by forests and by trees (which are actually two equivalent problems, when the instances are connected graphs) are solvable in linear time, however, and are included in the extension to the $\mathscr{A}_m$ and $\mathscr{B}_m$ domains through the following observation: each recursively constructed graph family uses a finite cardinality

rule set and therefore has some associated bound m on its connectivity, so at most m forests or trees are sufficient.

Another related problem that is solvable in linear time is f-factorization (note this is distinct from covering by f-factors, in which a given edge could occur in more than one f-factor). Any f-factorable graph must be cf-regular, where c is the number of factors. Each graph in a k-terminal recursive family has a vertex with degree not more than k (consider a vertex that is a nonterminal at any node which has only leaves as children in the decomposition tree). Hence the number of factors $c \leq k \div f$.

A similar argument to those above can be used to show that chromatic number can be solved in linear time. By an easy induction on the size of a decomposition tree, for each recursive family there exists a constant r such that any k-terminal graph in the family can be r-colored with each terminal assigned a distinct color. Thus at most r vertex sets suffice, but chromatic number fits into the calculus extended with $\mathscr{A}$ and $\mathscr{B}$, so this argument is unnecessary (but still interesting).

One more problem that appears to be polynomial but not linear is the optimization version of the degree-bounded spanning tree. For any fixed d, it suffices to have one homomorphism class for each possible combination of

- a partition of the k terminals into already-connected components and
- assignments of degrees $0 \leq d_i \leq d$ for $1 \leq i \leq k$.

This is $\pi(T) \times (d+1)^k$ classes, but in the optimization version it might be that $d = \Theta(n)$, so the best dynamic programming algorithm appears to take $\Omega(n^k)$ time.

Some of the problems that do have linear-time algorithms are quite similar to problems that remain $\mathscr{NP}$-hard on recursive graph families. Generalized H-matching appears to be nonlinear if H is not connected, and it appears to remain $\mathscr{NP}$-hard if H is connected and given as part of the instance. To see why no dynamic programming algorithm seems to be applicable, even on trees, consider a graph H with a central vertex v and paths of length $1, 2, \ldots, m$ emanating from it. The number of homomorphism classes needed just for the cases when vertex v coincides with a terminal vertex during application of an algorithm is at least $\Theta(2^m) = \Theta(2^{\sqrt{n}})$, where each class corresponds to a distinct subset of the m paths and indicates which paths begin with an interior edge.

A closely related problem, subgraph isomorphism, is solvable in linear time on recursive families and in $O(n^{|H|})$ time on all graphs for a fixed subgraph H. This problem is $\mathscr{NP}$-hard if H is given as part of the instance [21], even for the special case of subtree isomorphism on series-parallel graphs [17].

The graph partitioning problem with b given as part of the instance cannot be handled efficiently by dynamic programming, because the subexpression $|V_1| \leq b$ translates into $\neg(\exists v_1, \ldots, v_{b+1})$, which is not a fixed size expression. Notice that the expression given for graph b-partitioning uses the unboundedness of $|A_1|$ to ensure the permissibility of the assumption that V_1 induces a connected compo- graphs, because $|V| \leq b|A_1|$. Furthermore, if edges are ignored and the problem is to minimize $|A_1|$, then the problem becomes equivalent to bin packing, with a piece of the partition corresponding to a bin. If edges are ignored, $|A_1|$ is fixed,

and b is either part of the instance or is to be minimized, then the problem becomes equivalent to processor scheduling, with each piece of the partition corresponding to the jobs executed by a certain processor.

Certain problems are known to be difficult on recursively constructed graphs. For example, bandwidth is shown to be $\mathcal{NP}$-hard on trees in [13], while optimal linear arrangement and minimum cut linear arrangement are polynomial on trees but still open on series-parallel graphs. Interestingly, bandwidth can be expressed in a calculus with just a slight bit more freedom than our extended calculus, merely by generalizing to allow a domain of arbitrary sets of vertex sets:

t-Bandwidth

$$\begin{aligned}\mathrm{BW}_t(A_1) \quad \Leftrightarrow \quad & V \in A_1 \wedge (\forall V_1 \in A_1)((V_1 = \emptyset \vee (\exists v_1 \in V_1)(V_1 \backslash \{v_1\} \in A_1)) \\ & \qquad \wedge (\forall v_1 \in V_1,\, v_2 \in V_1) \\ & \qquad \times ((V_1 \backslash \{v_1\} \in A_1 \wedge V_1 \backslash \{v_2\} \in A_1) \rightarrow v_1 = v_2)) \\ & \wedge (\forall v_1, v_2)(\mathrm{Adj}(v_1, v_2, E) \\ & \qquad \rightarrow \neg(\exists V_1 \in A_1, \ldots, V_{t+1} \in A_1) \\ & \qquad \times (v_1 \in V_1 \wedge \neg v_2 \in V_{t+1} \\ & \qquad\quad \wedge V_i \subset V_{i+1} (1 \leq i \leq t))).\end{aligned}$$

(Informally, this expression maintains a set $S_v = \{u: l(u) \leq l(v)\}$ in place of the label $l(v)$ of each vertex v, and $A_1 = \{S_v: v \in V\} \cup \{\emptyset\}$.)

Finally, problems that are PSPACE-hard are not expressible in the calculus because expressions are restricted to a finite number of quantifiers. As an example consider the open problem Generalized Kayles [19], a game in which two players alternate moves, with a move being the removal of any vertex and all its neighbors from a graph, until the graph becomes empty. Given a recursive graph, does the first player have a winning strategy to defeat his adversary, i.e., one which guarantees he can remove the final vertex? Observe that the first player's moves are existential, while his opponent's moves are universal. It appears that for a dynamic programming strategy to prove useful, there would have to be a homomorphism class for each possible sequence of the form $(q_1 v_1) \cdots (q_m v_m)$, where each q_i is either $\forall$ or $\exists$, and $m = O(n)$ is the maximum size of an independent set of vertices. This is because there is no locality to the game—players can remove any vertex remaining on each move—so the sequence of $\forall$'s and $\exists$'s that occur within a particular subgraph will not necessarily alternate. Therefore it appears that the calculus cannot come near to expressing PSPACE-hard problems.

8. Conclusions. In this paper we have developed a calculus for expressing graph problems that are regular, and which can thus be solved in linear time using dynamic programming when instances are confined to recursive graphs families. Indeed, we have provided an automatic algorithm generation mechanism. Our demonstration of the strength of these results has included the resolution of various problems of apparently unknown or unverified complexity status. We have also

shown how to extend the calculus, including examples that illustrate the difficulties inherent with doing so.

Unfortunately, our attempts to discover a precise characterization of the set of *all* regular problems have been unsuccessful. We conjecture that there are theoretical reasons for this outcome.

CONJECTURE 13. *There exists no* finite *set of primitives such that every regular predicate can be expressed in terms of those primitives and the logical connectives* $\neg$, $\wedge$, $\vee$, $\forall$, *and* $\exists$.

Finally, note that there also exist some graph problems that are linear-time solvable by methods that bear no resemblance to dynamic programming. For instance, consider the problem of determining whether the number of edges in a graph is the square of an integer, which is linear-time solvable for *arbitrary* graphs. There might not exist any finite extension to the calculus that would allow the expression of such a problem.

CONJECTURE 14. *For any problem that can be solved in linear time when the instance is restricted to be a recursively constructed graph, but that cannot be solved in linear time on arbitrary graphs, there is a finite set of primitives such that this problem is expressible in terms of these primitives and the connectives* $\neg$, $\wedge$, $\vee$, $\forall$, *and* $\exists$.

Acknowledgments. The authors wish to acknowledge the many helpful suggestions of the anonymous referees.

References

[1] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability, *BIT*, **25** (1985), 2–23.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM Journal of Algebraic and Discrete Methods*, **8** (1987), 277–284.

[3] A. Arnborg, J. Lagergren, and D. Seese, Problems easy for tree-decomposable graphs, (extended abstract) *Proceedings of the 15th ICALP*, Lecture Notes in Computer Science, Vol. 317, Springer-Verlag, Berlin (1988), pp. 38–51; to appear in *Journal of Algorithms*.

[4] A. Arnborg and A. Proskurowski, Linear time algorithms for $\mathcal{NP}$-hard problems on graphs embedded in k-trees, TRITA-NA-8404, Royal Institute of Technology, Sweden (1984).

[5] M. Bauderon and B. Courcelle, Graph expressions and graph rewritings, *Mathematical Systems Theory*, **20** (1987), 83–127.

[6] M. W. Bern, E. L. Lawler, and A. L. Wong, Linear time computation of optimal subgraphs of decomposable graphs, *Jornal of Algorithms*, **8** (1987), 216–235.

[7] H. L. Bodlaender, Dynamic programming on graphs with bounded tree-width, Technical report, Laboratory for Computer Science, M.I.T. (1987); extended abstract in *Proceedings of ICALP* (1988).

[8] H. L. Bodlaender, Planar graphs with bounded tree-width, RUU-CS-88-14, University of Utrecht (1988).

[9] H. L. Bodlaender, $\mathcal{NC}$-algorithms for graphs with small treewidth, *Proceedings of the Workshop on Graph-Theoretic Concepts in Computer Science* (J. van Leeuwen, ed.) (1988), Lecture Notes in Computer Science, Vol. 344, Springer-Verlag, Berlin (1988), pp. 1–10.

[10] H. L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory* (1988), pp. 223–232.
[11] B. Courcelle, Recognizability and second-order definability for sets of finitegraphs, Report I-8634, Universite de Bordeaux (1987).
[12] B. Courcelle, An algebraic and logical theory of graphs, a survey, unpublished manuscript (1988).
[13] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth, Complexity results for bandwidth minimization, *SIAM Journal of Applied Mathematics*, **34** (1978), 477–495.
[14] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, San Francisco (1979).
[15] D. S. Johnson, The $\mathcal{NP}$-completeness column: an ongoing guide, *Journal of Algorithms*, **6** (1985), 434–451.
[16] S. Mahajan and J. G. Peters, Algorithms for regular properties in recursive graphs, *Proceedings of the 25th Annual Allerton Conference on Communications, Control, and Computing* (1987), pp. 14–23.
[17] R. L. Rardin and R. G. Parker, Tree subgraph isomorphism is $\mathcal{NP}$-complete on series-parallel graphs, unpublished manuscript (1985).
[18] M. B. Richey, Combinatorial optimization on series-parallel graphs: algorithms and complexity, Ph.D. Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology (1985).
[19] T. Schaefer, Complexity of some two-person perfect information games, *Journal of Computer and System Sciences*, **16** (1978), 185–225.
[20] P. Scheffler and D. Seese, A combinatorial and logical approach to linear-time computability, extended abstract (1986).
[21] M. M. Syslo, The subgraph isomorphism problem for outerplanar graphs, *Theoretical Computer Science*, **17** (1982), 91–97.
[22] K. Takamizawa, T. Nishizeki, and N. Saito, Linear-time computability of combinatorial problems on series-parallel graphs, *Journal of the Association for Computing Machinery*, **29** (1982), 623–641.
[23] T. V. Wimer, Linear algorithms on k-terminal graphs, Ph.D. Thesis, Report No. URI-030, Clemson University (1987).
[24] T. V. Wimer, S. T. Hedetniemi, and R. Laskar, A methodology for constructing linear graph algorithms, *Congressus Numerantium*, **50** (1985), 43–60.