A FIXPOINT SEMANTICS FOR DISJUNCTIVE LOGIC PROGRAMS

JACK MINKER AND ARCOT RAJASEKAR

We present a fixpoint semantics for disjunctive logic programs. We extend the concept of the Herbrand base of a logic program to consist of all positive clauses that may be formed using the atoms in the Herbrand base. A monotonic closure operator is defined, operating on the lattice formed by the power set of the extended Herbrand base. The closure operator is shown to achieve a least fixpoint which captures the intended meaning of derivability of disjunctive programs. The equivalence of the fixpoint semantics with the minimal model semantics is also shown. We provide a characterization for Minker's generalized closed-world assumption using the fixpoint operator. We introduce the concept of support for negation and develop a proof procedure for handling negation based on this concept. We describe a proof procedure based on SLINF derivation, a modification of SLI derivation (LUST resolution). We show that the proof procedure reduces to SLDNF resolution when applied to Horn programs.

1. INTRODUCTION

The semantics of a program deals with the definition of its intended meaning based on some interpretation. For logic programs two such definitions apply: procedural semantics and declarative semantics. Both of these semantics deal with positive consequences and theories of negation. A procedural semantics provides an implementation-independent proof procedure for deriving inferences from logic programs and is based on proof theory. Declarative semantics are defined using some interpretation (normally a Herbrand interpretation). Model-theoretic semantics specify a declarative semantics based on models which capture the logical consequences of programs. Fixpoint semantics are an alternative form of declarative semantics.
semantics based on closure operators. Theories of negation define the negative information that can be assumed by default from logic programs and is based on some rule of negation.

Fixpoint theory has been used to define semantics for definite logic programs. Van Emden and Kowalski [19] define a closure operator \( T_P \) of a program \( P \), the least fixpoint of which corresponds to the success set of a definite logic program. Apt and van Emden [3] use the operator \( T_P \) to define a finite-failure semantics and show that the finite-failure set corresponds to the complement of \( T_P \downarrow \omega \). SLD resolution is a procedural counterpart for the declarative semantics. SLDNF resolution has been shown to be sound and complete with respect to the finite-failure semantics when used on Horn logic programs.

In this paper we propose a fixpoint semantics for disjunctive logic programs. We extend the concept of the Herbrand base of a program, \( HB(P) \), to define a set of positive ground clauses called the extended Herbrand base, \( EHB(P) \). We define the extended Herbrand base of a program to consist of the set of all positive clauses that can be formed from the ground atoms in the Herbrand base. Hence, the Herbrand base, \( HB(P) \), is contained in \( EHB(P) \).

We define a closure transformation \( T_P^I \) over the extended Herbrand base of a program \( P \) and develop a declarative fixpoint semantics which captures the intended meaning of derivability of disjunctive logic programs. We show that the operator is monotonic and achieves a least fixpoint, since the extended Herbrand base over which it operates is a complete lattice under set inclusion. The least fixpoint of \( T_P^I \) (say \( S \)) has the property that \( S \) is the set of all positive ground clauses derivable from \( P \). We show that the fixpoint semantics proposed here extends the fixpoint semantics developed by van Emden and Kowalski for Horn programs [19].

We develop a theory of negation based on the proposed fixpoint semantics and show that it is consistent with the generalized closed-world assumption [9]. We also propose a proof procedure called SLINF resolution for answering negative queries in disjunctive logic programs. SLINF resolution is the counterpart to SLDNF resolution [4,8]. The procedure is based on a concept called support for negation for an atom \( A \), which defines a set of ground clauses which need to be logical consequences of the program to assume the negation of \( A \).

The next section provides the necessary background and motivation for developing the fixpoint semantics. Section 3 defines the fixpoint operator \( T_P^I \) and develops the declarative semantics. Section 4 discusses negation, and Section 5 describes a proof procedure for answering queries in disjunctive logic programs. Section 6 contrasts our approach with other approaches.

2. BACKGROUND AND MOTIVATION

2.1. Basic Definitions

We consider disjunctive logic programs to consist of a finite set of program clauses of the form

\[
A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m, \quad n \geq 1, \quad m \geq 0,
\]
where the expression on the left-hand side of the implication is a disjunction of atoms and the expression on the right-hand side is a conjunction of atoms. The expression on the left-hand side of the implication sign is called the **head** and the one on the right-hand side is called the **body** of the clause. **Horn** programs are a subclass of disjunctive logic programs with only one atom in the head \((n = 1)\). **General Horn** programs are Horn programs with literals in the body of the clauses. An **assertion** clause is a program clause that has no body. A **goal** clause is of the form

\[
\leftarrow B_1, \ldots, B_m, \quad m \geq 0,
\]

where the term \(B_1, \ldots, B_m\) is a conjunction of atoms. In this paper we also use the following alternative notation for program clauses and goal causes respectively:

\[
A_1 \lor \cdots \lor A_n \lor \neg B_1 \lor \cdots \lor \neg B_m, \quad n \geq 1, \quad m \geq 0,
\]

and

\[
\neg B_1 \lor \cdots \lor \neg B_m, \quad m \geq 0.
\]

The **Herbrand universe** \(U_P\) of a logic program \(P\) is the set of all ground terms which can be formed from the constants and function symbols that appear in \(P\) (if no constants appear in \(P\), then an arbitrary constant is placed in \(U_P\)). The **Herbrand base** \(HB(P)\) of a logic program \(P\) is defined as the set of all ground atoms which can be formed by using predicates from \(P\) with ground terms from the Herbrand universe \(U_P\) as arguments [8]. A **Herbrand interpretation** \(I\) for \(P\) is a subset of the **Herbrand base** of \(P\), in which all atoms are assumed to be **true**, while those not in \(I\) are assumed to be **false**. By a ground instance \(E\theta\) of a program clause, we mean that there is a substitution \(\theta\) for variables in the program clause \(E\) such that \(E\theta\) is ground. A **model** of a logic program \(P\) is a Herbrand interpretation of \(P\) that makes all clauses in \(P\) true.

### 2.2. Fixpoint Semantics for Horn Programs

Let \(T\) be a closure operator, \(T : S \rightarrow S\), operating on a set \(S\), which has the partial order relation \(\subseteq\). If \(X\) is a subset of \(S\), then \(a \in S\) is an upper bound of \(X\) if \(x \subseteq a\) \(\forall x \in X\). An element \(a \in S\) is the least upper bound (lub) of \(X\) of \(S\) if \(a\) is an upper bound of \(X\) and for all upper bounds \(a'\) of \(X\), we have \(a \subseteq a'\). We can define a greatest lower bound (glb) of \(X\) in a similar manner. \(S\) is a **complete lattice** if \(\text{lub}(X)\) and \(\text{glb}(X)\) exist for every subset \(X\) of \(S\). We say \(X\) (\(X \subseteq S\)) is **directed** if every finite subset of \(X\) has an upper bound in \(X\). An operator \(T\) is continuous if it operates on a complete lattice \(S\) and \(T(\text{lub}(X)) = \text{lub}(\{T(M) | M \in X\})\) for every directed subset \(X\) of \(S\). An operator \(T\) is monotonic if for \(X_1, X_2 \in S\) we have that \(X_1 \subseteq X_2\) implies \(T(X_1) \subseteq T(X_2)\). \(X\) is a fixpoint (fp) of \(T\) if \(T(X) = X\). \(X\) is the least fixpoint (lfp) of \(T\) if \(X\) is a subset of all other fixpoints. A monotonic operator has a least fixpoint. A continuous operator is also monotonic. See Lloyd [8] for additional details.

The power set \(2^{\text{HB}(P)}\) of the Herbrand base of a program \(P\) is a complete lattice under set inclusion. van Emde and Kowalski [19] define a closure operator (called a fixpoint operator \(T_P\)) that maps a Herbrand interpretation to a Herbrand interpretation of a program \(P\).
Definition 1 [19]. Let $I$ be a Herbrand interpretation of $P$. Then

$$T_P(I) = \{ A \in \text{hb}(P) | A \leftarrow B_1, B_2, \ldots, B_m, m \geq 0 \text{ is a ground instance of a program clause in } P, \text{ and } \{ B_1, \ldots, B_m \} \subseteq I \}.$$ 

Van Emden and Kowalski have shown that the operator $T_P$, defined above, is monotonic for Horn programs and hence has a least fixpoint. The least fixpoint is also shown to define the intended meaning of a Horn program in the sense that the least fixpoint of the program is a Herbrand interpretation $I$ such that an atom is in $I$ if and only if it is a logical consequence of the program. Apt [1] shows that the operator $T_P$ can be applied to a general Horn program $P$ and that the pre-fixpoints of $T_P$ characterize models of $P$.

2.3. Motivation

The declarative fixpoint semantics defined above fails to convey the intended meaning of derivability of a program when the program is a disjunctive logic program.

Note. In all our examples we use $p, q, r, s,$ and $t$ as predicate symbols, $a, b, c, d,$ and $e$ as constants, and $f, g,$ and $h$ as function symbols.

Example 1. Consider the program $P: p(a) \lor q(b)$, and rewrite $P$ as $P_1: p(a) \leftarrow q(b)$. If we modify Definition 1 to permit literals in the right-hand side instead of atoms, as in [1], and apply the fixpoint operator, we obtain

$$T_{P_1} \uparrow \omega = \{ p(a) \}.$$ 

When we rewrite $P$ as $P_2: q(b) \leftarrow p(a)$ and apply the revised fixpoint operator, we obtain

$$T_{P_2} \uparrow \omega = \{ q(b) \}.$$ 

But neither $\{ p(a) \}$ nor $\{ q(b) \}$ is a logical consequence of $P$. □

One of the reasons for the inconsistency of the fixpoint semantics is the nonmonotonicity of the fixpoint operator. That is, the operator does not necessarily achieve a fixpoint for disjunctive logic programs. Another reason for the failure is that the operator is applied to a domain consisting of atoms whereas the logical consequences of disjunctive logic programs consist of clauses. Therefore, a fixpoint semantics for disjunctive programs should be based on a lattice related to sets of clauses, not on one restricted to atoms. In our approach we use a lattice which is formed using sets of positive clauses. We define such a clausal set called the extended Herbrand base, consisting of positive clauses formed using atoms from the Herbrand base.

We consider disjunctive programs to contain function symbols, and the theory we develop here is applicable for disjunctive programs containing function symbols. A clause is positive when it consists only of atoms.

Definition 2. Let $\text{ehb}^k(P)$, the extended Herbrand base of size-$k$ positive clauses of the program $P$, be the set of all positive clauses formed by taking the disjunction
of \( k \) distinct ground atoms from the Herbrand base, \( \mathbb{H}(P) \), of the program \( P \). Let \( EHB_k(P) \) be defined as follows:

\[
EHB_k(P) = \bigcup_{i=1}^{k} EHB^i(P).
\]

We define \( EHB(P) \) to be the extended Herbrand base of the program \( P \), as

\[
EHB(P) = EHB_k(P) = \bigcup_{i=1}^{\infty} EHB^i(P).
\]

\( EHB^i(P) \) is the Herbrand base of the program \( P \) itself, that is \( EHB^0(P) = \mathbb{H}(P) \).

**Example 2.** Consider the program \( P = \{ r(a), p(X) \lor q(X) \leftarrow r(X) \} \). Then

\[
EHB^0(P) = EHB_0(P) = \{ \},
\]

\[
EHB^1(P) = EHB_1(P) = \{ p(a), q(a), r(a) \},
\]

\[
EHB^2(P) = \{ p(a) \lor q(a), p(a) \lor r(a), q(a) \lor r(a) \},
\]

\[
EHB^3(P) = \{ p(a), q(a), r(a), p(a) \lor q(a), p(a) \lor r(a), q(a) \lor r(a) \},
\]

\[
EHB^4(P) = \{ p(a) \lor q(a) \lor r(a) \}.
\]

**Lemma 1.** Given a disjunctive program \( P \) and a positive ground clause \( C \), if \( P \) derives \( C \), then there exists a clause \( C' \leftarrow B_1, B_2, \ldots, B_m \) which is a ground instance of a program clause in \( P \) such that \( C' \) is a subclause of \( C \).

**Proof.** Let \( C \) be a ground clause derivable from \( P \), and assume that for any subclause \( K \) of \( C \), there is no program clause \( D \) in \( P \) such that \( D \leftarrow C' \leftarrow B_1, \ldots, B_m \), \( n \geq 0 \), and some substitution \( \theta \) s.t. \( C'\theta = K \). We show that we will never reach a refutation using \( P \cup \{ \neg C \} \), contradicting the hypothesis that \( C \) is derivable from \( P \). We show this by induction on the number of steps in a linear resolution. That is, we show that the clauses generated during a linear refutation of \( C \) will contain one or more positive literals which are not in \( C \).

**Base case:** First step in the resolution. Let \( \neg A \) be the literal selected from \( \neg C \), and \( D : A' \lor C' \leftarrow B_1, \ldots, B_n \) be the program clause used in the resolution step.
Then there exists a substitution $\gamma$ s.t. $A = A'\gamma$. From our assumption there exists a positive literal $B$ in $C'$ s.t. $C$ does not contain $B\theta$ for any substitution $\theta$.

Hence the resolvent contains at least one positive literal which is not in $C$.

**Induction hypothesis**: Clauses generated in less than $r$ steps contain one or more positive literals which are not in $C$.

**Induction step**: $r$th step in the resolution. Let the $(r - 1)$th resolvent in a linear derivation be $R_{r-1}$, and let $L$ be a positive literal in $R_{r-1}$ such that $L\theta \notin C$ for any substitution $\theta$ (i.e., $L$ is not an atom in $C$). This clause can resolve with either a clause in $P \cup \{\neg C\}$ or a clause generated in the previous steps. We examine the cases:

**Case 1**: Resolve a program clause or a literal from the negation of $C$ with $L_i \neq L$, as the resolving literal. Let $\theta$ be the substitution used in the resolution step. Then an instantiation of $L_i, L\theta$, remains in $R_r$.

**Case 2**: Resolve a program clause with $L$ as the resolving literal. $R_r$ contains at least the same number of positive literals which are not in $C$ as $R_{r-1}$, since no program clause has a consequent made only of atoms in $C$ (assumption), which is the only case which can reduce the number of atoms which are not in $C$.

**Case 3**: Resolve with an ancestor clause. By the induction hypothesis, all the ancestors contain one or more positive literals which are not in $C$. So the analysis is similar to that used for program clauses (cases 1, 2), and $R_r$ contains at least the same number of positive literals which are not in $C$ as $R_{r-1}$.

Each of these cases generates a clause containing at least one positive literal which is not in $C$. Therefore, by the completeness of linear resolution, for any resolvent $r$ the null clause is never generated. This is a contradiction of the assumption that $C$ is derivable from $P$.

Hence, there exists a program clause of the form

$$C' \leftarrow B_1, \ldots, B_m \text{ in } P, \quad 1 \leq i \leq n,$$

such that $C'\theta \subseteq C$, where $C$ is a ground positive clause derivable from $P$.

This condition states that a positive ground clause $C$ is derivable from a program $P$ only when there is a subclause of $C$ which forms the head of a ground instance of a program clause in $P$. The usefulness of this condition is twofold: one, it motivates a fixpoint declarative semantics, and two, it can be used to define a proof procedure for answering queries in disjunctive programs. A similar but trivial condition (and the resulting extensions) also holds for Horn programs: a ground atom $A$ is derivable from a Horn program only if $A$ is in the head of a ground instance of a program clause. In [19], van Emde and Kowalski view the resolution process as procedure invocation. Each goal results in one or more subgoals to be solved, the subgoals being the procedures derived in the body of the resolved program clause. The fixpoint semantics in [19] is a direct result of this procedural interpretation. A similar interpretation can be identified for disjunctive programs also, as seen from the lemma given below.
Lemma 2. Let $P$ be a disjunctive program and $C$ a ground positive clause. If $P$ derives $C$ in $m$ resolution steps, then there is some partition of $C$ into $C' \lor C_1 \lor \cdots \lor C_n$ such that there exist ground clauses $B_1 \lor C_1, \ldots, B_n \lor C_n$ derivable from $P$, where

$\forall 1 \leq i \leq n, C_i$ are positive ground clauses (possibly empty),

$C'$ is a positive ground clause,

$B_1, \ldots, B_n$ are ground atoms,

$\forall i, 1 \leq i \leq n, B_i \lor C_i$ are derivable from $P$ in less than or equal to $m$ resolution steps, and

$C' \leftarrow B_1, \ldots, B_n$ is a ground instance of a program clause in $P$.

PROOF.

$P$ derives $C$

$\Rightarrow$ there exists a ground instance of a program clause $CL = C' \leftarrow B_1, \ldots, B_n \in P$ which takes part in the linear derivation of $C$ from $P$ from Lemma 1

$\Rightarrow$ there exists an SL refutation from $P \cup \{-C\}$ with top clause $CL$ having $\neg B_n$ as the rightmost literal (and resolved upon first)

$\Rightarrow P \cup \{-C\}$ derives $B_n$, since there exists a resolvent $\neg B_n$ in the SL refutation from $P \cup \{-C\}$

$\Rightarrow P$ derives $B_n \lor C_n$, where $C_n$ is a subclause of $C$.

Also note that $P$ derives $B_n \lor C_n$ in at most as many steps as there are in a derivation of $C$ from $P$, since $\neg B_n$ is a resolvent in the SL refutation from $P \cup \{-C\}$.

Similar arguments can be given $\forall i, 1 \leq i \leq n, B_i \lor C_i$: We can rewrite the top clause $CL$ s.t. $\neg B_i$, $1 \leq i \leq n$, is the rightmost literal and use the same argument as above to show $P$ derives $\forall i, 1 \leq i \leq n, B_i \lor C_i$.

Hence there exist clauses $\forall i, 1 \leq i \leq n, B_i \lor C_i$, which are derivable in at most as many resolution steps as there are in the derivation of $C$ from $P$, where $C_i$ is a subclause of $C$. \( \square \)

The above lemma shows that a disjunctive goal can be reduced to subgoals which require fewer resolution steps to solve. In the next section we use the motivation provided by the above results to define a closure operator and develop a fixpoint semantics for a disjunctive logic programs.

3. FIXPOINT SEMANTICS

3.1. Closure Operator

Definition 3. A state of a program $P$ is a subset of the extended Herbrand base of $P$, $\text{EHB}(P)$. A derivable state of a program $P$ is a state in which all clauses are derivable from $P$:

$s\text{der}(P) = \{S | S$ is a derivable state of $P\}$. \( \square \)
The set of all states of a program $P$ is the power set of $\text{EHB}(P)$, $2^{\text{EHB}(P)}$. The power set is a complete lattice under the partial order of set inclusion $\subseteq$. The bottom element of the lattice is the null set, $\emptyset$, and the top element is $\text{EHB}(P)$. The closure operator that maps states to states of a program $P$ is defined as follows:

**Definition 4.** For a program $P$, a mapping $T_P^I : 2^{\text{EHB}(P)} \rightarrow 2^{\text{EHB}(P)}$ is defined as follows: Let $S$ be state of a program $P$ [i.e., $S$ is a subset of $\text{EHB}(P)$]. Then

$$T_P^I(S) = \{ C \in \text{EHB}(P) | C' \leftarrow B_1, B_2, \ldots, B_n \text{ is a ground instance of a program clause in } P \text{ and } B_1 \vee C_1, \ldots, B_n \vee C_n \text{ are in } S \text{ and } C'' = C_1 \vee \cdots \vee C_n, \text{ where } \forall i, 1 \leq i \leq n, C_i \text{ can be null, and } C \text{ is the smallest factor of } C'' \}. \tag*{\Box}$$

The superscript $I$ in the operator $T_P^I$ is used to distinguish the operator from that given by van Emden and Kowalski [19] for Horn programs. We use $I$ to indicate that the operator is applicable to indefinite logic programs. The smallest factor of a ground clause $C'$ is defined as the clause $C$ such that $C$ contains only distinct atoms and $C \equiv C'$. Since $C$ in the above definition contains only distinct atoms, it will be in $\text{EHB}(P)$.

**Example 3.** Consider the program

$$P = \{ p(X) \lor q(f(X)) \leftarrow r(X), t(X) \lor q(X), p(b) \lor q(b), r(a) \lor s(a) \}. \tag*{\Box}$$

Consider the state $S_1 = \{ p(b) \lor q(b), r(a) \lor s(a) \}$. Then

$$T_P^I(S_1) = \{ p(b) \lor q(b), r(a) \lor s(a), p(a) \lor q(f(a)) \lor s(a), p(b) \lor t(b) \}.$$

If $S_2 = T_P^I(S_1)$ then

$$T_P^I(S_2) = \{ p(b) \lor q(b), r(a) \lor s(a), p(a) \lor q(f(a)) \lor s(a), p(b) \lor t(b), p(a) \lor t(f(a)) \lor s(a) \}. \tag*{\Box}$$

We next show that the fixpoint operator is continuous and hence monotonic.

**Lemma 3 [8].** Let $X$ be a directed subset of $2^{\text{EHB}(P)}$. Then, for a set of positive ground clauses $\{ A_1, \ldots, A_n \}$, we have

$$\{ A_1, \ldots, A_n \} \subseteq \text{lub}(X) \text{ iff } \{ A_1, \ldots, A_n \} \subseteq I \text{ for some } I \in X. \tag*{\Box}$$

**Theorem 1.** Given a program $P$, then the mapping $T_P^I$ is continuous, and hence monotonic.

**Proof.** $2^{\text{EHB}(P)}$ is a complete lattice under the partial order of set inclusion. Let $X$ be a directed subset of $2^{\text{EHB}(P)}$. 

We have to show that $T'_p(\text{lub}(X)) = \text{lub}(T'_p(M) \mid M \in X)$ (definition of continuous). But

$A \in T'_p(\text{lub}(X))$

iff $C' \leftarrow B_1, B_2, \ldots, B_n$ is a ground instance of a program clause in $P$ s.t.

$A = C' \lor C_1 \lor \cdots \lor C_n$

and $B_1 \lor C_1, \ldots, B_n \lor C_n$ are in $\text{lub}(X)$, where for each $i$, $C_i$ can be null (by definition of $T'_p$)

iff $C' \leftarrow B_1, B_2, \ldots, B_n$ is a ground instance of a program clause in $P$ s.t.

$A = C' \lor C_1 \lor \cdots \lor C_n$

and $B_1 \lor C_1, \ldots, B_n \lor C_n$ are in $S$ for some $S \in X$ (by Lemma 3)

iff $A \in T'_p(S)$ for some $S \in X$ (by definition of $T'_p$)

iff $A \in \text{lub}(T'_p(M) \mid M \in X)$.

Hence $T'_p$ is continuous and monotonic. $\Box$

3.2. Fixpoint Theorems

We next show that when a state is in $\text{SDER}(P)$ of a program $P$ and the state is a fixpoint of $T'_p$, then the state contains all positive clauses which are derivable from the program. This brings us to a point where we have to distinguish between the terms derivability and provability for a disjunctive program and associate them with what we consider as the intended meaning of a logic program. We say a disjunctive program $P$ derives a clause $C$ (written as $P \vdash_D C$) if there is a finite sequence $C_1, C_2, \ldots, C_k$ of clauses such that $C_i$ is either a clause in $P$ or a resolvent of clauses preceding $C_i$, and $C_k = C$. A clause is provable from a program when it is a logical consequence of the program. In the case of Herbrand interpretations the notions of provability and derivability coincide. For the extended Herbrand base this is not valid. With respect to the semantics we are developing, we are only interested in the intended meaning of a program in the sense of derivability. That is, our intended semantics will achieve a state that contains all (and only) the clauses which are derivable from a logic program. Since any provable clause also has a subclause that is derivable, we believe we can restrict our intended meaning of a logic program to derivable clauses without losing generality.

**Theorem 2.** Given a program $P$ and a state $S$ which is in $\text{SDER}(P)$, then $T'_p(S) = S$ iff $S$ is the set of all ground clauses derivable from $P$.

**Proof.** ($\Rightarrow$): $S$ is the set of all ground clauses derivable from $P$. To show $S = T'_p(S)$:

$C \in T'_p(S)$

$\Rightarrow$ there exists a ground instance $C' \leftarrow B_1, B_2, \ldots, B_n$ of a program clause in $P$ s.t.

$C = C' \lor C_1 \lor \cdots \lor C_n$

and $B_1 \lor C_1, \ldots, B_n \lor C_n$ are in $S$, from the definition of $T'_p(S)$
there exists a ground instance \( C' \leftarrow B_1, B_2, \ldots, B_n \) of a program clause in \( P \) s.t.
\[
C = C' \vee C_1 \vee \cdots \vee C_n
\]
and \( B_1 \vee C_1, \ldots, B_n \vee C_n \) are derivable from \( P \), since \( S \) contains all ground clauses derivable from \( P \)
\( \Rightarrow C \) is derivable from \( P \) (from Lemma 2 and linear resolution principles)
\( \Rightarrow C \in S \).
\[ \Rightarrow: T^f(S) = S. \] We have to show \( S \) is the set of all ground clauses derivable from \( P \).
\[ \geq: T^f(S) = S \text{ and } P \text{ derives } C. \] To show \( C \in S \), we use induction on the number of steps required for the shortest derivation of \( C \) from \( P \).

Base case: \( C \) is a ground instance of an assertion clause in \( P \)
\[ \Rightarrow C \text{ is in } T^f(S) \]
\[ \Rightarrow C \in S, \text{ since } S = T^f(S). \]

Induction hypothesis: If a clause is derivable in less than \( k \) steps, then the clause is in \( S \).

Induction step:
\( C \) is derivable in \( k \) steps
\[ \Rightarrow \text{there exists a ground instance of a program clause } C' \leftarrow B_1, B_2, \ldots, B_n \text{ in } P \] s.t.
\[ C = C' \vee C_1 \vee \cdots \vee C_n \]
and \( B_1 \vee C_1, \ldots, B_n \vee C_n \) are derivable from \( P \) in less than \( k \) steps, which takes part in the derivation of \( C \), from Lemma 2
\[ \Rightarrow B_1 \vee C_1, \ldots, B_n \vee C_n \text{ are in } S, \text{ from the induction hypothesis} \]
\[ \Rightarrow C \in T^f(S), \text{ by definition of } T^f(S) \]
\[ \Rightarrow C \in S, \text{ since } T^f(S) = S. \]
\[ \geq: T^f(S) = S \text{ and } C \in S. \] To show \( P \) derives \( C \), \( C \in S \Rightarrow P \text{ derives } C \), since \( S \) is in \( S\text{DER}(P) \).
\[ \square \]

The next theorem is an extension of the above theorem and gives the fixpoint semantics for a disjunctive logic program. It shows that the least fixpoint achieved using the fixpoint operator captures the intended meaning of derivability of the program.

**Theorem 3.** Given a program \( P \),
\[ \text{lfp}(T^f_P) = \{ C | C \text{ is derivable from } P \}. \]

**Proof.** Since \( \text{EHR}(P) \) is continuous, we have
\[ \text{lfp}(T^f_P) = T^f_P \uparrow \omega. \]

We first show by induction on \( n, n \geq 0 \), that \( T^f_P \uparrow n \) contains only clauses derivable from \( P \), i.e., it contains no clauses which are not derivable from \( P \).
A FIXPOINT SEMANTICS FOR DISJUNCTIVE PROGRAMS

Base case: $T_p^0 \uparrow 0 = \emptyset$ (bottom element) contains no clauses which are not derivable from $P$.

Induction hypothesis: $T_p^k \uparrow k - 1$ contains only clauses which are derivable from $P$, $k \geq 1$.

Induction step:

$$T_p^k \uparrow k = T_p^k(T_p^{k-1})$$

$\Rightarrow T_p^k \uparrow k = \{C \mid C' \leftarrow B_1, \ldots, B_m \text{ is a ground instance of a program clause in } P,$

$\text{and } B_1 \lor C_1, \ldots, B_m \lor C_m \text{ are in } T_p^{k-1}, \text{ and } C \text{ is the smallest factor of }$

$C' \lor C_1 \lor C_m, \text{ and } \forall i, 1 \leq i \leq m, C_i \text{ can be null} \}$

$\Rightarrow T_p^k \uparrow k$ contains only clauses derivable from $P$, since $\forall i, 1 \leq i \leq m, B_i \lor C_i$ are derivable from $P$.

Hence, $\text{lfp}(T_p^k) = T_p^k \uparrow \omega$ contains only clauses derivable from $P$.

Using Theorem 2, we have

$$\text{lfp}(T_p^k) = \{C \mid C \text{ is derivable from } P\} \ . \ \square$$

3.3. Equivalence to Model Theory

In this section we provide an equivalence to the model theory defined by Minker [9] for disjunctive programs. Here we consider only Herbrand models and use “Herbrand models” and “models” interchangeably. There is no unique Herbrand model that characterizes a disjunctive program. Instead there is a set of Herbrand models which capture its intended meaning. We give the formal definitions below:

**Definition 5** [9]. Given a program $P$, a Herbrand model $M$ of $P$ is a **minimal Herbrand model** if no proper subset of $M$ is also a model of $P$. The set of minimal models of $P$ is denoted by $\text{MM}(P)$.

Minker [9] has defined the model-theoretic semantics for disjunctive programs based on minimal models:

**Theorem 4** [9]. A positive clause $C$ is a logical consequence of a program $P$ iff $C$ is true in every minimal model of $P$. That is,

$$P \vdash C \iff \forall M \in \text{MM}(P), \ M \models C \ . \ \square$$

Using the above theorem and Theorem 3 in Section 3.2, we have the following result:

**Lemma 4.** Given a program $P$,

$$\forall M \in \text{MM}(P), \ M \models C \iff \text{lfp}(T_p^k) \uparrow C . \ \square$$

An example illustrates this:

**Example 4.** Consider the program $P = \{r(X) \lor s(f(X)) \leftarrow p(sX), p(a) \lor q(g(b))\}$. We have

$$\text{lfp}(T_p^k) = \{p(a) \lor q(g(b)), r(a) \lor s(f(a)) \lor q(g(b))\}$$
and minimal models

\[ \text{MM}(P) = \{ \{ q(g(b)) \}, \{ r(a), p(a) \}, \{ s(f(a)), p(a) \} \}. \]

The set of clauses which are true in the models of \( \text{MM}(P) \) is given by

\[ \{ p(a) \vee q(g(b)), p(a) \vee q(g(b)) \vee s(f(a)), p(a) \vee q(g(b)) \vee r(a), r(a) \vee s(f(a)) \vee q(g(b)) \}. \]

We can see that each clause in the above set is a logical consequence of \( \text{lfp}(T'_p) \).

3.4. Fixpoint Operator and Horn Programs

In this subsection we show the effect of the fixpoint operator \( T'_p \) on Horn programs. First we extend the pre-fixpoint theorem for the operator \( T_p \) given by van Emden and Kowalski [19]:

**Theorem 5** [19]. Let \( P \) be a Horn program and \( I \) be a Herbrand interpretation of \( P \). Then \( I \) is a model for \( P \) if \( T_p(I) \subseteq I \).

**Proof.** [8, Proposition 6.4]. \( \square \)

To generalize the above theorem we define a new state called the \( S \)-model.

**Definition 6.** The \( S \)-interpretation \( \text{st}(P) \) of a program \( P \) is the set of states defined as

\[ \text{st}(P) = \{ S | S \text{ is a state of } P \text{ such that every nonunit clause in } S \text{ has at least one of its atoms in } S \}. \]

The \( S \)-model \( \text{sm}(P) \) of a program \( P \) is the subset of \( \text{st}(P) \) defined as

\[ \text{sm}(P) = \{ S | S \in \text{st}(P) \text{ and all clauses in } P \text{ are logically implied by the atoms in } S \}. \]

**Example 5.**

\( \{ p(a), p(a) \vee p(b) \} \) is an SI.

\( \{ p(c), p(a) \vee p(b) \} \) is not an SI. \( \square \)

In the case of Horn programs a Herbrand interpretation would be in the set \( S \)-interpretation and a Herbrand model would be in the set \( S \)-model.

**Theorem 6.** If \( S \) is in \( \text{sm}(P) \) of a program \( P \), then for every clause in \( T'_p(S) \) there is an atom in the clause which is also in \( S \). Hence \( S \) logically implies \( T'_p(S) \).

**Proof.** Let \( C \in T'_p(S) \). We show that there exists an atom \( A \in S \) s.t. \( A \in C \). \( C \in T'_p(S) \) if \( \text{P} \Rightarrow \) there exists a ground instance of a program clause in \( P \), \( C' \leftarrow B_1, B_2, \ldots, B_n, \) s.t.

\[ C = C' \vee C_1 \vee \cdots \vee C_n \]

and \( B_1 \vee C_1, \ldots, B_n \vee C_n \) are in \( S \) (by definition of \( T'_p \)). Now there are two cases.
Case 1:
\( B_1, \ldots, B_n \) are in \( S \)
\( \Rightarrow \) \( C' \) or a subclause of \( C' \) is in \( S \)
\( \Rightarrow \) an atom \( A \in C' \) is in \( S \)
\( \Rightarrow \) an atom \( A \in C \) is in \( S \).

Case 2:
\( \exists i, 1 \leq i \leq n, \ B_i \notin S \)
\( \Rightarrow \) an atom of \( C_i \) is in \( S \), because \( S \) is in \( \text{sm}(P) \)
\( \Rightarrow \) an atom of \( C \) is in \( S \)
Hence, an atom in \( C \) is in \( S \). \( \square \)

We use \( \text{sm}(P) \) to define criteria for determining if a disjunctive logic program \( P \) has an equivalent Horn program. We show that a program has a Horn equivalent program iff the least fixpoint is in \( \text{sm}(P) \).

**Theorem 7.** Given a disjunctive program \( P \) and \( \text{lfp}(T_P) = S \), then \( S \) is not in \( \text{sm}(P) \) iff \( P \) has no equivalent Horn program.

**Proof.**
\( \Leftarrow \): Suppose \( P \) has no equivalent Horn program. We show that \( S \) is not in \( \text{sm}(P) \):
Assume that \( S \) is in \( \text{sm}(P) \)
\( \Rightarrow \) for all clauses \( C \in S \), there exists an atom \( A \) in \( T_P(S) = S \) (since \( S \) is a fixpoint) where \( A \) is part of \( C \)
\( \Rightarrow \) there exists a state \( S_1 \) s.t. \( S_1 \) contains all the unit clauses in \( S \)
\( \Rightarrow S_1 \) is also a model of \( P \), since \( S \) is in \( \text{sm}(P) \) and a fixpoint of \( T_P \) (also, \( S_1 \) is a least Herbrand model of \( P \), since it contains all the atoms derivable from \( P \))
\( \Rightarrow \) there exists a Horn program \( P_1 \) which also has \( S_1 \) as a least Herbrand model (a Horn program \( P_1 \) can be constructed by removing all the disjunctive clauses from \( P \))
\( \Rightarrow P \) has an equivalent Horn program \( P_1 \), which contradicts the assumption that \( P \) has no equivalent Horn program.
\( \Rightarrow s \) is not in \( \text{sm}(P) \)
\( \Rightarrow \text{:} \): Suppose \( S \) is not in \( \text{sm}(P) \). We show that \( P \) has no equivalent Horn program:
\( S \) is not in \( \text{sm}(P) \) and \( S \) is the lfp of \( P \)
\( \Rightarrow \) there exists a nonunit clause \( C \subseteq S \) s.t. \( P \) derives \( C \) and there is no subclause of \( C \) in \( S \)
\( \Rightarrow P \) has no equivalent Horn program: otherwise an atom in \( C \) could be derived from \( P \). \( \square \)

Shepherdson [17] provides an equivalent result for determining whether a general Horn program is consistent or not. He gives a criterion for defining inconsistency of
the program augmented with the closed-world assumption. He shows that the inconsistency is due to the existence of an indefinite clause which is a logical consequence of the general Horn program such that no atom of the clause is provable from the program. Our result provides a criterion for finding the existence of a Horn program which is equivalent to a disjunctive program.

Next we show that the fixpoint semantics based on $T'$ gives the same result as that of $T$, when operating on Horn programs. When operating on a state $S$ which is a Herbrand interpretation, $T'_p(S)$ is equal to $T_p(S)$ when $P$ is a Horn program. This can be seen from the definitions of $T'_p$ and $T_p$ (Definitions 1 and 4). $T'_p$ is also closed for Horn programs when restricted to Herbrand interpretations, since they are subsets of disjunctive programs, which leads to the following result.

Theorem 8. Given a Horn program $P$, $\text{lfp}(T'_p) = \text{lfp}(T_p)$. □

4. NEGATION

The closed-world assumption (CWA) [15] is a rule that interprets negation as failure to prove. That is, a negative ground predicate, $\neg A$, can be inferred from a Horn program $P$ if $A$ is not provable from $P$. We define $\text{CWA}(P)$ as the set of ground atoms whose negation can be inferred from the CWA rule. The following theorem is a direct result of this definition.

Theorem 9 [8]. Given a Horn program $P$, $\text{CWA}(P) = \text{HB}(P) \setminus T_p \uparrow \omega$. □

We prove a similar result in this section for disjunctive programs. We show that the fixpoint semantics developed in the previous section can be used to define a theory of negation for disjunctive logic programs. We also show that the theory of negation is consistent with the generalized closed-world assumption (GCWA) [9].

4.1. Definitions

We first define the generalized closed-world assumption [9].

Definition 7 [9]. Let $P$ be a disjunctive logic program (function-free) and $C$ a ground atom. Then $\neg C$ can be inferred from $P$ iff $C \notin E$, where

\[ E = \{ A | A \text{ is a ground atom and } P \vdash A \lor K, \]

\[ K \text{ is a positive (possibly null) clause, and } K \text{ is not provable from } P \}. \]

The definition given above is also applicable to disjunctive programs with functions [18]. We define the set of positive atoms which are not provable under the generalized closed-world assumption.
Definition 8. The failure set of a program $P$ under the GCWA, $\text{GCWA}(P)$, is defined as
\[
\text{GCWA}(P) = \{ A | A \in \text{HB}(P) \text{ and } P \not\vdash A \text{ and } \forall K, K \text{ is a positive ground clause, } P \vdash A \lor K \rightarrow P \vdash K \}.
\]

We define a canonical set of positive clauses which are derivable from a disjunctive logic program. We use the canonical set to define the set of ground atoms whose negations can be assumed from the given program.

Definition 9. Given a set of positive ground clauses $S$, the canonical set $\text{can}(S)$ of $S$ is defined as
\[
\text{can}(S) = \{ C | C \in S \text{ and } \exists C' \text{ s.t. } C' \in S \text{ and } C' \text{ is a subclause of } C \}.
\]

Definition 10. The canonical set $\text{cs}(P)$ of positive ground clauses derivable from a disjunctive program $P$ is defined as
\[
\text{cs}(P) = \text{can}(T^I_p \uparrow \omega).
\]

Example 6. Let $P = \{ p(a), p(a) \lor q(b) \}$. Then,
\[
\text{lfp}(T^I_p) = T^I_p \uparrow \omega = \{ p(a), p(a) \lor q(b) \}
\]
and
\[
\text{cs}(P) = \{ p(a) \}.
\]

From the definitions of $\text{cs}(P)$ and $\text{GCWA}(P)$ we can see that if an atom is not in any clause in $\text{cs}(P)$, then it will be in $\text{GCWA}(P)$. The canonical set $\text{cs}(P)$ in the example above does not contain $q(b)$, and $\text{GCWA}(P)$ contains $q(b)$. We give a formal definition of the failure set using the fixpoint operator.

Definition 11. The failure set $\text{fst}(P)$ of a program $P$ under a fixpoint theory is defined as
\[
\text{fst}(P) = \{ A | A \in \text{HB}(P) \text{ and } \exists C \text{ s.t. } C \in \text{cs}(P) \text{ and } A \rightarrow C \}.
\]

4.2. Equivalence of FST and GCWA

Theorem 10. The failure set of a program $P$ under the GCWA is equivalent to its failure set under fixpoint theory. That is,
\[
\text{GCWA}(P) = \text{fst}(P).
\]

Proof. We first show that $\text{GCWA}(P) \subseteq \text{fst}(P)$. We have
\[
A \in \text{GCWA}(P)
\]
\[
\rightarrow P \not\vdash A
\]
\[
\Rightarrow A \not\in T^I_p \uparrow \omega \text{ (from Theorem 3)}
\]
\[
\Rightarrow A \not\in \text{cs}(P) \text{ [from the definition of cs}(P)].
\]
Also

\[ A \in GCWA(P) \]

\[ \Rightarrow \forall K(P \vdash A \lor K \rightarrow P \vdash K) \], where \( K \) is a positive ground clause

\[ \Rightarrow \forall K' \exists K''(P \text{ derives } A \lor K' \rightarrow P \text{ derives } K'') \], where \( K'' \) is a subclause of \( K' \)

\[ \Rightarrow \forall K' \exists K''(K' \in T_P \uparrow \omega \rightarrow K'' \in T_P \uparrow \omega) \], where \( K'' \) is a subclause of \( K' \) (from Theorem 3)

\[ \Rightarrow \forall K' \exists K''(K' \subseteq T_P \uparrow \omega K'' \in CS(P)) \], where \( K'' \) is a subclause of \( K' \)

\[ \Rightarrow \forall K'(A \lor K' \notin CS(P)) \], from the definition of \( CS(P) \), and \( K'' \) is a subclause of \( A \lor K' \).

Hence we have shown that

\[ \forall C, \ C \in CS(P), \ A \text{ is not in } C \Rightarrow \neg \exists C, \ C \in CS(P), \ A \rightarrow C. \]

Hence \( A \in FST(P) \).

We now show that \( FST(P) \subseteq GCWA(P) \).

\[ A \in FST(P) \]

\[ \Rightarrow A \notin CS(P) \]

\[ \Rightarrow A \notin T_P \uparrow \omega \] (since \( A \) is an atom if it is not in \( CS(P) \) it is also not in \( T_P \uparrow \omega \))

\[ \Rightarrow A \) is not derivable from \( P \) (from Theorem 3)

\[ \Rightarrow P \not\vdash A. \]

Also,

\[ A \in FST(P) \]

\[ \Rightarrow \neg \exists C, \ C \in CS(P), \ A \rightarrow C \]

\[ \Rightarrow \forall C, \ C \in CS(P), \ A \text{ is not in } C. \]

Now assume that for some \( K, \ P \vdash A \lor K \), where \( K \) is a positive clause. If \( P \vdash K \) the theorem is proved. Otherwise, \( A \lor K' \) is derivable from \( P \) where \( K' \) is a subclause of \( K \), whence

\[ A \lor K' \in T_P \uparrow \omega \] (from Theorem 3).

But \( A \lor K' \notin CS(P) \). Since \( \forall C, \ C \in CS(P) \), so \( A \) is not in \( C \). Therefore

\[ \exists K'', \ K'' \in T_P \uparrow \omega, \]

where \( K'' \) is a subclause of \( K' \) [from the definition of \( CS(P) \)]. Therefore

\( P \) derives \( K'' \) (from Theorem 3).

Therefore \( P \vdash K \), since \( K'' \) is a subclause of \( K \).

Hence \( A \in GCWA(P) \).

The above theorem can be paraphrased in terms similar to Theorem 9.

**Theorem 11.** Given a disjunctive program \( P \),

\[ GCWA(P) = HB(P) \setminus \{\text{atoms in } \text{can}(T_P \uparrow \omega)\}. \]
5. PROCEDURAL INTERPRETATION FOR DISJUNCTIVE PROGRAMS

In this section we present inference procedures for disjunctive programs. There are several resolution-based procedures which are sound and complete for answering queries from disjunctive programs. Kowalski and Kuehner [7] introduced a modified form of linear resolution for logic programs called SL resolution (Linear resolution with Selection function). The selection function in SL resolution is restricted to certain literals. Minker and Zanon [10] defined a complete and sound modification to SL resolution called LUST resolution (Linear resolution with Unrestricted Selection function based on Trees) for logic programs; it has no restrictive selection function. For Horn programs, Hilf [6] presented a complete and sound linear resolution procedure called LUSH resolution (Linear resolution with Unrestricted Selection function for Horn programs) for answering positive queries. The advantage of this procedure is that one can arbitrarily select literals in a clause on which to expand, and neither ancestry resolution nor factoring is required. Apt and van Emden [3] renamed LUSH resolution SLD resolution (SL resolution for Definite programs). Clark [4] proposed negation as finite failure and developed a query-answering procedure for answering negative queries from Horn programs. This procedure was later renamed SLDNF resolution (SLD resolution with Negation-as-Failure rule).

Since we are dealing with disjunctive programs, ancestry resolution and factoring are necessary. LUST resolution is the basis for our proof procedure for answering queries in disjunctive programs. We use it for the following reasons:

1. LUST resolution is sound and complete for theorem proving [10].
2. It allows arbitrary literal selection.
3. It provides a convenient basis for developing a procedure for answering negative queries.
4. When restricted to Horn clauses, it reduces to SLD resolution.

We rename LUST resolution SLI-resolution (SL resolution for Indefinite clauses). This is consistent with the nomenclature used by Apt and van Emden in renaming LUSH resolution as SLD resolution. SLI resolution is described in the subsection below.

5.1. SLI Resolution

SLI resolution is defined using trees as the basic representation. Each node in the tree is a literal, and there are two types of literals: a marked literal, referred to as an A-literal, and an unmarked literal, called a B-literal. A nonterminal literal is always an A-literal, whereas a terminal literal can be either an A-literal or a B-literal. A t-clause is a special representation of a clause and embeds the information about the ancestry of each literal.

**Definition 12 [10].** A t-clause \((C, m)\) is an ordered pair where

- \(C\) is a labeled tree whose root is labeled with the distinguished symbol epsilon (\(\epsilon\)) and whose other nodes are labeled with literals, and
- \(m\) is a marking relation on the nodes such that every nonterminal node is marked (i.e., is in \(m\)). \(\Box\)
A t-clause can also be viewed as a well-parenthesized expression such that every opening parenthesis is followed by an A-literal. In our discussion we identify an A-literal by marking it with an asterisk (*). A B-literal is unmarked. Input clauses are represented by t-clauses which have only one A-literal, $e^*$. A program consists of a finite set of input clauses. The parenthesized expression of a derived t-clause is a preorder representation of the SLI derivation tree.

Example 7. $(e^*p(X)q(a))$ is a t-clause representation of a $p(X) \lor q(a)$. $(e^*p(X)(q(Y)r(a)(s(b)^*t(c))t(d))r(Z))$ is another example of a t-clause.

An SLI derivation starts with a t-clause called the goal t-clause and successively derives further goal t-clauses by resolving with program clauses. During the derivation, an unmarked literal in the goal t-clause is selected and marked. This literal can be either positive or negative. The selected literal is unified with a complementary literal in a program clause. The resolvent is attached as a subtree to the literal in the goal clause. Factoring, ancestry resolution, and truncation are then performed on the t-clause. The notions of factoring, ancestry resolution, and truncation are similar to those in SL resolution [7].

There are two sets of literals used during resolution. They are defined as follow:

\[
\gamma_L = \{ M: M \text{ is a B-literal, and } M \text{ is a child of a node in the path from the root to the literal } L \}\.
\]

\[
\delta_L = \{ N: N \text{ is an A-literal, and } N \text{ is on the path between the root and the literal } L \}\.
\]

A t-clause is said to satisfy the admissibility condition (AC) if for every occurrence of a B-literal $L$ in the t-clause the following conditions hold:

(i) No two literals from $\gamma_L \cup \{ L \}$ have the same atom.

(ii) No two literals from $\delta_L \cup \{ L \}$ have the same atom.

A t-clause is said to satisfy the minimality condition (MC) if there is no A-literal which is a terminal node.

$\gamma_L$ and $\delta_L$ are used while performing factoring and ancestry respectively. AC and MC make sure that the factoring, ancestry, and truncation are performed as soon as possible.

Now we have the framework for describing an SLI resolution. We next give a formal definition for an SLI derivation. We use $\alpha$ and $\beta$ (with subscripts) to denote sequences of symbols which are parts of a t-clause and are not of current interest. Note that if $(\alpha L \beta)$ is a t-clause then $(\alpha \beta)$ is also a t-clause, where $\alpha$ and $\beta$ may be empty and need not be balanced with respect to parentheses.

Definition 13. An SLI-derivation of a t-clause $E$ from a set of t-clauses, $S$, with top t-clause $C$ is a sequence of t-clauses $D = (C_1, \ldots, C_n)$ such that:

- $C_1$ is $C$, and $C_n$ is $E$;
- $C_{i+1}$ is obtained from $C_i$ by either t-extension, t-factoring, t-ancestry, or t-truncation;
- if $C_{i+1}$ is obtained from $C_i$ by t-extension or t-truncation, then $C_i$ satisfies the admissibility condition.
if \( C_{i+1} \) is obtained from \( C_i \) by \( t \)-extension, \( t \)-ancestry, or \( t \)-factoring, then \( C_i \) satisfies the minimality condition.

\( C_{i+1} \) is obtained from \( C_i \) by \( t \)-extension with input \( t \)-clause \( B_j \) iff

1. \( C_i \) is \((e^* \alpha_1 L \beta_1)\);
2. \( B_j \) is \((e^* \alpha_2 M \beta_2)\);
3. \( L \) and \( M \) are complementary and unify with \( \text{mgu} \ \theta \);
4. \( C_{i+1} \) is \((e^* \alpha_1 \theta (L \theta^* \alpha_2 \theta \beta_2 \theta) \beta_1 \theta)\).

\( C_{i+1} \) is obtained from \( C_i \) by \( t \)-factoring iff

1. \( C_i \) is \((\alpha_1 L \alpha_2 M \alpha_3) \) or \( C_i \) is \((\alpha_1 M \alpha_2 L \alpha_3) \);
2. \( L \) and \( M \) have the same sign and unify with \( \text{mgu} \ \theta \);
3. \( L \) is in \( \gamma_M \) (i.e., \( L \) is in a higher level of the tree);
4. \( C_{i+1} \) is \((\alpha_1 \theta L \theta^* \alpha_2 \theta \alpha_3 \theta) \) when \( C_i \) is \((\alpha_1 L \alpha_2 M \alpha_3) \), or \( C_{i+1} \) is \((\alpha_2 \theta \alpha_3 \theta L \theta^* \alpha_2 \theta \alpha_3 \theta) \) when \( C_i \) is \((\alpha_1 M \alpha_2 L \alpha_3) \).

\( C_{i+1} \) is obtained from \( C_i \) by \( t \)-ancestry iff

1. \( C_i \) is \((\alpha_1 (L^* \alpha_2 (\alpha_3 M \alpha_4) \alpha_5) \alpha_6) \);
2. \( L \) and \( M \) are complementary and unify with \( \text{mgu} \ \theta \);
3. \( C_{i+1} \) is \((\alpha_1 \theta (L \theta^* \alpha_2 \theta \alpha_3 \theta \alpha_4 \theta) \alpha_5 \theta) \).

\( C_{i+1} \) is obtained from \( C_i \) by \( t \)-truncation iff either

\( C_i \) is \((\alpha (L^*) \beta) \) and \( C_{i+1} \) is \((\alpha \beta) \)

or

\( C_i \) is \((e^*) \) and \( C_{i+1} \) is \( \square \).

**Definition 14.** An SLI refutation from the set of \( t \)-clauses \( S \) with top \( t \)-clause \( C \) is an SLI derivation of the null clause \( \square \) from the top \( t \)-clause \( C \). We write \( S \vdash C \) if the null clause is derived by an SLI refutation with \((e^* \neg C)\) as the top \( t \)-clause. SLI resolution is the inference system consisting of \( t \)-extension, \( t \)-factoring, \( t \)-ancestry, and \( t \)-truncation as inference rules.

**Example 8.** Consider the program clauses

1. \((e^* p(f(X)) \rightarrow q(X) \rightarrow r(X))\),
2. \((e^* p(f(X)) q(X))\),
3. \((e^* r(X))\)

and the goal clause

\((e^* \neg p(f(a)))\).
We show that there is an SLI refutation:

\[
\begin{align*}
(e^* \rightarrow p(f(a))) & \quad \text{goal clause} \\
(e^* (\neg p(f(a))^* \rightarrow q(a) \rightarrow r(a))) & \quad t\text{-extension with (1)} \\
(e^* (\neg p(f(a))^* \rightarrow q(a)^* p(f(a)) \rightarrow r(a))) & \quad t\text{-extension with (2)} \\
(e^* (\neg p(f(a))^* \rightarrow q(a)^*) \rightarrow r(a))) & \quad t\text{-ancestry} \\
(e^* (\neg p(f(a))^* \rightarrow r(a))) & \quad t\text{-truncation} \\
(e^* (\neg p(f(a))^* (\neg r(a)^*))) & \quad t\text{-extension with (3)} \\
(e^* \rightarrow p(f(a))^*) & \quad t\text{-truncation} \\
(e^*) & \quad t\text{-truncation} \quad \square
\end{align*}
\]

Minker and Zanon [10] show that SLI resolution is complete and sound for theorem proving with arbitrary clauses.

Theorem 12 [10]. Let \( S \) be a set of input \( t \)-clauses. Then \( S \vdash C \) by SLI refutation iff \( C \) is a logical consequence of \( S \). \( \square \)

In the next section, we use SLI resolution and develop a procedure called SLINF resolution for answering negative queries from a disjunctive program.

5.2. Negation and SLINF resolution

In this section we introduce the concept of support for negation and use it for answering ground unit negative queries in disjunctive logic programs. We also provide a query-answering procedure using support for negation based on SLINF resolution. The procedure is also extended to answer disjunctive and conjunctive queries.

The concept of support for negation stems from the definition of the generalized closed-world assumption.

Definition 15. Given a logic program \( P \) and a ground atom \( A \), the support for negation of \( A \), \( SN(A) \), is defined as

\[
SN(A) = \{ K | K \text{ is a ground positive (possibly null) clause and } P \vdash A \lor K \}.
\]

From the definition for the GCWA, we can see that \( \neg A \) can be assumed if all clauses in \( SN(A) \) are logical consequences of \( P \).

The definition for the support set can be tightened by allowing only those clauses which are derivable from \( P \) instead of all logical consequences. This definition also relates to the least fixpoint of disjunctive logic programs (Section 3.2, Theorem 3).

Definition 16. Given a logic program \( P \) and a ground atom \( A \), the support for negation of \( A \) (using least-fixpoint semantics), is defined as

\[
SNLFP(A) = \{ K | (A \lor K) \in \text{lfp}(T_P) \},
\]

\( K \) is either a positive ground clause or a null clause \( \square \).
The next result follows from the above definitions and Lemma 4.

**Corollary 1.** Given a disjunctive program $P$, a ground atom $A$ in $HB(P)$ is in $GCWA(P)$ if either $SNFLP(A)$ is empty or all clauses in $SLNFP(A)$ are logical consequences of $P$. \(\Box\)

The case where $SNFLP(A)$ is empty is when no clause that contains $A$ can be derived from $P$.

We have given two definitions of support for negation. The one based on the GCWA defines a complete support-for-negation set (SN), which has been reduced using the fixpoint semantics (SNLFP). The negation of an atom $A$ in a program $P$ can be assumed if all the clauses in a nonempty $SNLFP(A)$ are provable from $P$.

The support-for-negation set as defined by $SNLFP(A)$ can be very large (even infinite), and any reduction in the size of this set would be useful for practical implementation. Next, we give a definition for such a reduced set. We define a subset of the set $SNLFP$ and show that this subset is sufficient for inferring negation in a program. We show that if all clauses in the subset are provable from the program, then all clauses in the set $SNLFP$ are also provable from the program. So, to infer the negation of a ground atom using the GCWA, we have to show that this subset (rather than the whole set $SNLFP$) is a logical consequence of the program. The definition of this subset is procedural and uses a modified SLI derivation called SLINF derivation. We use this definition of support for negation to develop a query answering procedure.

We define a $t$-clause to be positive if all its $B$-literals are positive. We define a clause to be the disjunction of all $B$-literals in a $t$-clause. We use the terms $t$-clause and clause interchangeably.

An SLINF derivation is a variation of SLI derivation such that only negative literals in $C_i$, $i = 1, 2, \ldots$ (Definition 13), can take part in the application of a $t$-extension. The rationale for this restriction is that our interest is in deriving positive clauses from the top clause. When we have an SLI derivation with top clause $\neg C$, the clause formed using the $B$-literals at any step of the derivation is a logical consequence of the program and $\neg C$. So restricting the $t$-extension rule from selecting positive literals derives positive clauses wherever possible. (There may be cases where a $t$-clause may have some negative $B$-literal which cannot resolve with any input clause. In such cases we do not reach a refutation in the SLINF derivation.) The modified $t$-extension rule is given below:

$C_{i+1}$ is obtained from $C_i$ by $t$-extension with input $t$-clause $R_j$ iff

1. $C_i$ is $(\epsilon^* \alpha_1 \neg L \beta_1)$, where $L$ is an atom,
2. $B_j$ is $(\epsilon^* \alpha_2 M \beta_2)$,
3. $\neg L$ and $M$ are complementary and unify with mgu $\theta$,
4. $C_{i+1}$ is $(\epsilon^* \alpha_1 \theta (\neg L \theta^* \alpha_2 \theta \beta_2 \theta \beta_i \theta))$.

**Definition 17.** An SLINF refutation with top ground $t$-clause $(\epsilon^* \neg C)$ is an SLINF derivation which ends in a $t$-clause $(\epsilon^* K)$ whose $B$-literals are all positive or which has no $B$-literals (i.e., $K$ is the empty clause). No other SLINF derivation is an SLINF refutation.

**SLINF resolution** is the inference system for finding an SLINF refutation. \(\Box\)
We use SLINF refutation to define a support-for-negation set for an atom.

\textbf{Definition 18.} Given a logic program $P$ and a ground atom $A$, the \textit{support for negation} of $A$ (using SLINF) is defined as

\[\text{SNSLINF}(A) = \{ K | K \text{ is derived by an SLINF refutation from } P \text{ with top clause } (e^* \rightarrow A) \}. \]

The set of clauses defined by $\text{SNSLINF}(A)$ is contained in or equal to that of $\text{SNLFP}(A)$. That is, if a clause is derivable using an SLINF derivation, it is also derivable using an SLI derivation. We show that these two sets are logically equivalent under the program $P$. We do this by showing that the clauses in $\text{SNLFP}(A)$ are logical consequences of $\text{SNSLINF}(A) \cup P$ and that $\text{SNSLINF}(A)$ is a subset of $\text{SNLFP}(P)$. The advantage, as pointed out earlier, is that the reduced size of the support-for-negation set makes it easier to compute negation. Note that this reduced set need not be the optimal support-for-negation set, since two clauses $K$ and $K'$ can be in the set $\text{SNSLINF}(A)$. $K \vee K'$ is a redundant clause in the set, since the provability of $K$ from the program also implies the provability of $K \vee K'$. The notation $X \Rightarrow Y$, used in the following theorem, implies that all clauses in the set $X$ can be proved from the set $Y$ using the inference system $Y$.

\textbf{Theorem 13.} Given a program $P$ and a ground atom $A$, \[P \cup \text{SNSLINF}(A) \Rightarrow \text{SLI} \text{SNLFP}(A).\]

\textbf{PROOF.} Let $C$ be a ground positive clause in $\text{SNLFP}(A)$. If $C$ is also in $\text{SNSLINF}(A)$, the theorem is proved.

Otherwise [note that $C$ cannot be in $P$, since $C$ is in $\text{SNLFP}(A)$], there is an SLI derivation of $C$ from $P$ using the top clause $(e^* \neg A)$. Since $C \in \text{SNLFP}(A)$, we have $P \vdash A \vee C$. But there is no SLINF derivation of $C$ from $P$, by assumption.

Having an SLI derivation and no SLINF derivation implies that there is at least one positive literal in the SLI derivation which inhibits the SLINF derivation from deriving $C$, since SLINF derivation cannot expand positive literals. The diagram in Figure 1 illustrates the case.

Assume $C_0, C_1, \ldots, C_n, C_{n+1}$ to be the positive clauses formed from $\alpha_0, \alpha_1, \ldots, \alpha_n, \alpha_{n+1}$ respectively in the diagram. Then $C_0 \vee C_1 \vee \cdots \vee C_n \vee C_{n+1}$ is derived using an SLI derivation from $P \cup \{\neg A\}$, and $C_0 \vee L_1 \vee \cdots \vee L_n \vee C_{n+1}$ is derived using an SLINF derivation from $P \cup \{\neg A\}$.

Now, an SLI derivation is sound for theorem proving. Hence, we have $P \cup (C_0 \vee L_1 \vee \cdots \vee L_n \vee C_{n+1}) \vdash C_0 \vee C_1 \vee \cdots \vee C_n \vee C_{n+1}$. So for every clause $C$ in $\text{SNLFP}(A)$ there exists a clause $C'$ in $\text{SNSLINF}(A)$ such that $C$ is derivable from $P$ and $C'$. Since SLI is a sound and complete inference system, we have the result

\[P \cup \text{SNSLINF}(A) \Rightarrow \text{SLI} \text{SNLFP}(A).\]

\textbf{Theorem 14.} Given a program $P$ and a ground atom $A$, \[\text{SNSLINF}(A) \subseteq \text{SNLFP}(A).\]
PROOF.

(1) From the definition of SNLFP(A) and since SLI is a sound and complete inference system, it follows that

\[ \forall K, A \lor K \in \text{SNLFP}(A) \]

implies that \( K \) is derivable from \( P \) using an SLI derivation with top clause \((e^* \cdash A)\).

(2) From the definitions of SLI and SLINF derivations, it is obvious that if a positive clause \( K \) is derivable using an SLINF derivation from \( P \) with top clause \((e^* \cdash A)\), then \( K \) is also derivable using an SLI derivation from \( P \) with top clause \((e^* \cdash A)\).

Hence, from (1) and (2), we have \( \text{SNSLINF}(A) \subseteq \text{SNLFP}(A) \).

Example 9. Let \( P = \{ p(X) \lor q(X), r(X) \leftarrow q(X), s(X) \lor t(X) \leftarrow q(X) \} \). Then

\[ \text{SNLFP}(p(a)) = \{ q(a), r(a), s(a) \lor t(a) \} \]

\[ \text{SNSLINF}(p(a)) = \{ q(a) \} \].

We can see that \( \text{SNSLINF}(p(a)) \subseteq \text{SNLFP}(p(a)) \), and if \( P \vdash q(a) \) then \( P \vdash r(a) \land s(a) \lor t(a) \). That is, \( \text{SNSLINF}(p(a)) \cup P \vdash \text{SNLFP}(p(a)) \).

5.3. Query-Answering Procedure

In this subsection we provide a procedure which can be used for answering queries. For negative query \( \neg A \) the procedure derives a positive \( B \)-clause \( K \) from the top
t-clause \((e^* \rightarrow A)\) using an SLINF derivation. If \(K\) is provable from the program, then the procedure tries another SLINF refutation and a new \(K\). If at any time a \(K\) is not provable (finitely), then the procedure fails the query. If there are no more SLINF refutations, the query succeeds.

**Procedure 1 (Query-Answering Procedure QAP).**

**Procedure:** (Unit Query: \(L\), Program: \(P\))

**Positive Query:** \(L\) is atomic and \(L = A\)
- If there is an SLI refutation from \(P\) and top clause \((e^* \rightarrow A)\)
  - then SUCCEED \(L\)
  - else FAIL \(L\)

**Negative Query:** \(L\) is ground and \(L = \neg A\)
1. Construct an SLINF refutation from \(P\) and top clause \((e^* \rightarrow A)\) which is distinct from the previous SLINF refutations.
   - Let \(K\) be the positive B-clause derived
   - If \(P \vdash_{SLI} K\) then step 2
   - else FAIL \(L\)
2. If no more distinct SLINF refutations can be constructed
   - then SUCCEED \(L\)
   - else step 1.

**Example 10.** Let \(P = \{p \lor q \leftarrow m, r \lor m, s \lor m, q \lor r\}\), and let the query be \(\neg p\). To find if \(\neg p\) can be assumed, we construct SLINF refutations for \(p\). For this we start with \((e^* \rightarrow p)\) as top t-clause and perform SLINF derivations:

1. Step 1 in QAP constructs an SLINF refutation and ends in a B-clause \(q \lor r\).
2. Since \(P \vdash_{SLI} q \lor r\), we go to step 2 in QAP.
3. Since another distinct SLI refutation is possible, we go to Step 1 of QAP.
4. Step 1 in QAP constructs an SLINF refutation which is distinct from the one in step 1 of this example and ends in a B-clause \(q \lor s\).
5. Since \(P \not\vdash_{SLI} q \lor s\), the query FAIL and we cannot conclude \(\neg p\). □

The procedure QAP is sound with respect to the generalized closed-world assumption but is not complete for two reasons: First, for the same reasons as SLDNF resolution is not complete with respect to the closed-world assumption, i.e., an infinite SLINF-derivation tree may result. Second, an infinite number of SLINF refutations may be present.

The above procedure can be extended to answer nonunit negative queries using the following lemmas. The semantic definition of the GCWA [9,21] allows one to infer a clause from a program \(P\) if and only if the clause is true in every minimal model of \(P\). For a negative disjunctive query \(\neg p \lor \neg q\) to be inferred we have to find the support for negation for \(\neg (\neg p \lor \neg q)\), that is, the support for negation for \(p \land q\). Lemma 6 can be used to find it.

**Lemma 5 (Support for negation for disjunctions).** If \(K_1\) is in SNSLINF\((A_1)\) and \(K_2\) is in SNSLINF\((A_2)\), then \(K_1 \land K_2\) is in SNSLINF\((A_1 \lor A_2)\). □
Lemma 6 (Support for negation for conjunctions). If $K_1$ is in $\text{SNLS}_{\text{INF}}(A_1)$ and $K_2$ is in $\text{SNLS}_{\text{INF}}(A_2)$, then $K_1 \lor K_2$ is in $\text{SNLS}_{\text{INF}}(A_1 \land A_2)$. □

We illustrate Lemma 6 with a simple example.

Example 11. Let $P = \{ p(a) \lor q(a), p(a) \lor r(a) \}$. Consider the disjunctive query $\neg p(a) \lor \neg q(a)$. The negation of the query yields $p(a) \land q(a)$.

First, we require $\text{SNLS}_{\text{INF}}(p(a) \land q(a))$:

$\text{SNLS}_{\text{INF}}(p(a)) = \{ q(a), r(a) \}$,
$\text{SNLS}_{\text{INF}}(q(a)) = \{ p(a) \}$.

From Lemma 6, we have $\text{SNLS}_{\text{INF}}(p(a) \land q(a)) = \{ (p(a) \lor q(a), p(a) \lor r(a)) \}$. Since all clauses in $\text{SNLS}_{\text{INF}}(p(a) \land q(a))$ are provable from $P$ using SLI refutations, we can assume that the query $\neg p(a) \lor \neg q(a)$ is true.

The set of minimal Herbrand models for $P$ is $\{ \{ p(a) \}, \{ q(a), r(a) \} \}$. We can see that $\neg p(a) \lor \neg q(a)$ is true in both the minimal models of $P$. □

5.4. SLI, SLINF, and Horn Programs

Next we relate SLI and SLINF procedures to Horn programs. We show that SLI reduces to SLD and QAP to SLDNF when dealing with Horn programs.

We first give the definitions for SLD resolution for Horn programs:

Definition 19. A goal is of the form

$$\leftarrow A_1, \ldots, A_n, \quad n \geq 0,$$

where the $A$'s are atoms. □

Definition 20. Let $P$ be a Horn logic program, and $G$ be a goal. An SLD derivation from $P$ with top goal $G$ consists of a (finite or infinite) sequence of goals $G_0 = G, G_1, \ldots$ such that for all $i \geq 0$, $G_{i+1}$ is obtained from $G_i$ as follows:

1. $A_m$ is a clause in $G_i$. $A_m$ is called the selected clause.
2. $A \leftarrow B_1, \ldots, B_q$ is a program clause in $P$.
3. $A_m \theta = A \theta$, where $\theta$ is a substitution (most general).
4. $G_{i+1}$ is the goal $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k) \theta$. □

Definition 21. An SLD refutation from $P$ with top goal $G$ is a finite SLD derivation of the null clause □ from $P$ with top goal $G$. If $G_n = \square$, we say the SLD refutation has length $n$. □

Now we compare SLI and SLD derivation procedures. When using SLI resolution with Horn programs, all literals in the derivation tree (both $A$-literals and $B$-literals) are positive. This implies that ancestry resolution (i.e. the $t$-ancestry rule) is not used in the derivation. A $t$-extension step followed by any required $t$-truncation steps is equivalent to an SLD-derivation step. The advantage of SLI derivation over SLD derivation is that it does $t$-reduction, which reduces expansion redundan-
ties. So performing an SLI derivation with Horn programs is equivalent to performing an SLD resolution such that each t-expansion step can be synchronized to an SLD-derivation step. Hence, an SLI derivation (modulo t-reduction) reduces to an SLD derivation when the program is Horn. From the definition of SLINF derivation we see that an SLINF derivation also reduces to an SLD derivation when used on Horn programs.

Example 12. Consider the program \( P = \{ p(X, Y) \leftarrow q(X), r(Y), r(X) \leftarrow s(X), q(a), s(f(b)) \} \), and let the query be \( p(X, Y) \). The two derivation procedures are shown below:

\[
\begin{align*}
\text{SLI derivation} & \quad \text{SLD derivation} \\
(e^* \leftarrow p(X, Y)) & \quad \leftarrow p(X, Y) \\
(e^* (\neg p(X, Y)^* \neg q(X) \neg r(Y))) & \quad \leftarrow q(X), r(Y) \\
(e^* (\neg p(a, Y)^* \neg q(a)^* \neg r(Y))) & \quad \leftarrow r(Y) \\
(e^* (\neg p(a, Y)^* \neg r(Y))) & \quad \leftarrow s(Y) \\
(e^* (\neg p(a, f(b))^* \neg r(f(b)^* \neg s(f(b))))) & \quad \leftarrow \\
(e^* (\neg p(a, f(b))^* \neg r(f(b)^*)) & \quad \leftarrow \\
(e^* \leftarrow p(a, f(b))^*) & \quad \leftarrow \\
\end{align*}
\]

Next we show how an SLDNF procedure relates to the query-answering procedure QAP. The SLDNF procedure is an extension of SLD derivation with negation-as-failure rule. We can look at the query-answering procedure QAP (Procedure 1) given in the previous subsection as a negation-as-failure rule when dealing with Horn programs. We show how QAP is equivalent to the negation-as-failure rule. The negation-as-failure rule can be stated as follows:

**Definition 22.** For answering a query \( \neg A \) from a Horn program \( P \), follow the following procedure:

- if an SLD derivation of \( A \) finitely fails then return SUCCESS
- else if an SLD derivation of \( A \) derives a null clause return FAIL.

Step 1 of QAP returns FAIL when there is a null clause derived for a negative query. Derivation of a null clause is equivalent to an SLI refutation (and hence an SLD refutation). That is, \( A \) is a logical consequence of \( P \). Step 2 returns SUCCESS when no positive B-clauses are derivable. That is, there is no SLINF refutation. That is equivalent to saying that there is no SLD refutation for \( A \) and hence \( A \) is not a logical consequence of \( P \). So QAP returns FAIL when \( A \) is a logical consequence of \( P \), and returns SUCCESS when \( A \) is not a logical consequence of \( P \). Hence the QAP and negation as failure (NAF) are equivalent.

Note that we are dealing with finite SLD derivations. SLINF derivations are slightly more powerful than SLD derivations when operating on Horn programs.
This is because of the loop-checking capability provided by the admissibility condition of SLI derivations. That is, given a rule \( p(X) \leftarrow p(X) \) and a goal \( \leftarrow p(a) \), the only SLD derivation possible would be infinite, whereas the corresponding SLI derivation would fail.

6. OTHER APPROACHES

6.1. Fixpoint Semantics

Recently, in an attempt to define a semantics for disjunctive logic programs, the concept of a stratified logic program has been developed [2,20,13,11]. A stratified program based primarily on considering disjunctive programs as Horn programs with negative literals in the body (general Horn programs). Such programs are categorized as stratified programs based on certain rules which inhibit recursion through negative literals.

Apt, Blair, and Walker [2] have developed a fixpoint semantics based on van Emden and Kowalski's closure operator \( T_p \) (see Definition 1), and show that the operator reaches a fixpoint. The fixpoint semantics of [2] differs from our approach in two ways. The first is that the fixpoint reached corresponds to one of the minimal models. That is, one of the minimal models is preferred over the other minimal models. This implies that the intended meaning captured by the semantics is not strictly logical consequences (which correspond to all the minimal models), but slightly more than that. That is, the success set is different in our approach. The second difference is that the theory of negation corresponds to the closed-world assumption (based on a preferred minimal model called the standard model) instead of the generalized closed-world assumption. This again implies that the failure set is larger than in our case.

6.2. Negation

Below, we discuss related work by Henschen and Park [5] and Przymusinski [12], who have developed procedures for answering negative queries from disjunctive logic programs. We also discuss work by Ross and Topor [16], who recently developed a semantics for negation for disjunctive programs based on a closure operator.

The Henschen-Park procedure answers a negative unit query \( \neg Q \) by showing that the set of minimal positive indefinite ground clauses (PIGC) containing an instance of \( Q \) is empty. This is possible because if there are any such minimal clauses (minimal in the sense that no other positive clause derivable from the theory subsumes any of the clauses in PIGC), then \( Q \) becomes indefinite. The procedure is complicated but provides several effective strategies.

Przymusinski has developed a procedure that is similar to the one described in this paper. His procedure uses a restricted form of OL resolution called MILO resolution to find the existence of a minimal model satisfying a given formula \( Q \). If no such minimal model exists, then the negation of \( Q \) can be assumed by the
generalized closed-world assumption. The procedure stems from the semantic definition of the GCWA.

The procedure described in this paper takes the syntactic definition of the GCWA, generates a set of positive clauses called support for negation, and uses it to answer negation. Our procedure differs from the Przymusinski procedure in two ways. First, it allows more freedom in the selection of literals during resolution. Second, it is based on a theory which is an extension of the negation theory for Horn programs. We have shown that our procedure extends the negation-as-failure rule for Horn programs to the disjunctive domain, using the support-for-negation concept. The set $\Pi_{GC[Q]}$ defined in the Henschen-Park procedure is similar to the support-for-negation set defined in our procedure. The difference is in the way the support for negation is generated and used for answering negation. Our main focus has been to develop a theory of support for negation on the basis of the GCWA and use it to develop a procedure for answering negative queries in disjunctive programs. The GCWA is used to define a complete support-for-negation set ($SN$), which is reduced using the fixpoint semantics ($SNLF$). The size of the set is reduced further in the procedural counterpart for the support for negation ($SNLINF$).

Ross and Topor [16] define a semantics for negation for disjunctive programs based on a new rule called the disjunctive database rule (DDR). They define the DDR for stratified disjunctive logic programs and show that it reaches a fixpoint. Rajasekar, Lobo, and Minker [14] provide a negation rule for disjunctive programs called the weak generalized closed-world assumption (WGCWA), which is equivalent to the DDR, and develop a query-answering procedure for disjunctive programs using the WGCWA. The main difference between the DDR (or the WGCWA) and the GCWA is that the DDR and the WGCWA deal with the head of a disjunctive clauses as an inclusive OR, whereas the GCWA deals with it as an exclusive OR. This leads to a stricter interpretation of negation by the GCWA than by the DDR. They define a procedure called PL resolution for answering queries from disjunctive programs. PL resolution is used to infer negation under a rule called negation as positive failure, which is sound with respect to the DDR but not complete. The negation-as-positive-failure rule states that a negative literal $\neg A$ can be inferred when all PL derivations from $A$ fail finitely.

7. CONCLUSION

We have presented a fixpoint semantics for disjunctive logic programs. The fixpoint operator defined is shown to be monotonic and hence achieves a least fixpoint. We have shown that the least fixpoint reached captures the intended meaning of derivability of disjunctive logic programs. We have shown that for Horn programs the theory behaves exactly like the fixpoint theory defined by van Emden and Kowalski. We have defined negation based on this theory, and we have shown that it is equivalent to the generalized closed-world assumption. The equivalence of the fixpoint semantics with the minimal model semantics has been shown. We have developed a proof procedure to handle negation, based on the concept of support for negation, and described a proof procedure based on SLINF derivation. The proof procedure for SLINF resolution is based on SLI resolution, which reduces to
SLD resolution for a Horn program. The SLINF inference system for handling
negation reduces to the SLDNF inference system for Horn programs.

We wish to express our appreciation to the National Science Foundation for support of our work under
grant IRI-86-09170, and the Army Research Office for support under grant number DAAG-2985-K-0-177. We wish to thank Jorge Lobo for his insightful suggestions and useful discussions during the
development of the paper. We also wish to thank three anonymous referees for their helpful suggestions.

REFERENCES

1. Apt, K. J., Introduction to Logic Programming, in: J. van Leeuwen (ed.), Handbook of
J. Minker (ed.), Foundations of Deductive Databases and Logic Programming, Morgan
Minker (ed.), Foundations of Deductive Databases and Logic Programming, Morgan
9. Minker, J., On indefinite databases and the closed world assumption, in: Lecture Notes in
10. Minker, J., and Zanon, G., An Extension to Linear Resolution with Selection Function,
of the Workshop on Foundations of Deductive Databases and Logic Programming,
12. Przymusinski, T., Query Answering in Circumscriptive and Closed World Theories, in:
186–190.
13. Przymusinski, T. C., On the Semantics of Stratified Deductive Databases and Logic
Programs, J. Logic Programming, to appear.
Technical Report CS-TR-1994, Computer Science Dept., Univ. of Maryland, College
15. Reiter, R., On Closed World Data Bases, in: H. Gallaire and J. Minker (eds.), Logic and
17. Shepherdson, J. C., Negation as Finite Failure: A Comparison of Clark's Completed
Database and Reiter's Closed World Assumption, J. Logic Programming 1(15):51–79
(June 1984).

