

ON THE PARALLEL COMPLEXITY OF ACYCLIC LOGIC PROGRAMS

SHIVA CHAUDHURI, YANNIS DIMOPOULOS and CHRISTOS D. ZAROLIAGIS

*Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany**

Received (received date)

Revised (revised date)

Communicated by (Name of Editor)

ABSTRACT

In this paper we investigate the parallel complexity of computing the stable model of acyclic general logic programs. Within this class of logic programs, we consider the cases of negative and definite logic programs. Both cases are proved to be \mathcal{P} -complete. We prove the same for a related problem, namely that of computing the kernel of a directed acyclic graph.

Keywords: Theory of parallel computation, logic programming, stable model, kernel of a graph.

1. Introduction

The semantics of a *general* logic program (roughly the problem of what a general logic program should entail) has been an active research area over the last years. A *propositional (general) logic program* is a set of rules of the form $A \leftarrow L_1, \dots, L_n$, where A is an atom (called the *head* of the rule) and L_1, \dots, L_n are literals (called the *body* of the rule). A literal is a formula of the form B (called *positive literal*) or $\neg B$ (called *negative literal*), where B is an atom. A logic program is called *definite* or *positive*, if every literal in the body of the rules is positive. On the other hand, if every literal in the body of the rules is negative, then the logic program is called *negative*.

Due to the differences in the way negation is interpreted, several different semantics for general logic programs exist [14] (e.g. stable models, partial stable models and the well-founded model semantics). While these semantics in general draw different inferences from a general logic program, there are two cases where these inferences coincide. The first is the case of definite logic programs, where the semantics are defined by the minimal model of the definite program. The second is the case of *acyclic* general logic programs. A general logic program P is acyclic if a certain directed graph G_P associated with it (Section 2) does not contain cycles.

*Email: {shiva, yannis, zaro}@mpi-sb.mpg.de.

In this paper we investigate the parallel complexity of computing the *stable model* of an *acyclic* propositional general logic program. Our restriction to acyclic programs is a necessary one. The problem of deciding whether a general logic program has a stable model is \mathcal{NP} -complete, even in the case of negative logic programs [5]. On the other hand, although for every definite logic program there is a polynomial time algorithm that computes its minimal model [7], the problem has been proved to be \mathcal{P} -complete in [12]. Furthermore, though severely restricted, acyclic general logic programs are of particular practical importance (see e.g. [1]).

In this paper, we show the following: (i) Computing the stable model of a negative acyclic logic program is \mathcal{P} -complete (Section 3). (ii) Computing the stable model (or equivalently the minimal model) of a definite acyclic logic program is also \mathcal{P} -complete (Section 4). The proof of the first result is based on the \mathcal{P} -completeness of finding a *kernel* in a directed acyclic graph (Section 3) and on an interesting property relating the kernel of G_P and the stable model of P (Section 2). Note that the former problem is of independent graph-theoretic interest. (It is known that computing the kernel of a general directed graph is \mathcal{NP} -complete [8] and moreover, there is a trivial, linear time, sequential algorithm for finding the kernel of a directed acyclic graph.) The proof of the second result is based on an interesting connection between the circuit value problem and definite logic programs (Section 4).

2. Preliminaries

An *interpretation* of a logic program P is an assignment of values True or False to the atoms of P . An interpretation of a definite logic program P is a *model* of P , iff for every rule $A \leftarrow L_1, \dots, L_n$ in P for which every L_1, \dots, L_n is assigned the value True, A is also assigned the value True. A model M is a *minimal model* of a definite logic program P , if there is no other model of P that assigns True to a subset of the atoms that M does. A model (and in general an interpretation) is represented by a set of atoms M which contains only the atoms that are assigned the value True. Hence, if an atom is not in M then this atom is False in M . We follow this convention throughout the paper. We also assume that the reader is familiar with basic graph-theoretic terminology.

Definition 1 [10] *Let P be a logic program and M be an interpretation of P . Define $G_M(P)$ to be the logic program obtained by: (i) deleting every rule with a negative literal $\neg p$ in its body such that $p \in M$; and (ii) deleting all negative literals from the remaining rules.*

Note that $G_M(P)$ is a definite logic program and has a unique minimal model [6].

Definition 2 [10] *Let P be a logic program and let M be an interpretation of P . Then, M is called the *stable model* of P if M is the minimal model of $G_M(P)$.*

The following proposition is a direct consequence of Definition 2.

Proposition 3 *Let M be a stable model of a logic program P and let \mathcal{M} be the set $\{p : p \in \text{head}(r), r \text{ is a rule of } P, \text{ such that for every positive literal } m \in \text{body}(r),$*

$m \in M$ holds, while for every negative literal $\neg q \in \text{body}(r)$, $q \notin M$ holds}. Then, $\mathcal{M} = M$.

A logic program may have none, one or several stable models.

Definition 4 (Rule Graph) [4,5] *Let P be a logic program. The rule graph of P is a directed graph $G_P = (V, E)$ constructed as follows. The set of vertices $V = R \cup L \cup \{S\}$, where R contains a vertex for every rule in P (called rule vertex), L contains a vertex for every literal that occurs in the head of a rule in P (called literal vertex), and S is a distinguished vertex. The set of edges E is constructed as follows:*

- *If $v_i \in R$ is the rule vertex associated with the rule r_i , then:*
 - (a) *For every positive literal $l_m \in \text{body}(r_i)$ and its associated literal vertex $v_m \in L$, create the edge (v_m, v_i) . If l_m does not occur in the head of a rule in P , then create the edge (S, v_i) .*
 - (b) *If l_h is the atom in $\text{head}(r_i)$, then create the edge (v_i, v_h) , where v_h is the literal vertex associated with l_h .*
- *If $v_i, v_j \in R$ are the vertices associated with the rules r_i, r_j respectively and moreover, $\text{head}(r_i) = h_i$ and $\neg h_i \in \text{body}(r_j)$, then create the edge (v_i, v_j) .*

We call a logic program *acyclic* if its rule graph does not contain cycles. Acyclic logic programs have always a unique stable model [2]. We next give the definition of a graph-theoretic concept which has been proved useful in logic programming [4,5].

Definition 5 *Let $G = (V, E)$ be a directed graph. A set of vertices $K \subseteq V$ is a kernel for G , if K is an independent set (i.e. for any two vertices $u, w \in K$, there is no edge (u, w) and/or (w, u) in E) and also K is a dominating set (i.e. for every vertex $u \in V - K$, there exists a vertex $w \in K$ such that $(w, u) \in E$.)*

The next theorem establishes one more interesting connection between logic programming and graph theory.

Theorem 6 *Every stable model of a logic program P determines a kernel in the rule graph G_P of P .*

Proof. Let M be a stable model of P and let $G_P = (R \cup L \cup \{S\}, E)$ be its rule graph. Define $K_M = R' \cup L' \cup \{S\}$, where $R' = \{r_i : r_i \in R \text{ such that for every positive literal } p \in \text{body}(r_i), p \in M \text{ holds, while for every negative literal } \neg q \in \text{body}(r_i), q \notin M \text{ holds}\}$ and $L' = \{v_i : v_i \in L \text{ such that for the associated with } v_i \text{ literal } l_i, l_i \notin M \text{ holds}\}$. From Proposition 3 it follows that $M = \{p : p \in \text{head}(r) \text{ such that for the rule vertex } v \text{ associated with } r, v \in R' \text{ holds}\}$. We show that K_M is a kernel for G_P .

We first prove that K_M is independent. It suffices to prove that for any $(v_i, v_j) \in E$, both its endpoints do not belong to K_M . If $(v_i, v_j) \in E$, then (by the definition of G_P) both v_i and v_j cannot belong to L . Let $v_i \in R'$. There are two cases for v_j : (a) $v_j \in L$. Then, for the literal l_j associated with v_j , $l_j \in M$ holds, hence $v_j \notin L'$. (b) $v_j \in R$. Then, there must be a negative literal $\neg p \in \text{body}(r_j)$ which

is associated with the rule vertex v_j , such that $p \in M$. Hence, $v_j \notin R'$. Therefore, in either case $v_j \notin K_M$. Let now $v_i \in L'$. Since $v_j \notin L$, we must have $v_j \in R$. But then, v_j corresponds to a rule that contains an atom in its body and this atom is False in M . Hence, $v_j \notin R'$ and consequently $v_j \notin K_M$. Finally, let $v_i = S$. Then, $v_j \in R$ and v_j corresponds to a rule that contains an atom p in its body, with $p \notin M$. Hence $v_j \notin R'$ and thus $v_j \notin K_M$.

We now prove that K_M is dominating. It suffices to prove that for every vertex v_i in $(R - R') \cup (L - L')$, there exists an edge (x, v_i) , where $x \in K_M$. Assume that $v_i \in R - R'$. This can happen for two reasons. First, because there is a negative literal $\neg p$ in the body of the rule corresponding to v_i such that $p \in M$. But then $p \in \text{head}(r_j)$, $i \neq j$, such that for its associated rule vertex v_j , $v_j \in R'$ holds, which (by the definition of G_P) implies that $(v_j, v_i) \in E$. Second, because for some atom $p \in \text{body}(r_i)$, $p \notin M$ holds. But then, the vertex v_p associated with p belongs to L' and also $(v_p, v_i) \in E$ by construction of G_P . Assume now that $v_i \in L - L'$. Then, for the atom p associated with v_i , $p \in M$ holds. Hence, p belongs to the head of some rule r_j whose corresponding vertex v_j is in R' . Again by the definition of G_P , $(v_j, v_i) \in E$.

Hence, K_M is an independent and dominating set of vertices, therefore a kernel of G_P . \square

The converse of the above theorem does not hold in general. However, since directed acyclic graphs have a unique kernel [3] and G_P is acyclic in the cases we study, the unique kernel of G_P determines the unique stable model of P . In particular, it can be easily verified (using Proposition 3 and the proof of Theorem 6) that if K is the kernel of the acyclic rule graph $G_P = (R \cup L \cup \{S\}, E)$ of a logic program P , then the stable model of P is $M = \{p : p = \text{head}(r) \text{ such that for the rule vertex } v \text{ associated with } r, v \in R \cap K \text{ holds}\}$. Hence, we have the following:

Corollary 7 *The unique stable model of an acyclic logic program P determines the unique kernel of the rule graph G_P of P and vice versa.*

3. The kernel problem and the stable model semantics

In this section we shall prove that computing the kernel of a directed acyclic graph is \mathcal{P} -complete. To show this, we will use the *circuit value problem* (CVP). The CVP asks for the value of a single output gate of a Boolean circuit C consisting of NOT, AND and OR gates, for a given set of inputs. Also C is assumed to be a fan-in 2 circuit, that is, a circuit with gates having only two inputs. (If not otherwise stated, in this paper the term circuit will refer to fan-in 2 circuits.) More precisely, the circuit is defined as a sequence $C = \langle g_1, g_2, \dots, g_n \rangle$, where each g_i is either an input, or $g_i = g_j \vee g_k$ (an OR gate), or $g_i = g_j \wedge g_k$ (an AND gate), or $g_i = \neg g_j$ (a NOT gate), and $j, k < i$. Then, given a circuit C and a set of inputs, the CVP is to determine whether the value of g_n is equal to True. The CVP is \mathcal{P} -complete [13]. The variation of the problem, where C contains only NOR gates, is called NOR-CVP. It is easy to prove that NOR-CVP is also \mathcal{P} -complete, since the NOR gates form a complete basis for the set of Boolean functions. Note that for each

circuit, there is a corresponding directed acyclic graph which is constructed in the obvious way.

Let $G = (V, E)$ be a directed graph and let w be a vertex. Recall that a vertex $w \in V$ has indegree 0, if $|\{(y, w) : y \in V \text{ and } (y, w) \in E\}| = 0$. For each vertex $v \in V$, we define a function $\phi(v)$ as follows:

$$\phi(v) = \begin{cases} 1, & \text{if } v \text{ has indegree } 0 \\ \text{NOR}_{(y,v) \in E} \{\phi(y)\}, & \text{otherwise} \end{cases}$$

The next proposition comes easily by the above definition and Definition 5.

Proposition 8 *The kernel of a directed graph $G = (V, E)$ is the set $K = \{v \in V : \phi(v) = 1\}$.*

Let $G' = (V', E')$ be a directed acyclic graph and let $z \in V'$. We refer to the problem of determining whether z belongs to the kernel of G' , as the KERNEL problem for G' and z .

Lemma 9 *The KERNEL problem for directed acyclic graphs is \mathcal{P} -complete.*

Proof. We reduce NOR-CVP to KERNEL. The reduction can be done in \mathcal{NC}^a . Given a circuit C , i.e. an instance of NOR-CVP, let $G(C)$ be the directed acyclic graph corresponding to C . Let t be the vertex of $G(C)$ corresponding to the output gate of C . Then, solve KERNEL for $G(C)$ and vertex t . By Proposition 8, it is clear that the solution of the KERNEL problem on $G(C)$ is the same as the solution of the NOR-CVP on C (i.e. t belongs to the kernel of $G(C)$ iff the output of C is True). Hence, KERNEL is \mathcal{P} -hard. Since KERNEL is in \mathcal{P} , it is also \mathcal{P} -complete. \square

Theorem 10 *Determining whether an atom belongs to the stable model of an acyclic negative logic program is \mathcal{P} -complete.*

Proof. By Lemma 9, it suffices to reduce KERNEL to the problem stated in the theorem. Let $H = (V, E)$ be a directed acyclic graph. We can construct an associated negative logic program P_H as follows: for every vertex $v_i \in V$ add the atom h_i and the rule $h_i \leftarrow \neg h_1, \neg h_2, \dots, \neg h_k$ to P_H , where for every $1 \leq j \leq k$, $(v_j, v_i) \in E$. Let K be the kernel of H . Define $M = \{h_i : h_i \in \text{head}(r_i) \text{ and } v_i \in K\}$.

We first show that M is the stable model of P_H . To show this, it suffices to show (by Definition 2) that M is the minimal model of $G_M(P_H)$. Recall Definition 1. For every $v_i \notin K$, its corresponding rule $r_i \notin G_M(P_H)$, because $\exists v_j \in K$, $(v_j, v_k) \in E$ and $\neg h_j \in \text{body}(r_i)$. Moreover, if $v_i \in K$, then r_i cannot contain a literal $\neg h_j \in \text{body}(r_i)$ such that $h_j \in M$ (otherwise, $v_j \in K$ and thus K is not a kernel since $(v_j, v_k) \in E$). This implies that $\forall v_i \in K$, $r_i \in G_M(P_H)$. Hence, $G_M(P_H)$ consists precisely of rules of the form $h_i \leftarrow, \forall v_i \in K$. Consequently, the minimal model of $G_M(P_H)$ is identical to M .

*For a definition of the class \mathcal{NC} see e.g. [9].

Now, since H and P_H are acyclic, both K and M are unique. This and the definition of M above imply that $v_i \in K$ iff $h_i \in M$. Consequently, the KERNEL problem in H reduces to determining whether an atom belongs to the stable model of P_H . \square

4. Acyclic Definite Logic Programs

In this section we investigate the case of acyclic definite logic programs. We start with a brief discussion concerning the *Horn satisfiability* problem which justifies our restriction to this class of programs.

Let S be a set of propositional Horn clauses, that is, a definite logic program. We say that an atom A is *solvable* wrt S , if either A belongs to the head of a rule in S with an empty body, or there is a rule $A \leftarrow L_1, \dots, L_n$ in S such that every L_i is solvable. The problem of determining whether a given atom A is solvable wrt S , is called the *propositional Horn satisfiability* problem. It is easy to see that this problem is equivalent to determining whether A belongs to the minimal model of the definite logic program S . As it has been proved in [12], the propositional Horn satisfiability problem is \mathcal{P} -complete. As a direct consequence, we have that computing the minimal model of an arbitrary definite logic program is also \mathcal{P} -complete. Hence, it is natural to concentrate on the complexity of acyclic definite logic programs.

Let $B = (B_1, \dots, B_n)$ be a monotone circuit (i.e. containing only AND and OR gates) with an assignment of values to the inputs. With every gate B_i we associate an atom A_i , called the *associated atom* of B_i . (Similarly, B_i is called the *associated gate* of atom A_i .) Then, we can associate with B a definite logic program P_B , defined as follows: (i) If B_i is True, then add the rule $A_i \leftarrow$. (ii) If $B_i = B_k \wedge B_j$, then add the rule $A_i \leftarrow A_k, A_j$. (iii) If $B_i = B_k \vee B_j$, then add the rules $A_i \leftarrow A_k$ and $A_i \leftarrow A_j$. We call P_B the *associated logic program* of B .

Let the *level* of a gate B_i , denoted as $\ell(B_i)$, be defined as follows:

$$\ell(B_i) = \begin{cases} 0, & \text{if } B_i \text{ is an input gate.} \\ 1 + \min\{\ell(B_j), \ell(B_k)\}, & \text{if } B_i = B_j \vee B_k, \text{ or } B_i = B_j \wedge B_k. \end{cases}$$

The level of an atom A_i in the program P_B , denoted as $\ell(A_i)$, is defined to be equal to the level of its associated gate B_i in the circuit B . The following lemma establishes the connection between a monotone circuit and its associated logic program.

Lemma 11 *Let $B = (B_1, \dots, B_n)$ be a monotone circuit, A_1, \dots, A_n its associated atoms and P_B its associated logic program. Then the value of B_i in B is identical to the value of A_i in the minimal model of P_B .*

Proof. We shall use induction on $\ell(A_i)$. For the basis, $\ell(A_i) = 0$. Then B_i is an input gate (i.e. a vertex in the graph representing B of indegree 0). If B_i is True, then there is a rule of the form $A_i \leftarrow$, so that the minimal model of P_B must assign

True to A_i . If B_i is False, then there is no rule where A_i appears in the LHS, so the minimal model for P_B must assign False to A_i (otherwise it is not minimal).

For the induction hypothesis, assume that for each A_j such that $\ell(A_j) < k$, $\text{value}(P_j) = \text{value}(B_j)$.

Now, let $\ell(A_i) = k$. If B_i is an OR gate with inputs B_m and B_j , then the only rules with A_i on the LHS are $A_i \leftarrow A_m$ and $A_i \leftarrow A_j$. If B_i is True, then at least one of B_m or B_j is True. W.l.o.g. assume that B_m is True. By the induction hypothesis, since $\ell(A_m) < k$, $\text{value}(A_m) = \text{value}(B_m) = \text{True}$, thus A_i is True. If B_i is False, both B_m and B_j are False, and hence both A_m and A_j are False by the induction hypothesis. Then, in the minimal model A_i is assigned False, since the only rules in which A_i appears on the LHS have their RHS False (otherwise it is not minimal).

A symmetric argument proves the case where B_i is an AND gate. \square

Since the transformation of a monotone circuit to its associated definite logic program (described above) can be easily done in \mathcal{NC} , the following theorem is an immediate consequence of Lemma 11 and the result in [11] for the \mathcal{P} -completeness of the monotone CVP.

Theorem 12 *Deciding whether an atom belongs to the minimal model of a definite acyclic logic program P is \mathcal{P} -complete. This is true even if every rule in P contains at most two atoms in its body and every atom occurs at most twice in the head of a rule in P .*

5. Closing Remarks

In the previous section we showed how we can associate with any monotone circuit a definite logic program. However, the reverse is also true. Namely, with any definite logic program P we can associate an unbounded fan-in monotone circuit C_P . It turns out that the circuit C_P is essentially identical to the rule graph, G_P , of P . To see this, it suffices to consider every literal vertex as an OR gate, every rule vertex associated with a rule having empty body as an input gate set to True, and every other rule vertex as an AND gate. Finally, vertex S is to be considered as an input gate set to False. The circuit C_P associated with a definite logic program P is an unbounded fan-in one. (This is not a problem, since any circuit can be easily transformed to a fan-in 2 circuit, the size of which is a polynomial in the size of the original circuit and this transformation can be done in \mathcal{NC} .) Using arguments similar to those used in the proof of Lemma 11, we can prove that an atom p_i of P is True in the minimal model of P iff the AND or OR gate associated with p_i in C_P (depending on whether p_i occurs in the head of one or more rules) evaluates to True.

The above result establishes a direct correspondence between acyclic definite logic programs and monotone CVP. The second direct correspondence between negative logic programs and the KERNEL problem is established by Corollary 7. These results imply that whenever the monotone CVP or the KERNEL problem falls in \mathcal{NC} , the same holds for the problem of determining the stable model of their as-

sociated logic programs. Examples are circuits or directed acyclic graphs which have a tree structure or polylogarithmic depth, and monotone planar circuits (see e.g. [15]). (However, for the latter case it is not clear which logical properties could correspond to planarity of the circuit.)

Acknowledgements

We would like to thank Devdatt Dubhashi and Desh Ranjan for many helpful discussions.

References

- [1] K. Apt and M. Bezem, Acyclic Programs, *New Generation Computing* **9** (1991) 335-363.
- [2] K. Apt and R. Bol, Logic programming and negation: a survey, Tech. Rep. CS-R9402, CWI, Amsterdam, January 1994; to appear in *Journal of Logic Programming*.
- [3] C. Berge, *Graphs* (3rd revised edition, North-Holland, 1991).
- [4] Y. Dimopoulos, Classical methods in nonmonotonic reasoning, in *Proc. of the Int'l Symp. on Methodologies for Intelligent Systems (ISMIS-94)*, eds. Z. Ras and M. Zemankova, LNCS 869 (Springer-Verlag, 1994) 500-510.
- [5] Y. Dimopoulos and V. Magirou, A graph theoretic approach to default logic, *Information and Computation* **112:2** (1994) 239-256.
- [6] J. Dix, Semantics of logic programs: their intuitions and formal properties. An Overview, in *Logic, Action and Information, Proc. of the Konstanz Coll. in Logic and Information (LogIn'92)*, eds. A. Fuhrmann and H. Rott (DeGruyter, 1993); also Tech. Rep. 15/93, University of Koblenz-Landau, 1993.
- [7] W. Dowling and J. Gallier, Linear time algorithms for testing the satisfiability of propositional Horn formulae, *Journal of Logic Programming* **1:3** (1984) 267-284.
- [8] M. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness* (Freeman and Company, New York, 1979).
- [9] D. S. Johnson, A Catalog of Complexity Classes, in *Handbook of Theoretical Computer Science, Vol. A*, ed. J. van Leeuwen (Elsevier, Amsterdam, 1990) 67-161.
- [10] M. Gelfond and V. Lifschitz, The stable model semantics for logic programs, in *Proc. of the 5th Intern. Conf. and Symp. on Logic Programming*, eds. R. Kowalski and K. Bowen (MIT Press, 1990) 1070-1080.
- [11] L. Goldschlager, The monotone and planar circuit value problems are log-space complete for \mathcal{P} , *SIGACT News* **9:2** (1977) 25-29.
- [12] S. Kasif, On the parallel complexity of some constraint satisfaction problems, in *Proc. 5th National Conf. on Artificial Intelligence (AAAI-86)*, Philadelphia, PA (American Association for AI, 1986) 349-353.
- [13] R. Ladner, The circuit value problem is log space complete for \mathcal{P} , *SIGACT News* **7** (1975) 18-20.
- [14] J. Minker, An overview of nonmonotonic reasoning and logic programming, *Journal of Logic Programming* **17** (1993).
- [15] V. Ramachandran and H. Yang, An efficient parallel algorithm for the general planar monotone circuit value problem, in *Proc. 5th Symp. on Discrete Algorithms (SODA'94)* (ACM-SIAM, 1994) 622-631.