Disjunctive Logic Program = Horn Program + Control Program

Wenjin Lü and Ulrich Furbach
Department of Computer Science, University of Koblenz-Landau
Rheinau 1, D-56075 Koblenz, Germany
{lue,uli}@uni-koblenz.de

Abstract

This paper presents an alternative view on disjunctive logic program: Disjunctive program = Control program + Horn program. For this we introduce a program transformation which transforms a disjunctive logic program into a Horn program and a so called control program. The control program consists of only disjunctions of new propositional atoms and controls the "execution" of the Horn program. The relationship between original and transformed programs is established by using circumscription. Based on this relationship a new minimal model reasoning approach is developed. Due to the transformation it is straightforward to incorporate SLD-resolution into the proof procedure.

1 Introduction

Disjunctive logic programs may contain definite or indefinite information which reflects human limitation in understanding the world being modelled. In the absence of negation, that is, for positive disjunctive program, the semantics is defined by so called Generalized Closed World Assumption (GCWA) which is equivalent to minimal model reasoning [Min82]. GCWA allows one to assume an atom to be false if it does not appear in any minimal model of the program.

There are various proof procedures proposed for disjunctive logic programming: Minker and his co-workers used SLI-resolution [LMR92], which is in fact a version of the model elimination procedure; Loveland proposed the near-Horn-Prolog procedures [Lov91] and there are variants of model elimination to deal with disjunctive logic programs[BFS97]. Besides this goal oriented approaches there are as well bottom-up procedures, like SATCHMO[MB88, LRW95] or Hyper tableau[BFN96]. For a unifying view of bottom up and goal oriented approaches see [BF97].

In this paper we propose a novel point of view on disjunctive logic program which is motivated by the following observation.
Given a disjunctive logic program $\mathcal{P}$, for each clause C of the form

$$a_1 \vee ... \vee a_n \leftarrow b_1, ..., b_m$$

we split C into a set $Horn(C)$ of Horn clauses,

$$\begin{aligned} Horn(C) \quad = \quad & \{a_1 \leftarrow b_1, ..., b_m, \\ & a_2 \leftarrow b_1, ..., b_m, \\ & ...... \\ & a_n \leftarrow b_1, ..., b_m \ \} \end{aligned} \tag{1}$$

Considering the following two sets:

$$Horn(\mathcal{P}) = \prod_{C\in\mathcal{P}} Horn(C) \tag{2}$$

$$\bar{\mathcal{P}}_H = \bigcup_{C\in\mathcal{P}} Horn(C) \tag{3}$$

$Horn(\mathcal{P})$ can be regard as a set of Horn programs, each Horn program in it contains exactly one Horn clause from each $Horn(C)$ ($C \in \mathcal{P}$). It is know that this splitting is complete wrt. minimal models of $\mathcal{P}$ in the sense that for each minimal model M of $\mathcal{P}$, there is a Horn program $\mathcal{P}_M \in Horn(\mathcal{P})$ such that M is the least model of $\mathcal{P}_M$ [IKH92, Lu97]. On the other hand, $\bar{\mathcal{P}}_H$ is a Horn program. Notice that for any $\mathcal{P}_h \in Horn(\mathcal{P})$ $\mathcal{P}_h \subseteq \bar{\mathcal{P}}_H$[1], therefore we can say $\bar{\mathcal{P}}_H$ wrt. minimal models of $\mathcal{P}$ is complete in the sense that for each minimal model M of $\mathcal{P}$, there is a subset $\mathcal{P}_M$ of $\bar{\mathcal{P}}_H$ such that M is the least model of $\mathcal{P}_M$. So under minimal model semantics a disjunctive logic program can be regarded as a Horn program together with a control means which controls the execution of the Horn program (which clauses should be chosen to make a minimal model).

Based on this observation, in this paper we put forward

Disjunctive Program=Control Program + Horn Program

This point of view establishes a relationship between disjunctive logic programming and definite logic programs, the latter has a well-understood declarative semantics and an effective procedural implementation. As a consequence, we can expect to implement minimal model reasoning by using existing techniques (eg. SLD-resolution) developed for definite logic programs.

Other contributions of this paper are as follows:

- A program transformation is introduced which demonstrates the idea presented above. By introducing some new propositional atoms it transforms a disjunctive logic program $\mathcal{P}$ into $\mathcal{P}_C \cup \mathcal{P}_H$, where $\mathcal{P}_C$ is a set of disjunctive facts consisting of only new propositional atoms and $\mathcal{P}_H$ is a Horn program. It turns out that this transformation is sound and complete wrt. minimal models in the sense that M is a minimal model of $\mathcal{P}$ iff there is a minimal model M_C of $\mathcal{P}_C$ such that $M \cup M_C$ is (P, Z)-minimal model of $\mathcal{P}_C \cup \mathcal{P}_H$, where P are the set of all atoms in $\mathcal{P}$ and Z is the set of new propositional atoms introduced by the transformation.
- Based on above result, a novel minimal model reasoning procedure is developed which differs from similar work[LMR92, Gin89, Prz89, Ara96] in that it uses SLD-resolution as a basic inference mechanism.

The rest of the paper is organised as follows: after reviewing logic programs and related background knowledge, we provide the transformation in Section 3. Some properties of the transformation are discussed and the soundness and completeness are proved. In Section 4, we discuss how minimal model reasoning can be implemented based on SLD-resolution and develop such a procedure. The paper is finally concluded with some remarks on further work.

2 Preliminaries

Given a first order language L, a *disjunctive logic program* $\mathcal{P}$ is a finite set of *disjunctive program clauses* of the form

$$a_1 \vee \ldots \vee a_n \leftarrow b_1 \wedge \ldots \wedge b_m$$

[1] Here for convenience, we abuse an element in $\prod_{C\in\mathcal{P}} Horn(C)$ as a set

where every a_i and b_j are atoms with $n \geq 1$ and $m \geq 0$, and all variables are considered to be universally quantified. $a_1 \vee \ldots \vee a_n$ is referred to as the *head* and $b_1 \wedge \ldots \wedge b_m$ as the *body* of the program clause. Usually non-monotonic negations are allowed in the body of a clause, but in this paper we restrict ourselves to positive programs where such negations do not occur. This is not a severe restriction, since minimal model reasoning with positive programs is an important problem to tackle [NNS95, Oli92, IKH92, Gin89, Prz89, Ara96]. When $n = 1$, the program clause is called as a *Horn or definite program clause*, and when $m = 0$, it is called as a *fact*. Note that head of a program clause is not allowed to be empty, and consequently all disjunctive logic programs are consistent. A formula in disjunctive normal form, consisting only of ground atoms, is referred to as a *sentence*. The reader is referred to, for example, [Llo87, LMR92] and references therein, for more information on logic programming and disjunctive logic programs.

The Herbrand base of the language L is usually denoted as HB_L. When no ambiguity arises the subscript L is dropped. Usually, when we consider a program $\mathcal{P}$, we are interested in a Herbrand Base that is restricted to predicate, function and constant symbols that appear in $\mathcal{P}$. If $\mathcal{P}$ has no constant symbols then a dummy constant is assumed. In the sequel we call such Herbrand Base the Herbrand base of $\mathcal{P}$, denote it as $HB_{\mathcal{P}}$ and simply write interpretation and model to denote Herbrand interpretation and Herbrand model resp. of a logic program.

The meaning of a disjunctive logic program is given by the set of its logical consequences. Obviously, negative information can not be handled efficiently in this classical semantics, and hence *generalised closed world assumption* [Min82], referred to as GCWA for short, is usually employed to infer negative information from a disjunctive logic program (see [LMR92] for more discussion and details). GCWA allows one to assume an atom to be false, if it doesn't appear in any minimal model of the program. This has been independently extended for sentences in [YH85, GPP89]. All these versions are not fundamentally different, and for the purpose of this paper, we simply refer to the following definition of closed world assumption in disjunctive logic programming.

In the following we make use of the notion of *minimal models*, *mimimal entailment*, written as $\mathcal{P} \models_{min} \neg\alpha$, and the *immediate consequence operator* $T_{\mathcal{P}}$ for Horn programs.

3 CH-Transformation

In this section we first introduce a program transformation which realizes the idea presented in the introduction. Then we discuss the relationship between original and transformed programs and prove that under minimal model semantics they are equivalent. This result provides a basis for our further discussion.

Definition 3.1 *Let $\mathcal{P}$ be a disjunctive logic program. For each clause $C \in \mathcal{P}$ of form:*

$$a_1 \vee a_2 \vee ... \vee a_n \leftarrow b_1, ..., b_m$$

The CH-transformation of C, denoted by $CH(C)$, is a set of clauses defined by:

$$CH(C) = \begin{cases} \{C\}: & n = 1 \\ C_{ch}: & n > 1 \end{cases}$$

where

$$\begin{aligned} C_{ch} \quad = \{ \quad & C_1 \vee C_2 \vee ... \vee C_n, \\ & a_1 \leftarrow b_1, ..., b_m, C_1, \\ & a_2 \leftarrow b_1, ..., b_m, C_2, \\ & ...... \\ & a_n \leftarrow b_1, ..., b_m, C_n \ \} \end{aligned}$$

C_1, ..., C_n are new propositional atoms not in $\mathcal{P}$, they are called control variables, the disjunctive fact $C_1 \vee C_2 \vee ... \vee C_n$ is called a control clause. The CH-transformation of $\mathcal{P}$, denoted by $CH(\mathcal{P})$, is defined by

$$CH(\mathcal{P}) = \bigcup_{C \in \mathcal{P}} CH(C)$$

The set of all control variables in $CH(\mathcal{P})$ is denoted by $C_\mathcal{P}$

By the definition, the clause in $CH(\mathcal{P})$ is either a Horn clause or a disjunctive fact (control clause). Therefore $CH(\mathcal{P})$ can be represented as follows:

$$CH(\mathcal{P}) = \mathcal{P}_C + \mathcal{P}_H$$

where $\mathcal{P}_C$ is the program consisting of all control clauses in $CH(\mathcal{P})$ and $\mathcal{P}_H$ is the Horn program consisting of all Horn clauses in $CH(\mathcal{P})$. $\mathcal{P}_C$ is called the control program of $\mathcal{P}_H$. The following facts are trivial.

- The disjunctions in $\mathcal{P}_C$ consists of only control variables. No control variable occurs more than once in $\mathcal{P}_C$.
- Each Horn clause in $\mathcal{P}_H$ contains at most one control variable. No control variable occurs two times in $\mathcal{P}_H$.
- Each control variable occurs exactly two times in $CH(\mathcal{P})$, once in $\mathcal{P}_C$ and once in $\mathcal{P}_H$.

For any clause C of the form

$$a_1 \vee a_2 \vee ... \vee a_n \leftarrow b_1, ..., b_m$$

let $Horn(\mathcal{C})$, $Horn(\mathcal{P})$ and $\bar{\mathcal{P}}_H$ be given by formulae 1, 2 and 3 in the introduction resp. For any clause $C \in \mathcal{P}_H$, we denote $\bar{C}$ the clause obtained from C by deleting the control variable in the body of C (if any). Below we state the relationship between $Horn(\mathcal{P})$ and $CH(\mathcal{P})$.

Definition 3.2 *Let $\mathcal{P}$ be a disjunctive logic program and $C_\mathcal{P}$ be the set of all control variables. For any $S \subseteq C_\mathcal{P}$, the subprogram of $\mathcal{P}_H$ determined by S, written as $\mathcal{P}_H(S)$, is defined by*

$$\mathcal{P}_H(S) = \{\bar{C} \mid \ C \in CH(P) \textit{ such that } C \textit{ contains either no control variable or one in } S\}$$

Notice the fact that when without considering the control variables, $\bar{\mathcal{P}}_H$ and $\mathcal{P}_H$ are same and for a minimal model M_C of $\mathcal{P}_C$, it contains one and only one control variable from each control clause, then the following proposition is trivial.

Proposition 3.3 *Let $\mathcal{P}$ be a disjunctive logic program. Then for any $\mathcal{P}_h \in Horn(\mathcal{P})$, there is a minimal model M_C of $\mathcal{P}_C$ such that $\mathcal{P}_H(S) = \mathcal{P}_h$*

Because $Horn(\mathcal{P})$ is complete wrt. minimal model in the sense that for any minimal model M of $\mathcal{P}$, there is a $\mathcal{P}_h \in Horn(\mathcal{P})$ such that M is the least model of $\mathcal{P}_h$. Therefore we have

Theorem 3.4 *Let $\mathcal{P}$ be a disjunctive logic program. Then for any minimal model M of $\mathcal{P}$, there is a minimal model M_C of $\mathcal{P}_C$ such that M is the least model of $\mathcal{P}_H(M_C)$.*

Example 3.5 *let*

$$\begin{aligned}\mathcal{P} \quad = \{ \quad & p \leftarrow q, r \\ & q \leftarrow r \\ & r \\ & s \leftarrow r_1 \\ & t \leftarrow r_1 \\ & p \vee s \\ & t \vee u \leftarrow r \ \}\end{aligned}$$

then the CH-transformation of $\mathcal{P}$ *is as follows.*

$$CH(\mathcal{P}) = \mathcal{P}_C \cup \mathcal{P}_H$$

where

$$\begin{aligned}\mathcal{P}_C \quad = \{ \quad & A_1 \vee A_2, \\ & B_1 \vee B_2 \ \}\end{aligned} \qquad \begin{aligned}\mathcal{P}_H \quad = \{ \quad & p \leftarrow q, r \\ & q \leftarrow r \\ & r \\ & s \leftarrow r_1 \\ & t \leftarrow r_1 \\ & p \leftarrow A_1 \\ & s \leftarrow A_2 \\ & t \leftarrow r, B_1 \\ & u \leftarrow r, B_2 \ \}\end{aligned}$$

and $C_{\mathcal{P}} = \{\ A_1,\ A_2,\ B_1,\ B_2\ \}$. $M_C = \{\ A_1,\ B_1\ \}$ *is a minimal model of* $\mathcal{P}_C$ *and*

$$\begin{aligned}\mathcal{P}_H(M_C) \quad = \{ \quad & p \leftarrow q, r \\ & q \leftarrow r \\ & r \\ & s \leftarrow r_1 \\ & t \leftarrow r_1 \\ & p \\ & t \leftarrow r \ \}\end{aligned}$$

Now we discuss the relationship between program $\mathcal{P}$ and $CH(\mathcal{P})$. To do so we need some notations about circumscription [McC80, GPP89, Lif85]

Given first-order theory $\mathcal{T}$, let P and Z be disjoint tuples of predicates from $\mathcal{T}$, then circumscription of P in $\mathcal{T}$ with variable Z is defined as the second-order formula:

$$Circ(T; P; Z) = T(P, Z) \wedge \neg\exists P'Z'(T(P', Z') \wedge P' < P)$$

where $T(P, Z)$ is a theory containing predicate constants P, Z, and P', Z' are tuples of predicate variables similar to P, Z. The set of all predicates other than P, Z from T is denoted by Q, which is called the fixed predicates.

Definition 3.6 *For any two models M and N of $\mathcal{T}$ we write $M \leq N$ $mod(P, Z)$ if models M and N differ only in how they interpret atoms from P and Z and if the extension of every predicate from P in M is a subset of its extension in N. A model M of $\mathcal{T}$ is called (P, Z)-minimal if there is no model N of $\mathcal{T}$ such that $N < M$ $mod(P, Z)$ (ie. such that $M \leq N$ $mod(P, Z)$ but not $N \leq M$ $mod(P, Z)$).*

Note: When $\mathcal{T}$ is a propositional theory, the definition of the relation $\leq$ can be restated as follows: For any two models M and N of $\mathcal{T}$ we write $M \leq N$ $mod(P, Z)$ if models M and N differ only in how they interpret atoms from P and Z and $M \cap P \subseteq N \cap P$.

The following theorem is due to [McC80, GPP89, Lif85], which explains the semantics of circumscription.

Theorem 3.7 *M is a model of $Circ(T; P; Z)$ iff M is a (P, Z)-minimal model of T. In other words, for any formula F, we have $Circ(T; P; Z) \models F$ iff $M \models F$ for every (P, Z)-minimal model M of T.*

Let $\mathcal{P}$ be a disjunctive logic program and $CH(\mathcal{P}) = \mathcal{P}_C \cup \mathcal{P}_H$ be the CH-transformation of $\mathcal{P}$. In the following we always let $P = HB_{\mathcal{P}}$ and for any model M of $CH(P)$, denote $M_C = M \cap C_{\mathcal{P}}$ and $M_H = M \cap P$ and write $M = M_C \cup M_H$.

Lemma 3.8 *Let $\mathcal{P}$ be a disjunctive logic program and $CH(\mathcal{P}) = \mathcal{P}_C \cup \mathcal{P}_H$. If $M = M_C \cup M_H$ is a model of $CH(\mathcal{P})$, then M_H is a model of $\mathcal{P}$.*

Proof: Let $M = M_C \cup M_H$ be a model of $CH(\mathcal{P})$. If M_H is not a model of $\mathcal{P}$, then there is a clause C of form

$$a_1 \vee a_2 \vee ... \vee a_n \leftarrow b_1, ..., b_m$$

such that $b_1, ..., b_m$ are true in M_H and for any $1 \leq i \leq n$, $a_i \notin M_H$. It must be that $n > 1$, otherwise $C \in CH(\mathcal{P})$, which contradicts that M is a model of $CH(\mathcal{P})$. Let $C_1 \vee C_2 \vee ... \vee C_n$ be the control clause in $CH(C)$; because M is a model of $CH(\mathcal{P})$, there is i $(1 \leq i \leq n)$ such that $C_i \in M$, then $b_1, ..., b_m, C_i$ must be true in M, therefore, $a_i \in M$. But a_i is in $\mathcal{P}$, so $a_i \in M_H$. This is a contradiction and we conclude that M_H is a model of $\mathcal{P}$.

Theorem 3.9 *Let $\mathcal{P}$ be a disjunctive logic program and $CH(\mathcal{P}) = \mathcal{P}_C \cup \mathcal{P}_H$. Then M is a minimal model of $\mathcal{P}$ iff there exists a minimal model M_C of $\mathcal{P}_C$ such that $M_C \cup M$ is an $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$.*

Proof: Let M be a minimal model of $\mathcal{P}$, we prove that there is a M_C such that $M_C \cup M$ is an $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$. For the minimal model M, define M_C as follows: For each clause C of the form

$$a_1 \vee a_2 \vee ... \vee a_n \leftarrow b_1, ..., b_m$$

in $\mathcal{P}$, where $n > 1$, let $C_1 \vee C_2 \vee ... \vee C_n$ be the control clause in $CH(C)$. If $b_1, ..., b_m$ is true in M, then there exists i such that $a_i \in M$, let $C_i \in M_C$. Otherwise let $C_1 \in M_C$. Nothing else is in M_C. It is clear that M_C is a minimal model of $\mathcal{P}_C$.

$M_C \cup M$ is a model of $CH(\mathcal{P})$. If not so, then there is a clause of $a_i \leftarrow b_1, ..., b_m, C_i$ such $b_1, ..., b_m, C_i$ is true in $M_C \cup M$ but a_i is not in M. By the construction of M_C this is impossible because in this case $C_i \in M_C$ if and only if $a_i \in M$.

$M_C \cup M$ is an $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$. If not so, then there is a minimal model $M'_C \cup M'_H$ of $CH(\mathcal{P})$ such that $M'_H \subset M$, By lemma 3.8, M'_H is a model of $\mathcal{P}$. This contradicts that M is a minimal model of $\mathcal{P}$.

To prove the other direction of the theorem, let M_C be a minimal model of $\mathcal{P}_C$ such that $M_C \cup M$ is a $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$, then by lemma 3.8, M is a model of $\mathcal{P}$. If M is not a minimal model, then there is a minimal model M' of $\mathcal{P}$ such that $M' \subset M$, for this M', we can define M'_C as above such that $M'_C \cup M'$ is an $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$ but $M' \subset M$. This contradicts that $M_C \cup M$ is an $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$.

By theorem 3.7 and theorem 3.9 we finally conclude

Theorem 3.10 *Let $\mathcal{P}$ be a disjunctive logic program and $CH(\mathcal{P}) = \mathcal{P}_C \cup \mathcal{P}_H$. Then for any formula α which contains no new propositional atoms, we have $\mathcal{P} \models_{min} \alpha$ iff $Circ(CH(\mathcal{P}); P; C_{\mathcal{P}}) \models \alpha$.*

4 Query Answering under Minimal Model Semantics

In this section we discuss how to answer queries under minimal model semantics. Because positive query can be answered by any existing theorem prover, for example PROTEIN [BF94], we concentrate only on negative query, that is, for a given atom q, to evaluate if $\neg q$ is true in all minimal models. The algorithm presented below can be combined with a existing theorem prover to get a sound and complete minimal model reasoning procedure. An example of such an approach can be found in [Ara96]

Our method is based on SLD-resolution due to the CH-transformation. An SLD-resolution proof procedure provides a way to compute answers for a goal from a definite logic program. It starts with a goal clause, $\leftarrow a_1, ..., a_n$, and provides a refutation by deriving an empty goal clause $\square$. The SLD derivation for definite propositional logic programs can be stated as follows.

Definition 4.1 (SLD derivation) *Let $\mathcal{P}$ be a definite logic program and G be the goal $\leftarrow a_1, ..., a_m, ..., a_n$. An SLD derivation from $\mathcal{P}$ with top goal G consists of a (finite or infinite) sequence of goals $G_0 (= G), G_1, ...,$ such that for all $i \geq 0$, G_{i+1} is obtained from G_i as follows:*

1. *a_m is an atom in G_i and is called the selected atom.*
2. *$a_m \leftarrow b_1, ..., b_q$ is a program clause in $\mathcal{P}$.*
3. *G_{i+1} is the goal $\leftarrow a_1, ..., a_{m-1}, b_1, ..., b_q, a_{m+1}, ..., a_n$*

SLD-derivations can be finite or infinite. An SLD refutation from $\mathcal{P}$ with top goal is a finite SLD derivation of empty set, from $\mathcal{P}$. A finite SLD-derivation can be successful or failed. A successful derivation is one that ends in the empty clause, that is, it is a refutation. A failed SLD-derivation is one that ends in a non-empty goal with the property that no atom in it occurs in the head of clauses in $\mathcal{P}$. SLD resolution is the system that uses SLD derivation as an inference mechanism.

Definition 4.2 *Let $\mathcal{P}$ be a definite program, G a goal and R a computation rule. Then the SLD-tree for $\mathcal{P} \cup \{G\}$ via R is defined as follows:*

1. *Each node of the tree is a goal(possibly empty).*
2. *The root node is G.*
3. *Let $\leftarrow a_1, ..., a_m, ..., a_k$ $(k \geq 1)$ be a node in the tree and suppose that a_m is the atom selected by R. Then this node has descendent for each input clause $a_m \leftarrow b_1, ..., b_q$. The descendent is*

$$\leftarrow a_1, ..., b_1, ..., b_q, ..., a_k$$

4. *Nodes which are empty clause have no descendents.*

Each branch of the SLD-tree is a derivation of $\mathcal{P} \cup \{G\}$. Branches corresponding to successful derivation are called success branches, branches corresponding to infinite derivation are called infinite branches and branches corresponding to failed derivation are called failure branches.

It has been proved that SLD-resolution is sound and complete wrt. definite programs. In propositional case, it can be stated as follows:

Theorem 4.3 *Let $\mathcal{P}$ be a definite logic program and $G = \leftarrow a_1, ..., a_n$ be a goal. Then $\mathcal{P} \models a_1, ..., a_n$ iff there is SLD-refutation for $\mathcal{P}$ and top goal G.*

Now we are in a position to present a proof procedure for negative query under minimal model semantics of disjunctive logic program based on CH-transformation and SLD-resolution. The following trivial facts can help us to understand the basic idea behind the method.

Proposition 4.4 *Let $\mathcal{P}$ be a disjunctive program and $G = \leftarrow a_1, ..., a_n$ be a goal. Considering CH-transformation of $\mathcal{P}$*

$$CH(\mathcal{P}) = \mathcal{P}_C + \mathcal{P}_H$$

denote $\bar{\mathcal{P}}_H$ the Horn program obtained from $\mathcal{P}_H$ by deleting all control variables. Then

1. *For any $a \in T_{\mathcal{P}_H} \uparrow \omega$, $\mathcal{P} \models_{min} a$.*
2. *For each $a \in T_{\bar{\mathcal{P}}_H} \uparrow \omega$, there is a model M of $\mathcal{P}$ such that a is true in M*
3. *If M is a minimal model of $\mathcal{P}$, then $M \subseteq T_{\bar{\mathcal{P}}_H} \uparrow \omega$*

To prove $\neg q$, we construct a SLD-tree T for $\mathcal{P}_H \cup \{\leftarrow q\}$. Let $\bar{T}$ be the tree obtained from T by deleting all control variables, then $\bar{T}$ is a SLD-tree for $\bar{\mathcal{P}}_H \cup \{\leftarrow q\}$. For a branch b in T let $\bar{b}$ denote the corresponding branch in $\bar{T}$. A branch b in T may be

1. a success branch. By 1 of Proposition 4.4, it means q is true in all minimal models of $\mathcal{P}$.
2. an infinite branch. Then $\bar{b}$ is also an infinite branch.
3. a failure branch and ends in a non-empty goal containing atoms in $\mathcal{P}$ (and also possibly some new propositional atoms from $C_{\mathcal{P}}$). This case corresponds to a failure branch in $\bar{T}$.
4. a failure branch but ends in a non-empty goal consisting of only new propositional atoms. Let the leaf node of the branch b be $\leftarrow N_1, ..., N_m$ $(m > 0)$. In this case b corresponds to a success branch $\bar{b}$ in $\bar{T}$, by 2. of Proposition 4.4, q is true in some model of $\mathcal{P}$. Notice the fact that N_i $(1 \leq i \leq m)$ indicates that the clause with N_i in its body have been used as input clauses in the derivation corresponding branch b, therefore for any model of $CH(\mathcal{P})$, if it contains N_1,...,N_m, then q must be true in it. Among them if there is a $(P, C_{\mathcal{P}})$-minimal model of $CH(\mathcal{P})$, then by Theorem 3.9 we can conclude that there is a minimal model of $\mathcal{P}$ in which q is true.

Hence, the problem of answering a query G is reduced to answering the following problem:

> Let S be a subset of $C_{\mathcal{P}}$, is there a $(P, C_{\mathcal{P}})$-minimal model $M = M_C \cup M_H$ of $CH(\mathcal{P})$ such that $S \subseteq M_C$? In other words, is there a model $M = M_C \cup M_H$ of $CH(\mathcal{P})$ such that $S \subseteq M_C$ and M_H is a minimal model of $\mathcal{P}$?

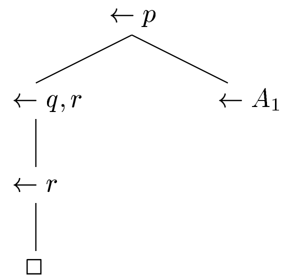

Figure 1: The SLD-tree for $\mathcal{P}_H$ and $\leftarrow p$

There are many ways to test the minimality of a model [Lu97, Nie96]. The following result is due to [Nie96]

Proposition 4.5 *[Nie96] Let $\mathcal{P}$ be a set of clauses. An interpretation M is a minimal model of $\mathcal{P}$ iff M is a model of $\mathcal{P}$ and for every atom a, $M \models a$ implies $\mathcal{P} \cup N_{\mathcal{P}}(M) \models a$, where $N_{\mathcal{P}}(M) = \{\neg a \mid a$ is an atom appearing in the head of a clause in $\mathcal{P}$ and $M \not\models a\}$.*

Now the problem can be solved as follows:

1. Computing all models containing S of $\mathcal{P}_C$, denote it by $MM(\mathcal{P}_C, S)$ (it can be done by any existing model generation procedure [Lu97, Nie96].
2. For each $M_C \in MM(\mathcal{P}_C, S)$, computing the least model of $\mathcal{P}_H(M_C)$[2] and testing if it is a minimal model of $\mathcal{P}$ using Proposition 4.5.

Let us summarize the procedure for minimal model reasoning.

Algorithm

Input: A disjunctive logic program $\mathcal{P}$ and an atom q.
Output: $\mathcal{P} \vdash_{min} \neg q$, if no minimal model contains q, otherwise either $\mathcal{P} \vdash_{min} q$ or a minimal model in which q is true.

1. Transform $\mathcal{P}$ into $\mathcal{P}_C + \mathcal{P}_H$.
2. Construct a SLD-tree T for $\mathcal{P}_H$ and $\leftarrow q$
3. If there is a success branch in T then return $\mathcal{P} \vdash_{min} q$, otherwise
4. If there is failure branches ending with subgoal consisting of only control variables and if there is a $(P, C_{\mathcal{P}})$-minimal model $M_C \cup M_{\mathcal{P}}$ such that the subgoal is contained in M_C, then return $M_{\mathcal{P}}$, otherwise
5. return $\mathcal{P} \vdash_{min} \neg q$.

Before ending this section we illustrate the algorithm with an example.

Example 4.6 *Consider the program $\mathcal{P}$ from example 3.5. Given goal $\leftarrow p$. A SLD-tree for the Horn program $\mathcal{P}_H$ and goal $\leftarrow p$ is depicted in Fig. 1. In this SLD-tree, there is a success branch, and hence we can conclude $\mathcal{P} \vdash_{min} p$.*

[2]This can be done by Hyper tableau or the immediate consequence operator for Horn programs.

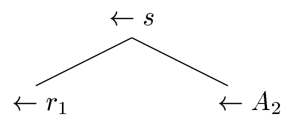

Figure 2: The SLD-tree for $\mathcal{P}_H$ and $\leftarrow s$

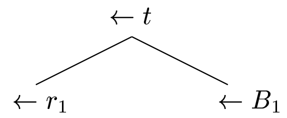

Figure 3: The SLD-tree for $\mathcal{P}_H$ and $\leftarrow t$

Fig. 2 gives an SLD-tree for the Horn program $\mathcal{P}_H$ and the goal $\leftarrow s$. It contains no success branch but has a branch ending with a subgoal $\leftarrow A_2$. Therefore a test is needed. There are two minimal models $\{A_2, B_1\}$ and $\{A_2, B_2\}$ of $\mathcal{P}_C$ containing A_2. The models of $CH(\mathcal{P})$ "controlled" by the two models are
$M_1 = T_{\mathcal{P}_H} \uparrow \omega(\{A_2, B_1\}) = \{A_2, B_1, r, p, q, s, t\}$ *and*
$M_2 = T_{\mathcal{P}_H} \uparrow \omega(\{A_2, B_2\}) = \{A_2, B_2, r, p, q, s, u\}$.
Both M_1 and M_2 are not $(P, C_{\mathcal{P}})$-minimal models, therefore we conclude $\mathcal{P} \vdash_{min} \neg s$.

Fig. 3 gives an SLD-tree for the Horn program $\mathcal{P}_H$ and goal $\leftarrow t$. It also contains no success branch, but it has a branch ending with a subgoal $\leftarrow B_1$. Therefore a test is needed. There are two minimal models $\{A_1, B_1\}$ and $\{A_2, B_1\}$ of $\mathcal{P}_C$ containing B_1, the models of $CH(\mathcal{P})$ "controlled" by the two models are
$M_1 = T_{\mathcal{P}_H} \uparrow \omega(\{A_1, B_1\}) = \{A_1, B_1, r, p, q, t\}$ *and*
$M_2 = T_{\mathcal{P}_H} \uparrow \omega(\{A_2, B_1\}) = \{A_2, B_1, r, p, q, s, t\}$
By test, M_1 is a $(P, C_{\mathcal{P}})$-minimal models and by Theorem 3.9, $M = \{r, p, q, t\}$ is a minimal model of P. So we return M as the output.

5 Conclutions and Further Work

In this paper we proposed a novel point of view on disjunctive logic programming. We gave a transformation which transforms a disjunctive logic program $\mathcal{P}$ into a special disjunctive logic program $CH(\mathcal{P}) = \mathcal{P}_C \cup \mathcal{P}_H$, where $\mathcal{P}_C$ is control program and $\mathcal{P}_H$ is a Horn program. Then the relationship between $\mathcal{P}$ and $CH(\mathcal{P})$ is established. Based on this relation we developed a minimal model reasoning procedure which differs from similar work in that the SLD-resolution can be directly incorporated into it, due to the transformation.

Many interesting topics remain to be done. Currently we are working on the following topics:

- Optimizing the query answering procedure. Many optimizations in the minimal test phase are posible. For example, the minimal model generation method based on E-hyper tableau [Lu97] can be used in the procedure to improve performence.
- Extending the idea for normal logic program under stable model semantics.
- Extending this method for general disjunctive logic program under stable model semantics.
- Extending this method for non-ground case.

Acknowledgements

The author would like to thank all the members of Artificial Intelligence Research Group at the University of Koblenz for their support and useful comments. Special thanks to Chandrabose Aravindan for his invaluable comments and encouragement. The first author is financially supported by the Graduate Scholarship of University of Koblenz.

References

[Ara96] Chandrabose Aravindan. An abductive framework for negation in disjunctive logic programming. In J. J. Alferes, L. M. Pereira, and E. Orlowska, editors, *Proceedings of Joint European workshop on Logics in AI*, number 1126 in Lecture Notes in Artificial Intelligence, pages 252–267. Springer-Verlag, 1996. A related report is available on the web from <http://www.uni-koblenz.de/~arvind/papers/>.

[BF94] Peter Baumgartner and Ulrich Furbach. PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface. In A. Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *Lecture Notes in Aritificial Intelligence*, pages 769–773. Springer, 1994. Available in the WWW, URL: `http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/`.

[BF97] Peter Baumgartner and Ulrich Furbach. Calculi for Disjunctive Logic Programming. In Jan Maluszynski, editor, *Logic Programming - Proceedings of the 1997 International Symposium*, Port Jefferson, New York, 1997. The MIT Press.

[BFN96] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, number 1126 in Lecture Notes in Aritificial Intelligence. European Workshop on Logic in AI, Springer, 1996.

[BFS97] Peter Baumgartner, Ulrich Furbach, and Frieder Stolzenburg. Computing Answers with Model Elimination. *Artificial Intelligence*, 90(1–2):135–176, 1997.

[Gin89] Matthew L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[GPP89] M. Gelfond, H. Przymusinska, and T. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:75–94, 1989.

[IKH92] K. Inoue, M. Koshimura, and R. Hasegawa. Embedding negation as failure into a model generation theorem prover. In *The 11th International Conference on Automated Deduction*, pages 400–415, Saratoga Springs, NY, USA, June 1992. Springer-Verlag.

[Lif85] V. Lifschitz. Computing circumscription. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 121–127, Los Angeles, California, USA, August 1985. Morgan Kaufmann Publishers.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer–Verlag, second extended edition, 1987.

[LMR92] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of disjunctive logic programming*. MIT Press, 1992.

[Lov91] D. Loveland. Near-Horn Prolog and Beyond. *Journal of Automated Reasoning*, 7:1–26, 1991.

[LRW95] D. Loveland, D. Reed, and D. Wilson. SATCHMORE: SATCHMO with RElevance. *Journal of Autmated Reasoning*, 14:325–351, 1995.

[Lu97] Wenjin Lu. Minimal model generation based on e-hyper tableau. In Cristopher Habel Gerhard Brewka and Bernhard Nebel, editors, *Proceedings of KI'97*, number 1303 in Lecture Notes in Artificial Intelligence, pages 99–110. Springer-Verlag, 1997.

[MB88] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In *Proc. 9th CADE*. Argonnee, Illinois, Springer LNCS, 1988.

[McC80] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[Min82] Jack Minker. On indefinite databases and the closed world assumption. In *Lecture Notes in Computer Science 138*, pages 292–308. Springer-Verlag, 1982.

[Nie96] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 278–294, Terrasini, Italy, May 1996. Springer-Verlag.

[NNS95] A. Nerode, R.T. Ng, and V.S. Subrahmanian. Computing circumscriptive databases: I. theory and algorithms. *Information and Computation*, 116:58–80, 1995.

[Oli92] N. Olivetti. A tableaux and sequent calculus for minimal entailment. *Journal of Automated Reasoning*, 9:99–139, 1992.

[Prz89] T.C. Przymusinski. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1989.

[YH85] A. Yahya and L. J. Henschen. Deduction in non-horn databases. *Journal of Automated Reasoning*, 1(2):141–160, 1985.