Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

PROPOSITIONAL SEMANTICS FOR DISJUNCTIVE LOGIC PROGRAMS

R. Ben-Eliyahu
R. Dechter

March 1992
CSD-920013

Propositional Semantics for Disjunctive Logic Programs

Rachel Ben-Eliyahu
Cognitive Systems Laboratory
Computer Science Department
University of California
Los Angeles, California 90024
rachel@cs.ucla.edu

Rina Dechter
Information & Computer Science
University of California
Irvine, California 92717
dechter@ics.uci.edu

Abstract

In this paper we study properties of the class of head-cycle-free extended disjunctive logic programs (HEDLPs), which includes, as a special case, all nondisjunctive extended logic programs. We show that any propositional HEDLP can be mapped in polynomial time into a propositional theory such that each model of the latter corresponds to an answer set, as defined by stable model semantics, of the former. Using this mapping, we show that many queries over HEDLPs can be determined by simply solving propositional satisfiability problems, that is, without enumerating answer sets.

This mapping suggests an alternative definition of stable model semantics, expressed in the familiar language of propositional logic, and it has several important implications: It establishes the NP-completeness of this class of disjunctive logic programs, allows existing algorithms and tractable subsets for the satisfiability problem to be used in logic programming, facilitates the evaluation of the expressive power of disjunctive logic programs, and leads to recognition of useful similarities between stable model semantics and Clark's predicate completion.

1 Introduction

Stable model semantics, proposed by Gelfond and Lifschitz [GL91], successfully bridges the gap between two lines of research, default reasoning and logic programming. Gelfond and Lifschitz pointed out the need for explicit representation of negated information in logic programs and accordingly defined *extended logic programs* as those that use classical negation in addition to the *negation-as-failure* operator.

One advantage of stable model semantics is that it is closely related to the semantics of Reiter's default logic [Rei80], in the framework of which an extended logic program may be viewed as a default theory with special features. This duality allows transference of insights, techniques, and analytical results from default logic to logic programming, and vice versa.

The work presented here puts this duality into practice. We use techniques developed for answering queries on default theories [BED91a] to compute answer sets for disjunctive logic programs (according to stable model semantics). We also show how this semantics can be given an interpretation in propositional logic.

Specifically, we show that a large class of extended disjunctive logic programs (EDLPs) can be compiled in polynomial time into a propositional theory such that each model of the latter corresponds to an answer set of the former. Consequently, query answering in such logic programs can be reduced to deduction in propositional logic. This reduction establishes the NP-completeness of various decision problems regarding query answering in such logic programs and suggests that any of a number of existing algorithms and heuristics known for solving satisfiability are now applicable for computing answer sets. Moreover, known tractable classes for satisfiability induce the identification of new tractable subsets of logic programs. As an example we introduce new tractable subsets of logic programs which correspond to tractable subsets of *constraints satisfaction problems* (CSPs).

Aside from the computational ramifications, our translation provides an alternative representation of stable model semantics, expressed in the familiar language of propositional logic. This facilitates the comparison of various semantic proposals for logic programs and also allows evaluation of their expressive power. In particular, it leads to the discovery of useful similarities between stable model semantics and Clark's predicate completion.

Our translation does not apply to the full class of EDLPs but only to a subclass (albeit a large one) of *head-cycle-free* extended disjunctive logic programs (HEDLPs). Note that this class contains any extended nondisjunctive logic program. The question of whether stable model semantics for the class of *all* disjunctive logic programs can be expressed in propositional logic in polynomial time remains open.

The rest of the paper is organized as follows: In Section 2 we review the definition of EDLPs and their stable model semantics and present some new characterizations of answer sets for such programs. Section 3 shows how an HEDLP can be mapped into a propositional theory and discusses he properties of our mapping. Section 4 illustrates four outcomes of the translation concerning the language's expressiveness, complexity, tractable classes, and relationship to Clark's predicate completion. In Section 5 we mention relevant work by others, and in Section 6 we provide concluding remarks. Omitted proofs can be found in the full paper [BED91b].

2 Extended Disjunctive Logic Programs

EDLPs are disjunctive logic programs with two types of negation: negation by default and classical negation. They were introduced by Gelfond and Lifschitz [GL91], who defined an EDLP as a set of rules of the form

$$L_1|...|L_k \longleftarrow L_{k+1}, ..., L_{k+m}, \textit{not } L_{k+m+1}, ..., \textit{not } L_{k+m+n} \qquad (1)$$

where each L_i is a literal and *not* is a negation-by-default operator. The symbol '|' is used to distinguish it from the '$\vee$' used in classical logic. A literal *appears positive* in the body of a rule if it is not preceded by the *not* operator. A literal *appears negative* in the body of a rule if it *is* preceded by the *not* operator.[1]

Example 2.1 *Suppose we know that a baby called Adi was born. We also know that a baby, if there is no reason to believe that it is abnormal, is normal and that normal babies are either boys or girls. This information could be encoded in a disjunctive logic program as follows:*

Baby(Adi) $\longleftarrow$
Normal_baby(x) $\longleftarrow$ *Baby(x), not Abnormal(x)*
Boy(x) | *Girl(x)* $\longleftarrow$ *Normal_baby(x).*

The literal Abnormal(x) appears negative in the body of the second rule. The literal Normal_baby(x) appears positive in the body of the third rule.

Gelfond and Lifschitz have generalized stable model semantics so that it can handle EDLPs. We next review this semantics with a minor modification: while Gelfond and Lifschitz's definition allows inconsistent answer sets, ours does not. Given a disjunctive logic program Π, the set of answer sets of Π under this modified semantics will be identical to the set of *consistent* answer sets under Gelfond and Lifschitz's original semantics. With slight changes, all the results in this paper apply to their semantics as well.

First, an *answer set* of an EDLP Π without variables and without the *not* operator is defined. Let $\mathcal{L}$ stand for the set of grounded literals in the language of Π. A *context of* $\mathcal{L}$, or simply "context", is any subset of $\mathcal{L}$.

An *answer set* of Π is any minimal[2] context S such that

1. for each rule $L_1|...|L_k \longleftarrow L_{k+1}, ..., L_{k+m}$ in Π, if $L_{k+1}, ..., L_{k+m}$ is in S, then for some $i = 1, ..., k$ L_i is in S, and

2. S does not contain a pair of complementary literals.[3]□

[1] Note that *positive (negative) literal* and *a literal that appears positive (negative) in a body of a rule* denote two different things (see Example 2.1).

[2] Minimality is defined in terms of set inclusion.

[3] Under Gelfond and Lifschitz's semantics this item would say, "If S contains a pair of complementary literals, then $S = \mathcal{L}$".

Suppose Π is a variable-free EDLP. For any context S of $\mathcal{L}$, Gelfond and Lifschitz define Π^S to be the EDLP obtained from Π by deleting

1. all formulas of the form *not* L where $L \notin S$ from the body of each rule and

2. each rule that has the formula *not* L where $L \in S$.

Note that Π^S has no *not* , so its answer sets were defined in the previous step. If S happens to be one of them, then we say that S is an *answer set* of Π. To apply the above definition to an EDLP with variables, we first have to replace each rule with its grounded instances.

Consider, for example, the grounded version of the program Π above:

Baby(Adi) ⟵
Normal_baby(Adi) ⟵Baby(Adi), *not* Abnormal(Adi)
Boy(Adi) | Girl(Adi) ⟵Normal_baby(Adi).

The reader can verify that Π has two answer sets:
$\{Baby(Adi), Normal\_baby(Adi), Girl(Adi)\}$ and
$\{Baby(Adi), Normal\_baby(Adi), Boy(Adi)\}$.

We will assume from now on that all programs are grounded and that their dependency graph has no infinitely decreasing chains. The *dependency graph* of an EDLP Π, G_Π, is a directed graph where each literal is a node and where there is an edge from L to L' iff there is a rule in which L appears positive in the body and L' appears in the head[4]. An EDLP is *acyclic* iff its dependency graph has no directed cycles. An EDLP is *head-cycle free* (that is, an HEDLP) iff its dependency graph does not contain directed cycles that go through two literals that belong to the head of the same rule. Clearly, every acyclic EDLP is an HEDLP. We will also assume, without losing expressive power, that the same literal does not appear more than once in the head of any rule in the program.

We next present new characterizations of answer sets. The declarative nature of these characterizations allows for their specification in propositional logic in a way such that queries about answer sets can be expressed in terms of propositional satisfiability.

We first define when a rule is satisfied by a context and when a literal has a proof w.r.t. a program Π and a context S.

A context S satisfies the body of a rule iff each literal that appears positive in the body is in S and each literal that appears negative in the body is not in S. *A context S satisfies a rule* iff either it does not satisfy its body or it satisfies its body and at least one literal that appears in its head belongs to S.

A *proof* of a literal is a sequence of rules that can be used to derive the literal from the program. Formally, a literal L has a *proof* w.r.t. a context S and a program Π iff there is a sequence of rules $\delta_1, ..., \delta_n$ from Π such that

[4]Note that our dependency graph ignores the literals that appear negative in the body of the rule.

1. for each rule δ_i, one and only one of the literals that appear in its head belongs to S (this literal will be denoted $h_S(\delta_i)$),
2. $L = h_S(\delta_n)$,
3. The body of each δ_i is satisfied by S, and,
4. δ_1 has an empty body, and for each $i > 1$, each literal that appears positive in the body of δ_i is equal to $h_S(\delta_j)$ for some $1 \leq j < i$.

The following theorem clarifies the concept of answer sets:

Theorem 2.2 *A context S is an answer set of an HEDLP Π iff*

1. *S satisfies each rule in Π,*
2. *for each literal L in S, there is a proof of L w.r.t Π and S, and*
3. *S does not contain a pair of complementary literals.* □

Note that the above theorem will not necessarily hold for programs having head cycles. Consider, for example, the program having the set of rules $\{P|Q \longleftarrow, P \longleftarrow Q, Q \longleftarrow P\}$. The set $\{P, Q\}$ is an answer set of this program but it violates condition 2 of the theorem, since neither P nor Q has a proof w.r.t. the answer set and the program.

Is there an easy way to verify that each literal has a proof? It turns out that for an acyclic EDLP the task is easier:

Theorem 2.3 *A context S is an answer set of an acyclic EDLP Π iff*

1. *S satisfies each rule in Π,*
2. *for each literal L in S, there is a rule δ in Π such that*
   (a) *the body of δ is satisfied by S,*
   (b) *L appears in the head of δ, and*
   (c) *all the literals other than L in the head of δ are not in S,*

   and
3. *S does not contain a pair of complementary literals.* □

To identify an answer set when Π is *cyclic*, we need to assign indexes to literals that share a cycle in the dependency graph:

Theorem 2.4 *A context S is an answer set of an HEDLP Π iff*

1. *S satisfies each rule in Π,*
2. *there is a function $f : \mathcal{L} \mapsto N^+$ such that, for each literal L in S, there is a rule δ in Π such that*

(a) the body of δ is satisfied by S,

(b) L appears in the head of δ,

(c) all literals in the head of δ other than L are not in S, and,

(d) for each literal L' that appears positive in the body of δ, $f(L') < f(L)$,

and

3. S does not contain a pair of complementary literals. □

The above characterizations of answer sets are very useful. In addition to giving us alternative definitions of an answer set, they facilitate a polynomial time compilation of any finite grounded HEDLP into a propositional theory, such that there is a one-to-one correspondence between answer sets of the former and models of the latter. The merits of this compilation will be illustrated in the sequel.

3 Compiling Disjunctive Logic Programs into a Propositional Theory

Each answer set of a given logic program can be viewed as representing a possible world compatible with the information expressed in the program. Hence, given an EDLP Π and a context V, the following queries might come up:

Existence: Does Π have an answer set? If so, find one or all of them.

Set-Membership: Is V contained in *some* answer set of Π?

Set-Entailment: Is V contained in *every* answer set of Π?

In this section we will present algorithms that translate a finite HEDLP Π into a propositional theory T_Π such that the above queries can be expressed as satisfiability problems on this propositional theory.

The propositional theory T_Π is built upon a new set of symbols $\mathcal{L}_\Pi$ in which there is a new symbol I_L for each literal L in $\mathcal{L}$. Formally,

$$\mathcal{L}_\Pi = \{I_L | L \in \mathcal{L}\}.$$

Intuitively, each I_L stands for the claim "The literal L is *In* the answer set", and each valuation of $\mathcal{L}_\Pi$ represents a context, which is the set of all literals L such that I_L is assigned **true** by the valuation. What we are looking for, then, is a theory over the set $\mathcal{L}_\Pi$ such that each model of the theory represents a context that is an answer set of Π.

Consider procedure *translate-1*, below, which translates an HEDLP Π into a propositional theory T_Π.

translate-1(Π)

1. For each body-free rule $L_1|...|L_k \longleftarrow$ in Π, add $I_{L_1} \vee ... \vee I_{L_k}$ into T_Π.
2. For each rule

$$L_1|...|L_k \longleftarrow L_{k+1}, ..., L_{k+m}, not\ L_{k+m+1}, ..., not\ L_{k+m+n} \qquad (2)$$

with no empty body add

$$I_{L_{k+1}} \wedge ... \wedge I_{L_{k+m}} \wedge \neg I_{L_{k+m+1}} \wedge ... \wedge \neg I_{L_{k+m+n}} \longrightarrow I_{L_1} \vee ... \vee I_{L_k}$$

into T_Π.

3. For a given $L \in \mathcal{L}$, let S_L be the set of formulas of the form

$$I_{L_{k+1}} \wedge ... \wedge I_{L_{k+m}} \wedge \neg I_{L_{k+m+1}} \wedge ... \wedge \neg I_{L_{k+m+n}} \wedge \neg I_{L_1} \wedge ... \wedge \neg I_{L_{j-1}} \wedge \neg I_{L_{j+1}} \wedge ... \wedge \neg I_{L_k}$$

where there is a rule of the form (2) in Π in which L appears in the head as L_j.

For each L in $\mathcal{L}$ such that the rule "$L\longleftarrow$" is not in Π add to T_Π the formula $I_L \longrightarrow [\vee_{\alpha \in S_L} \alpha]$ (note that if $S_L = \emptyset$ we add $I_L \longrightarrow$ **false** to T_Π).

4. For each two complementary literals L, L' in $\mathcal{L}$, add $\neg I_L \vee \neg I_{L'}$ to T_Π.

□

The theory T_Π, produced by the above algorithm, simply states the conditions of Theorem 2.3 in propositional logic: the first and second steps of algorithm *translate-1* express condition 1 of the theorem, step 3 expresses condition 2, and step 4 describes condition 3. Hence:

Theorem 3.1 *Procedure* translate-1 *transforms an acyclic EDLP* Π *into a propositional theory* T_Π *such that* θ *is a model for* T_Π *iff* $\{L|\theta(I_L) = \textbf{true}\}$ *is an answer set for* Π.

What if our program is cyclic? Can we find a theory such that each of its models corresponds to an answer set? Theorem 2.4 suggests that we can do so by assigning indexes to the literals.

When we deal with finite logic programs, the fact that each literal is assigned an index and the requirement that an index of one literal will be lower than the index of another literal can be expressed in propositional logic. Let $\#L$ stand for "L is associated with one and only one integer between 1 and n", and let $[\#L_1 < \#L_2]$ stand for "The number associated with L_1 is less than the number associated with L_2". These notations are shortcuts for formulas in propositional logic that express those assertions (see [BED91b]).

The size of the formulas $\#L$ and $[\#L_1 < \#L_2]$ is polynomial in the range of the indexes we need. We can show that the index variables' range can be bounded by the maximal length of an acyclic path in any *strongly connected component* in G_Π (the dependency graph of Π).

The strongly connected components of a directed graph are a partition of its set of nodes such that, for each subset C in the partition and for each $x, y \in C$, there are directed paths from x to y and from y to x in G. The strongly connected components can be identified in linear time [Tar72]. Thus, once again we realize that if the HEDLP is acyclic, we do not need any indexing.

The above ideas are summarized in the following theorem, which is a restricted version of Theorem 2.4 for the class of *finite* HEDLPs.

Theorem 3.2 *Let Π be a finite HEDLP, and let r be the length of the longest acyclic directed path in any component of G_Π. A context S is an answer set of an HEDLP Π iff*

1. *S satisfies each rule in Π,*
2. *there is a function $f : \mathcal{L} \mapsto 1, ..., r$ such that, for each literal L in S, there is a rule δ in Π such that*
   (a) *the body of δ is satisfied by S,*
   (b) *L appears in the head of δ,*
   (c) *all literals other than L in the head of δ are not in S, and,*
   (d) *for each literal L' that appears positive in the body of δ and shares a cycle with L in the dependency graph of Π, $f(L') < f(L)$,*

   and
3. *S does not contain a pair of complementary literals.* □

Procedure *translate-2* expresses the conditions of Theorem 3.2 in propositional logic. Its input is any finite HEDLP Π, and its output is a propositional theory T_Π whose models correspond to the answer sets of Π. T_Π is built over the extended set of symbols $\mathcal{L}_\Pi' = \mathcal{L}_\Pi \bigcup \{L = i | L \in \mathcal{L}, 1 \leq i \leq r\}$, where r is as above. Steps 1, 2, and 4 of *translate-2* are identical to steps 1, 2, and 4 of *translate-1*, so we will show only step 3.

translate-2(Π)-step 3

3 Identify the strongly connected components of G_Π. For each literal L that appears in a component of size > 1, add $\#L$ to T_Π.

For a given $L \in \mathcal{L}$, let S_L be the set of all formulas of the form

$$I_{L_{k+1}} \wedge ... \wedge I_{L_{k+m}} \wedge \neg I_{L_{k+m+1}} \wedge ... \wedge \neg I_{L_{k+m+n}} \wedge \neg I_{L_1} \wedge ... \wedge \neg I_{L_{j-1}} \wedge \neg I_{L_{j+1}} \wedge ... \wedge \neg I_{L_k} \wedge [\#L_{k+1} < \#L] \wedge ... \wedge [\#L_{k+t} < \#L]$$

such that there is a rule in Π

$$L_1|...|L_k \longleftarrow L_{k+1}, ..., L_{k+m}, not\ L_{k+m+1}, ..., not\ L_{k+m+n}$$

in which L appears in the head as L_j and $L_{k+1}, ..., L_{k+t}$ $(t \leq m)$ are in L's component.

For each L in $\mathcal{L}$ such that the rule "$L\longleftarrow$" is not in Π add to T_Π the formula $I_L \longrightarrow [\vee_{\alpha \in S_L} \alpha]$ (note that if $S_L = \emptyset$ we add $I_L \longrightarrow$ **false** to T_Π). □

Note that if *translate-2* gets as an input an acyclic HEDLP it will behave exactly the same as *translate-1*, thus it is a generalization of *translate-1*.

The following proposition states that the algorithm's time complexity and the size of the resulting propositional theory are both polynomial:

Proposition 3.3 *Let* Π *be an HEDLP. Let* $|\Pi|$ *be the number of rules in* Π, *n the size of* $\mathcal{L}$, *and r the length of the longest acyclic path in any component of* G_Π. *Algorithm* translate-2 *runs in time* $O(|\Pi|n^2r^2)$ *and produces* $O(n + |\Pi|)$ *sentences of size* $O(|\Pi|nr^2)$.

The following theorems summarize the properties of our transformation. In all of them, T_Π is the set of sentences resulting from translating a given HEDLP Π using *translate-2* (or *translate-1* when the program is acyclic).

Theorem 3.4 *Let* Π *be an HEDLP. If* T_Π *is satisfiable and if* θ *is a model for* T_Π , *then* $\{L|\theta(I_L) = \mathbf{true}\}$ *is an answer set for* Π.

Theorem 3.5 *If S is an answer set for an HEDLP* Π, *then there is a model* θ *for* T_Π *such that* $\theta(I_L) = \mathbf{true}$ *iff* $L \in S$.

Corollary 3.6 *An HEDLP* Π *has an answer set iff* T_Π *is satisfiable.*

Corollary 3.7 *A context V is contained in some answer set of an HEDLP* Π *iff there is a model for* T_Π *that satisfies the set* $\{I_L|L \in V\}$.

Corollary 3.8 *A literal L is in every answer set of an HEDLP* Π *iff every model for* T_Π *satisfies* I_L.

The above theorems suggest that we can first translate a given HEDLP Π to T_Π and then answer queries as follows: to test whether Π has an answer set, we test satisfiability of T_Π; to see whether a set V of literals is a member in some answer set, we test satisfiability of $T_\Pi \bigcup \{I_L|L \in V\}$; and to find whether V is included in every answer set, we test whether $T_\Pi \models \wedge_{L \in V} I_L$.

Example 3.9 *Consider again the program* Π *from the previous section ("Ba" stands for "Baby(Adi)", "Bo" stands for "Boy(Adi)", and each of the other literals is represented by its initial):*

$$Ba\longleftarrow$$
$$N\longleftarrow Ba, not\ A$$
$$Bo|G\longleftarrow N$$

The theory T_Π, produced by algorithm translate-2 *(and also by algorithm* translate-1*, since this program is acyclic), is as follows:*

{*following step 1:*

$$I_{Ba}$$

following step 2:

$$I_{Ba} \wedge \neg I_A \longrightarrow I_N,\ I_N \longrightarrow I_{Bo} \vee I_G$$

following step 3:

$$I_N \longrightarrow I_{Ba} \wedge \neg I_A,\ I_{Bo} \longrightarrow I_N \wedge \neg I_G,\ I_G \longrightarrow I_N \wedge \neg I_{Bo},\ \neg I_A$$

(no sentences will be produced in step 4 since there are no complementary literals in $\mathcal{L}$)

}.

This theory has exactly two models (we mention only the atoms to which the model assigns **true***): $\{I_{Ba}, I_N, I_G\}$, which corresponds to the answer set* { *Baby(Adi), Normal_baby(Adi), Girl(Adi)* }*, and $\{I_{Ba}, I_N, I_{Bo}\}$, which corresponds to the answer set* { *Baby(Adi), Normal_baby(Adi), Boy(Adi)* }*.*

4 Outcomes of Our Translation

4.1 NP-completeness

Gelfond and Lifschitz [GL91] have shown that there is a close relationship between default theories and logic programs interpreted by stable model semantics. Their observation allows techniques and complexity results obtained for default logic to be applied to logic programming, and vice versa. For example, the complexity results obtained for default logic [KS91, Sti90] establish the NP-hardness of the existence and membership problems and the co-NP-hardness of the entailment problem for the class of HEDLPs.

In view of these results, the polynomial transformation to satisfiability that we have presented in the last section implies the following:

Corollary 4.1 *The existence problem for the class HEDLP is NP-complete.*

Corollary 4.2 *The membership problem for the class HEDLP is NP-complete.*

Corollary 4.3 *The entailment problem for the class HEDLP is co-NP-complete.*

4.2 Tractability

The results obtained in the last subsection suggest that in general many types of queries on a disjunctive logic program are NP-hard or co-NP-hard under stable model semantics.

Our mapping of HEDLPs into propositional theories suggests a new dimension along which tractable classes can be identified. Since our transformation is tractable, any subset of HEDLPs that is mapped into a tractable

subset of satisfiability is tractable too. Among other known techniques, we can apply techniques from the CSP literature that solve satisfiability and characterize tractable subsets by considering the topological structure of the problem (for a survey, see [Dec92]).

For instance, in [BED91a] we have shown how a CSP technique called *tree-clustering*, developed by Dechter and Pearl [DP89], can be used to solve satisfiability. Based on this technique, we can characterize the tractability of HEDLPs as a function of the topology of their *interaction graphs*. The *interaction graph* is an undirected graph where each literal L in $\mathcal{L}$ is associated with a node and the set of all literals that appear in rules with L in the head are connected in a clique.

The first theorem considers the *clique width* of the interaction graph.

Definition 4.4 (clique width) *A graph is chordal if every cycle of length at least* 4 *has a chord. The* clique width *of a graph G is the minimal size of a maximal clique in any chordal graph that embeds G.*

Theorem 4.5 *For an HEDLP whose interaction graph has a clique width q, existence, membership, and entailment can be decided in $O(\pi^4(2r)^{q+2})$ steps.*

In the above theorem and in the next, π stands for the number of rules or the number of literals used in the program, whichever is bigger, and r stands for the length of the longest acyclic path in any component of the dependency graph of the program (so if the program is acyclic, $r = 1$).

Note that an upper bound to the clique width is the number of literals used in the program and that the upper bound is always at least as large as the size of the largest rule in the program. We believe, however, that tree-clustering is especially useful for programs whose interaction graphs have a repetitive structure. Programs for temporal reasoning, where the temporal persistence principle causes the knowledge base to have a repetitive structure are a good example. In [BED91b] we demonstrate the usefulness of tree-clustering for answering queries on such programs.

The next theorem relates the complexity to the size of the cycle-cutset of the interaction graph.

Theorem 4.6 *For an HEDLP (D, W) whose interaction graph has a cycle-cutset of cardinality c, existence, membership, and entailment can be decided in $O(\pi^4(2r)^{c+4})$ steps.*

4.3 Expressiveness

Are disjunctive rules really more expressive than nondisjunctive rules? Can we find a nondisjunctive theory for each disjunctive theory such that they have the same answer sets/extensions? This question has been raised by Gelfond *et al* [GPLT91]. They consider translating a disjunctive logic program

Π into a nondisjunctive program Π' by replacing each rule of the form (1) (see Section 2 above) with k new rules

$$L_1 \longleftarrow L_{k+1}, ..., L_{k+m}, not\ L_{k+m+1}, ..., not\ L_{k+m+n}, not\ L_2, ..., not\ L_k,$$

$$\vdots$$

$$L_k \longleftarrow L_{k+1}, ..., L_{k+m}, not\ L_{k+m+1}, ..., not\ L_{k+m+n}, not\ L_1, ..., not\ L_{k-1}.$$

Gelfond *et al* show that each extension of Π' is also an extension of Π, but not vice versa. They gave an example where Π has an extension while Π' does not. So, in general, Π' will not be equivalent to Π. We can show, however, that equivalence does hold for HEDLPs.

Theorem 4.7 *Let Π be an HEDLP. S is an answer set for Π iff it is an answer set for Π'.* □

The reader can verify the above theorem by observing that our translation will yield the same propositional theory for both Π and Π'. The theorem implies that under stable model semantics no expressive power is gained by introducing disjunction unless we deal with a special case of recursive disjunctive logic programs.

4.4 Relation to Clark's Predicate Completion

Clark [Cla78] made one of the first attempts to give meaning to logic programs with negation ("normal programs"). He has shown how each normal program Π may be associated with a first-order theory $COMP(\Pi)$, called its *completion.* His idea is that when a programmer writes a program Π, the programmer actually has in mind $COMP(\Pi)$, and thus a formula Q is implied by the program iff $COMP(\Pi) \models Q$ (See [BED91b] for review of these well-known results).

The translation that we provided in Section 3 leads to the discovery of a close relationship between stable model semantics[5] and Clark's predicate completion.

Theorem 4.8 *Let Π be a normal acyclic propositional logic program. Then M is a model for $COMP(\Pi)$ iff $\{I_P | P \in M\}$ is a model for T_Π.* □

proof (sketch): Let Π be an acyclic normal logic program, $\mathcal{L}$ the language of Π, and T'_Π the theory obtained from T_Π by replacing each occurrence of the atom I_P, where P is an atom in $\mathcal{L}$, with the symbol P. It is easy to see that the set of models of T'_Π projected on $\mathcal{L}$ is equivalent to the set of models of $COMP(\Pi)$. □

[5]Note that in normal logic programs we have only one negation operator, which we regard as a negation by default operator for the sake of computing answer sets.

Corollary 4.9 *Let* Π *be an acyclic normal propositional logic program.* Π *has a stable model iff* $COMP(\Pi)$ *is consistent. Furthermore,* M *is a model for* $COMP(\Pi)$ *iff* M *is an answer set for* Π. $\square$

Corollary 4.10 *Let* Π *be an acyclic normal propositional logic program. An atom* P *is in the intersection of the answer sets of* Π *(as defined by stable model semantics) iff* $COMP(\Pi) \models P$. $\square$

Corollary 4.11 *Let* Π *be an acyclic normal propositional logic program. An atom* P *does not belong to any of the answer sets of* Π *(as defined by stable model semantics) iff* $COMP(\Pi) \models \neg P$. $\square$

It is already known that each stable model for a normal logic program is a model of its completion [MS89] and that if an atom is implied by the completion of a locally stratified normal program, then it belongs to its (unique) answer set [ABW88, Prz89]. We believe that the observations in Corollaries 4.9-4.11 are new because they identify the class of *acyclic* normal propositional logic programs as a class for which stable model semantics (under "skeptical reasoning"[6]) is equivalent to Clark's predicate completion.

5 Related Work

Elkan [Elk90] has shown that stable models of *normal* logic programs can be represented as models of propositional logic, but did not provide an explicit propositional theory to characterize these models. The mapping described in Section 3 above can be regarded as one such characterization.

Marek and Truszczynski [MT91] have shown how questions of membership in expansions of an autoepistemic theory can be reduced to propositional provability and have stated that this reduction can be used for checking whether an atom is in the intersection of all stable models of a *normal* logic program. Since they do not provide an explicit algorithm for such decisions, it is hard to assess the complexity of their method, and it is not clear whether it would yield results similar to those presented here.

6 Conclusions

This paper provides several characterizations of answer sets of HEDLPs according to stable model semantics. It shows that any grounded HEDLP can be mapped in polynomial time into a propositional theory such that models of the latter and answer sets of the former coincide. This allows techniques developed for solving satisfiability problems to be applied to logic programming. It also enables evaluation of the expressive power of these programs,

[6] "Skeptical reasoning" means that a program entails an atom iff the atom belongs to all of the program's answer sets.

identification of their tractable subsets, and discovery of useful similarities between stable model semantics and Clark's predicate completion.

One of the possible drawbacks of stable model semantics is that it entails multiple answer sets. The approach proposed in this paper suggests that in order to compute whether a literal belongs to one or all answer sets we do not need to compute or count those sets. Thus, multiplicity of answer sets may not in itself be a severe computational obstacle to the practicality of disjunctive logic programming.

We are currently investigating extensions of our work to a class of ungrounded logic programs.

Acknowledgments

This work was partially supported by NSF grant IRI-9157636, by the Air Force Office of Scientific Research grant AFOSR 900136, by GE Corporate R&D, and by Toshiba of America.

The authors thank Michael Gelfond, Vladimir Lifschitz, Jack Minker, Stott Parker, Teodor Przymusinski, and Carlo Zaniolo for helpful conversations on the relations between logic programming and default reasoning. Michael Gelfond pointed out to us the difference between the uses of disjunction in default logic and in logic programs. Vladimir Lifschitz drew our attention to the close connection between our translation and Clark's predicate completion. We also thank Judea Pearl and Halina Przymusinska for valuable comments on earlier drafts of this paper.

References

[ABW88] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programs*, pages 89–148. Morgan Kaufmann, 1988.

[BED91a] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for default logic. Technical Report R-172, Cognitive systems lab, UCLA, 1991. Presented at the 4th international workshop on nonmonotonic reasoning, May 1992, Plymouth, Vermont. A short version appears in the proceedings of AAAI-91 under the name "Default Logic, Propositional logic, and Constraints".

[BED91b] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. Technical Report R-170, Cognitive systems lab, UCLA, 1991. Submitted.

[Cla78] Keith L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[Dec92] Rina Dechter. Constraint networks. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 276–285. John Wiley, 2nd edition, 1992.

[DP89] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[Elk90] Charles Elkan. A rational reconstruction of nonmonotonic truth maintenance systems. *Artificial Intelligence*, 43:219–234, 1990.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GPLT91] Michael Gelfond, Halina Przymusinska, Vladimir Lifschitz, and Miroslaw Truszczynski. Disjunctive defaults. In *KR-91: Proceedings of the 2nd international conference on principles of knowledge representation and reasoning*, pages 230–237, Cambridge,MA, 1991.

[KS91] Henry A. Kautz and Bart Selman. Hard problems for simple default logics. *Artificial Intelligence*, 49:243–279, 1991.

[MS89] Wiktor Marek and V.S. Subrahmanian. The relationship between logic program semantics and non-monotonic reasoning. In *Logic Programming: Proceedings of the 6th International conference*, pages 600–617, Lisbon, Portugal, June 1989. MIT Press.

[MT91] Wiktor Marek and Miroslaw Truszczynski. Computing intersection of autoepistemic expansions. In *Logic Programming and Non-monotonic Reasoning: Proceedings of the 1st International workshop*, pages 37–50, Washington, DC USA, July 1991.

[Prz89] Teodor Przymusinski. On the declarative and procedural semantics of logic programs. *journal of Automated Reasoning*, 5:167–205, 1989.

[Rei80] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Sti90] Jonathan Stillman. It's not my default: The complexity of membership problems in restricted propositional default logics. In *AAAI-90: Proceedings of the 8th national conference on artificial intelligence*, pages 571–578, Boston,MA, 1990.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.