DisLog – A System for Reasoning in Disjunctive Deductive Databases

Dietmar Seipel, Helmut Thöne
University of Tübingen
Sand 13, D – 72076 Tübingen, Germany
{seipel,thoene}@informatik.uni-tuebingen.de

Abstract

DisLog is a system for reasoning in disjunctive deductive databases. It seeks to combine features of disjunctive logic programming, such as the support for incomplete information, with those of deductive databases, such as all–result inference capabilities. Several basic operators are provided for logical and non–monotonic reasoning: The logical consequence operator derives all logically implied disjunctive clauses from a disjunctive logic program. The non–monotonic operators are semantically founded on generalizations of the well–known closed–world–assumption. Reasoning in disjunctive deductive databases is very complex, even for small examples. Many different optimization techniques are integrated in DisLog to speed up the application performance. The main techniques rely on a *clause tree* data structure allowing for an efficient and transparent evaluation. The operators of DisLog can be loaded from a library into a Prolog application. DisLog itself is implemented as a meta–interpreter in Sicstus–Prolog.

1 Introduction

During the last couple of years, a growing interest in the field of disjunctive logic programming can be observed (cf. workshops at the *ICLP*, *ILPS* and *IFIP*'94 conferences). A large number of theoretical publications have brought a deep insight into semantic issues and non–monotonic reasoning: Lobo, Minker, Rajasekar [18], Van Gelder, Ross, Schlipf [27] and Przymusinski [21] have developed alternative semantics. Marek, Truszczyński [19] and Dix [10] give an overview of non–monotonic reasoning. As a marriage of disjunctive logic programming and database technologies, disjunctive deductive databases can form the foundation for integrated high–end reasoning systems. Advanced applications in which incomplete information must be extensively analyzed are likely to benefit from this technology. For instance, the analysis of independencies in probabilistic reasoning (cf. Pearl [20] and Seipel, Thöne [25]), and the overlapping of three–dimensional objects in spatial databases (cf. Abdelmoty, Williams, Paton [1]) are applications falling into this category.

In this paper, we focus on the DisLog system which provides a great deal of operators for reasoning in disjunctive deductive databases. In particular, DisLog supports the derivation of logical consequences (cf. Seipel [23]), generalizations of the closed–world–assumptions (cf. Reiter [22], Lobo, Minker, Rajasekar [18]), generalizations of the well–founded semantics (cf. Van Gelder, Ross, Schlipf [27]) and generalizations of the concept of negation–as–failure (cf. Baral [2]). Thus, depending on the semantics of an application, a suitable set of operators can be chosen by the user. To provide an introductory understanding of the different semantics of logics programs, let us compare the meaning of a definite logic program with the meaning of a disjunctive logic program.

The meaning of a definite logic program P is usually given by a model theoretic and a fixpoint semantics. A definite logic program consists of a set of definite clauses of the form

$$A \text{ :- } B_1, \ldots, B_m$$

where A and the B_i $(1 \leq i \leq m, m \in I\!N_0)$ are atomic formulas. The model theoretic semantics of the program is given by a unique *minimal model* $\mathcal{M}_P$. It consists of a set of atoms which are *true* in all models of the program. On the other hand, the fixpoint semantics is based on a logical consequence operator $\mathcal{T}_P$ that maps sets of atoms to sets of atoms. Starting from the facts which are *true*, repeated application of $\mathcal{T}_P$ produces a fixpoint in a bottom–up fashion and this fixpoint is equal to $\mathcal{M}_P$. Besides the derivation of logically implied atoms, a mechanism for deriving negated atoms is offered by the *closed–world–assumption* $\mathcal{CWA}_P$. An atom is considered to be *false* if the atom is not a logical consequence of the program. $\mathcal{T}_P$ is a monotonic operator whereas $\mathcal{CWA}_P$ is a non–monotonic operator: If we enlarge the program, then the set of derived atoms is increasing whereas the set of derived negated atoms can decrease.

A different meaning is attached to a disjunctive logic program P. It consists of a set of disjunctive clauses of the form

$$A_1 \vee \ldots \vee A_k \text{ :- } B_1, \ldots, B_m$$

where the A_j $(1 \leq j \leq k, k \in I\!N)$ and the B_i $(1 \leq i \leq m, m \in I\!N_0)$ are atoms. In general, disjunctive logic programs do not have a unique minimal model. For instance, the program $P = \{A_1 \vee A_2\}$ has three models: $\{A_1\}$, $\{A_2\}$ and $\{A_1, A_2\}$. The models $\{A_1\}$ and $\{A_2\}$ are minimal. Therefore, the model theoretic semantics is given by a *set of minimal models* $\mathcal{MM}_P$. A disjunction over $\mathcal{MM}_P$ forms a unique *minimal model state* $\mathcal{MS}_P$. In the example, $\mathcal{MS}_P$ is equal to the set $\{A_1 \vee A_2\}$. On the other hand, the fixpoint semantics is based on a logical consequence operator $\mathcal{T}_P^S$ that maps sets of disjunctive clauses to sets of disjunctive clauses. Starting from the disjunctive facts which are *true*, repeated application of $\mathcal{T}_P^S$ produces a fixpoint which is equal to $\mathcal{MS}_P$. That is, $\mathcal{T}_P^S$ derives all disjunctive clauses logically implied by each minimal model of the program. For the derivation of negated atoms, the closed–world–assumption cannot be used since it may lead to inconsistencies. In our example, by $\mathcal{CWA}_P$, we get $\{\neg A_1, \neg A_2\}$, but this is inconsistent with $P = \{A_1 \vee A_2\}$. Here, the *extended generalized closed–world–assumption* $\mathcal{EGCWA}_P$ is an appropriate non–monotonic operator. A conjunction of atoms is considered to be *false* if it is not a logical consequence of any minimal model of the program. That is, by $\mathcal{EGCWA}_P$, we conclude that $A_1 \wedge A_2$ is *false* in the above example.

The rest of this paper is organized as follows: In section 2, we will review some of the definitions and theoretical foundations for disjunctive logic programming, their fixpoint semantics and concepts of non–monotonic reasoning, as given in [18]. Then in section 3, we present the main data structure, called *clause tree*, for the compact representation of sets of disjunctive clauses in DisLog. Section 4 describes concepts for bottom–up evaluation techniques to compute $\mathcal{MS}_P$ and $\mathcal{EGCWA}_P$. Some sample sessions with DisLog are given in the appendix.

2 Disjunctive Logic Programming

A *disjunctive normal logic program* P consists of a set of *disjunctive normal logic rules* r of the form

$$A_1 \vee \ldots \vee A_k \text{ :- } B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_{m+n}$$

where the disjunction $A_1 \vee \ldots \vee A_k$ of atoms A_i ($1 \leq i \leq k$, $k \in I\!N$) is called the head of r and the conjunction $B_1 \wedge \ldots \wedge B_m \wedge \neg B_{m+1} \wedge \ldots \wedge \neg B_{m+n}$ with atoms B_i ($1 \leq i \leq m+n$, $m \in I\!N_0$, $n \in I\!N_0$) is called the body of r. If $n = 0$, then r is called a *disjunctive rule*. If $m + n = 0$, then r is called a *disjunctive fact*, also denoted by

$$A_1 \vee \ldots \vee A_k.$$

P is called *disjunctive logic program* if all rules in P are disjunctive. The set of ground instances of the rules in P is denoted by $Ground(P)$.

Let HB_P be the Herbrand base of P. The *disjunctive Herbrand base* DHB_P of P is a set of positive ground disjunctions $A_1 \vee \ldots \vee A_k$ of atoms $A_i \in HB_P$ ($1 \leq i \leq k$, $k \in I\!N_0$). The *conjunctive Herbrand base* CHB_P of P is a set of positive ground conjunctions $A_1 \wedge \ldots \wedge A_k$ of atoms $A_i \in HB_P$ ($1 \leq i \leq k$, $k \in I\!N_0$). A *disjunctive Herbrand state* of P is a subset $T \subseteq DHB_P$. A *conjunctive Herbrand state* of P is a subset $F \subseteq CHB_P$. Then $S = (T, F)$ is called *state pair*. The disjunctions in T will be considered *true* and the conjunctions in F will be considered *false*. The *union* of two state pairs is computed state–by–state: $(T_1, F_1) \cup (T_2, F_2) = (T_1 \cup T_2, F_1 \cup F_2)$.

The *canonical form* and the *expansion* are defined for $T \subseteq DHB_P$ as

$$\begin{aligned} can(T) &= \{\, C \in T \mid \text{there is no proper subdisjunction } C' \text{ of } C \text{ in } T \,\}, \\ exp(T) &= \{\, C \in DHB_P \mid \text{there is a subdisjunction } C' \text{ of } C \text{ in } T \,\}, \end{aligned}$$

and for $F \subseteq CHB_P$ as

$$\begin{aligned} can(F) &= \{\, C \in F \mid \text{there is no proper subconjunction } C' \text{ of } C \text{ in } F \,\}, \\ exp(F) &= \{\, C \in CHB_P \mid \text{there is a subconjunction } C' \text{ of } C \text{ in } F \,\}. \end{aligned}$$

Therefore, $can(H) \subseteq H \subseteq exp(H)$, for a disjunctive or conjunctive Herbrand state H. H is called expanded if $exp(H) = H$.

The *set difference with subsumption* $T_1 \setminus_s T_2$ and the *set disjunction* $T_1 \vee T_2$ are defined for two disjunctive Herbrand states T_1 and T_2 as

$$\begin{aligned} T_1 \setminus_s T_2 &= \{ \ C \in T_1 \mid \text{there is no subdisjunction } C' \text{ of } C \text{ in } T_2 \ \}, \\ T_1 \vee T_2 &= \{ \ C_1 \vee C_2 \mid C_1 \in T_1, \ C_2 \in T_2 \ \}. \end{aligned}$$

The *negation* of a conjunctive Herbrand state F is given by

$$\neg F \ = \ \{ \ \neg A_1 \vee \ldots \vee \neg A_k \mid A_1 \wedge \ldots \wedge A_k \in F \ \}.$$

Example 1 For the Herbrand base $HB_P = \{a, b, c\}$ and the disjunctive Herbrand states $T_1 = \{a \vee b, a \vee c\}$ and $T_2 = \{a \vee b, b \vee c\}$, we get $T_3 = T_1 \vee T_2 = \{a \vee b, a \vee b \vee c\}$, $T_1 \setminus_s T_3 = \{a \vee c\}$, $T_4 = can(T_3) = \{a \vee b\}$ and $exp(T_4) = T_3$.

2.1 The Disjunctive Consequence Operator

The fixpoint semantics of a disjunctive logic program is based on a disjunctive consequence operator.

Definition 2 Given a disjunctive logic program P.

(i) The disjunctive *consequence operator* $\mathcal{T}_P^S : 2^{DHB_P} \rightarrow 2^{DHB_P}$ of P is defined as

$$\begin{aligned} \mathcal{T}_P^S(T) \ = \ \{ \ & C' \vee C_1 \vee \ldots \vee C_n \mid C' \ :\!- \ B_1, \ldots, B_n \text{ is in } Ground(P), \\ & B_1 \vee C_1, \ldots, B_n \vee C_n \text{ are in } T \ \}. \end{aligned}$$

(ii) The *powers* of $\mathcal{T}_P^S$ are defined as

$$\begin{aligned} \mathcal{T}_P^S \uparrow 0 &= \emptyset, \\ \mathcal{T}_P^S \uparrow n &= \mathcal{T}_P^S(\mathcal{T}_P^S \uparrow n-1), \text{ for } n \in I\!N, \\ \mathcal{T}_P^S \uparrow \omega &= \cup_{n<\omega} \mathcal{T}_P^S \uparrow n. \end{aligned}$$

$\mathcal{T}_P^S$ is a *monotonic* and *continuous* mapping on 2^{DHB_P}. Since 2^{DHB_P} is a complete lattice w.r.t. set inclusion, the least fixpoint $lfp(\mathcal{T}_P)$ is equal to $\mathcal{T}_P \uparrow \omega$, cf. the book of Lloyd [17].

Example 3 Let us consider the classical example of transitive closure:

$$\begin{aligned} R \ = \ \{ \ & path(X,Y) \ :\!- \ path(X,Z), path(Z,Y); \\ & path(X,Y) \ :\!- \ arc(X,Y) \ \}. \end{aligned}$$

Suppose we have disjunctive information about the *arc*–relation, cf. Figure 1:

$$T \ = \ \{ \ arc(a,b) \vee arc(a,c), \ arc(b,d), \ arc(c,d) \ \}.$$

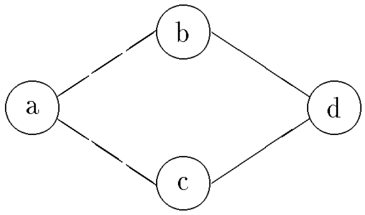

Figure 1: Graph with disjunctive arcs

Then, for $P = R \cup T$ we get $\mathcal{T}_P^S \uparrow n = T_1 \cup \ldots \cup T_n$ ($n \in I\!N$), where

$$
\begin{aligned}
T_1 &= T, \\
T_2 &= \{\ arc(a,b) \vee path(a,c),\ path(a,b) \vee arc(a,c),\ path(b,d),\ path(c,d)\ \}, \\
T_3 &= \{\ path(a,b) \vee path(a,c),\ path(a,d) \vee arc(a,c),\ path(a,d) \vee arc(a,b)\ \}, \\
T_4 &= \{\ path(a,d) \vee path(a,c),\ path(a,d) \vee path(a,b)\ \}, \\
T_5 &= \{\ path(a,d)\ \}, \\
T_n &= \emptyset,\ \mathit{for}\ n \geq 6.
\end{aligned}
$$

Thus, $\mathcal{T}_P^S \uparrow \omega = \mathcal{T}_P^S \uparrow 5$. This example shows that $\mathcal{T}_P^S$ can also derive definite facts from disjunctive facts.

The notation for the $\mathcal{T}_P^S$–operator in DisLog is as follows:

tps_operator(*+Program*,*+State1*,*?State2***)**
computes the disjunctive consequence *State2* from a disjunctive logic program *Program* and a Herbrand state *State1*.

2.2 The Minimal Model State

For a definite logic program P, the minimal Herbrand model $\mathcal{M}_P$ is equal to the set of logical consequences $A \in HB_P$ and the least fixpoint $\mathit{lfp}(\mathcal{T}_P)$ of the definite consequence operator $\mathcal{T}_P$, cf. [17].

For a disjunctive logic program P, however, in general there is not a unique minimal Herbrand model, but a set $\mathcal{MM}_P$ of minimal Herbrand models.

mm_operator(*+Program*,*?Models***)**
computes the set *Models* of all minimal models of a disjunctive logic program *Program*.

The models of definite logic programs correspond to model states of disjunctive logic programs. As for Herbrand models of definite logic programs, every intersection of model states again is a model state.

Definition 4 Given a disjunctive logic program P.

(i) Let T be an expanded disjunctive Herbrand state of P. T is called a *model state* of P if $\mathcal{T}_P^S(T) \subseteq T$.

(ii) Let MS_P be the set of all model states of P. The intersection $\mathcal{MS}_P = \bigcap_{T \in MS_P} T$ of all model states in MS_P is called the *minimal model state*.

$\mathcal{MS}_P$ is the unique minimal element in the set of all model states of P. Model states can also be characterized by Herbrand models: an expanded disjunctive Herbrand state T of P is a model state of P iff every minimal Herbrand model of T is a Herbrand model of P. According to [18], $\mathcal{MS}_P$ is equal to the set of logical consequences $C \in DHB_P$ of P, the expansion of the least fixpoint $lfp(\mathcal{T}_P^S)$, and the set of disjunctions logically implied by all minimal Herbrand models of P, i.e.

$$\mathcal{MS}_P \;=\; \{\; C \in DHB_P \mid \forall M \in \mathcal{MM}_P : M \models C \;\}$$

For a definite logic program P, $\mathcal{M}_P = can(\mathcal{MS}_P)$.

mms_operator(*+Program*, *?State* **)**
computes the minimal model state *State* of a disjunctive logic program *Program*.

2.3 The Extended Generalized Closed–World–Assumption

Closed world assumptions are one of the major formalisms of non–monotonic reasoning in artificial intelligence. For a definite logic program P, the closed–world–assumption considers all ground atoms to be *false* which are not logical consequences of P. That is, these atoms are not logical consequences of the unique minimal Herbrand model of P. For a disjunctive logic program P, the *extended generalized closed–world–assumption* $\mathcal{EGCWA}_P$ considers all ground conjunctions to be *false* which are not logical consequences of any minimal Herbrand model of P.

Definition 5 Given a disjunctive logic program P.

$$\mathcal{EGCWA}_P \;=\; \{\; E \in CHB_P \mid \forall M \in \mathcal{MM}_P : M \not\models E \;\}.$$

Example 6 For the disjunctive logic program $P \;=\; \{\; c \text{ :- } a, \quad a \vee b \;\}$, we get the minimal model state $\mathcal{MS}_P = \{a \vee b, b \vee c, a \vee b \vee c\}$ and the set $\mathcal{MM}_P = \{\{a, c\}, \{b\}\}$ of minimal Herbrand models. $\mathcal{EGCWA}_P = \{a \wedge b, b \wedge c, a \wedge b \wedge c\}$. But, for the stronger disjunctive logic program $P' = P \cup \{c \vee a\}$, we get the weaker set $\mathcal{EGCWA}_{P'} = \{a \wedge b \wedge c\}$ of *false* conjunctions (*non–monotonicity*).

egcwa_operator(*+Program*,*?Egcwa*)

computes the extended generalized closed–world–assumption *Egcwa* of a disjunctive logic program *Program*.

The negation $\neg\mathcal{EGCWA}_P$ is maximal consistent with the minimal model state $\mathcal{MS}_P$: it contains exactly the ground disjunctions $\neg A_1 \vee \ldots \vee \neg A_k$ which can be assumed to be *true* without making it possible to derive new positive ground disjunctions from $\mathcal{MS}_P \cup \neg\mathcal{EGCWA}_P$, cf. [18].

2.4 Minimal Model Reasoning

For a disjunctive logic program P, the minimal model semantics is given by the state pair $S = (\mathcal{MS}_P, \mathcal{EGCWA}_P)$. This concept cannot be applied directly to disjunctive normal logic programs P. Therefore we transform P into a logically equivalent ground disjunctive logic program $Dis(P,S)$ w.r.t. a derived state pair S. $Dis(P,S)$ is produced from $Ground(P)$ in three steps: First, all negated body atoms of each rule are moved as atoms to the head. Secondly, all rules with *false* bodies or *true* heads w.r.t. S are removed, since these rules do not contribute to the deduction of new disjunctions. Thirdly, *true* body atoms and *false* head atoms are removed from each rule, cf. [18].

Definition 7 Given a disjunctive normal logic program P and given a state pair $S = (T, F)$.

(i) The *disjunctive transformation* of P is defined as

$$\begin{aligned} Dis(P) \;=\; \{\; & C \vee B_{m+1} \vee \ldots \vee B_{m+n} \text{ :- } B_1, \ldots, B_m \mid \\ & \qquad C \text{ :- } B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_{m+n} \in P \;\}. \end{aligned}$$

(ii) The *disjunctive transformation* $Dis(P,S)$ of P w.r.t. S is derived from $P' = Ground(Dis(P))$ as follows. Let $r = A_1 \vee \ldots \vee A_k \text{ :- } B_1, \ldots, B_m \in P'$:

- Remove the rule r if $A_{i_1} \vee \ldots \vee A_{i_l} \in T$ for some subdisjunction of its head or $A_1, \ldots, A_k \in F$ or $B_{i_1} \wedge \ldots \wedge B_{i_l} \in F$ for some subconjunction of its body, i.e. the head is *true* in S or *false* in S or the body is *false* in S, respectively.
- Reduce the rule r by removing *true* atoms $B_i \in T$ form the body of r and *false* atoms $A_i \in F$ from the head of r.

From $Dis(P,S)$ we can derive the minimal model state and the extended generalized closed–world–assumption.

Definition 8 Given a disjunctive normal logic program P.

The operator $\mathcal{S}_P^{\mathcal{E}}$ is defined for a state pair $S = (T, F)$ as

$$\mathcal{S}_P^{\mathcal{E}}(S) \;=\; S \;\cup\; (\mathcal{MS}_{Dis(P,S)}, \mathcal{EGCWA}_{Dis(P,S) \cup T}) \,.$$

Example 9 (cf. [18]) Given the disjunctive normal logic program P:

$$\begin{array}{ll} e \text{ :- } p, \neg f. & q \text{ :- } \neg a. \\ f \text{ :- } \neg e. & t \text{ :- } \neg b. \\ p \text{ :- } q, t. & a \vee b. \end{array}$$

The disjunctive transformation $Dis(P, S_0)$ of P w.r.t. the state pair $S_0 = (\emptyset, \emptyset)$ is

$$\begin{array}{ll} e \vee f \text{ :- } p. & a \vee q. \\ e \vee f. & b \vee t. \\ p \text{ :- } q, t. & a \vee b. \end{array}$$

and the operator $\mathcal{S}_P^{\mathcal{E}}$ derives the state pair

$$S_1 = \mathcal{S}_P^{\mathcal{E}}(S_0) = (\{a \vee b, a \vee q, b \vee t, e \vee f\}, \{a \wedge q, b \wedge t, e \wedge f, p, q \wedge t\}).$$

$Dis(P, S_1) = \emptyset$ and $\mathcal{S}_P^{\mathcal{E}}(S_1) = S_1$.

Since $Dis(P, S) \cup T$ is a disjunctive logic program, $\neg\mathcal{EGCWA}_{Dis(P,S)\cup T}$ is maximal consistent with $\mathcal{MS}_{Dis(P,S)\cup T}$. Thus, $\neg\mathcal{EGCWA}_{Dis(P,S)\cup T}$ is also consistent with $T \cup \mathcal{MS}_{Dis(P,S)}$, the first state in the state pair of $\mathcal{S}_P^{\mathcal{E}}(S)$.

se_operator(_+Program_**,**_+State_Pair1_**,**_?State_Pair2_**)**
computes _State_Pair2_ from the disjunctive normal logic program _Program_ and _State_Pair1_ by $\mathcal{S}_P^{\mathcal{E}}$.

Note that, although the disjunctive transformation $Dis(P, S)$ of a disjunctive normal logic program may be an infinite ground disjunctive logic program, there exists a method for deriving the minimal model state $\mathcal{MS}_{Dis(P,S)}$ of $Dis(P, S)$ based on the finite disjunctive logic program $Dis(P)$ and S, cf. the work of Seipel [24].

2.5 The Disjunctive Well–Founded Semantics

In Example 9, the $\mathcal{S}_P^{\mathcal{E}}$–operator could not derive any further information from the state pair S_1. But, in S_1 the atom p is *false*. Since only the first rule in P could derive the atom e as *true*, and this rule is blocked since p is *false*, we would like to derive the negative information that e is *false*. Moreover, by the second rule of P, we would like to derive the positive information that the atom f is *true*.

Definition 10 Given a disjunctive normal logic program P and given a state pair $S = (T_S, F_S)$.

(i) The *positive consequence operator* $\mathcal{T}_S^D : 2^{DHB_P} \to 2^{DHB_P}$ of P w.r.t. S operates on disjunctive Herbrand states:

$$\mathcal{T}_S^D(T) = \{\ C \in DHB_P \mid C \text{ :- } B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_{m+n} \in Ground(P) \text{ and } \{B_1 \vee C_1, \ldots, B_m \vee C_m\} \subseteq T_S \cup T \text{ and } \{B_{m+1}, \ldots, B_{m+n}\} \subseteq F_S\ \}.$$

(ii) The *negative consequence operator* $\mathcal{F}_S^D : 2^{HB_P} \to 2^{HB_P}$ of P w.r.t. S operates on subsets of the Herbrand base:

$$\begin{aligned}\mathcal{F}_S^D(F) = \{ & A \in HB_P \mid A \text{ is } \textit{unknown} \text{ w.r.t. S, i.e. } A \notin T_S \cup F_S, \text{ and} \\ & \text{for all } A \vee C \text{ :- } B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_{m+n} \in Ground(P) : \\ & B_1 \wedge \ldots \wedge B_m \in exp(F_S \cup F) \text{ or } B_{m+1} \vee \ldots \vee B_{m+n} \in exp(T_S) \}.\end{aligned}$$

The positive consequence operator derives *true* ground disjunctions from S. As for the disjunctive consequence operator, a ground instance of a disjunctive normal logic rule derives new disjunctions, if there are resolving disjunctions for the positive body atoms. Additionally, the negated body atoms must be known to be *false* from F_S.

The negative consequence operator derives *false* ground atoms. A ground instance of a disjunctive normal logic rule cannot be used for resolution if the conjunction of the positive body atoms is known to be *false* from $F_S \cup F$, or if the disjunction of the negated body atoms in known to be *true* from T_S.

Definition 11 Given a disjunctive normal logic program P.

(i) The powers of $\mathcal{T}_S^D$ and $\mathcal{F}_S^D$ are given by

$$\begin{aligned}\mathcal{T}_S^D \uparrow 0 &= \emptyset, \\ \mathcal{T}_S^D \uparrow n &= \mathcal{T}_S^D(\mathcal{T}_S^D \uparrow n-1), \text{ for } n \in I\!N, \\ \mathcal{T}_S^D \uparrow \omega &= \bigcup_{n<\omega} \mathcal{T}_S^D \uparrow n, \\ \mathcal{F}_S^D \downarrow 0 &= HB_P, \\ \mathcal{F}_S^D \downarrow n &= \mathcal{F}_S^D(\mathcal{F}_S^D \downarrow n-1), \text{ for } n \in I\!N, \\ \mathcal{F}_S^D \downarrow \omega &= \bigcap_{n<\omega} \mathcal{F}_S^D \downarrow n.\end{aligned}$$

(ii) The operator $\mathcal{S}_P^{\mathcal{D}}$ for a state pair S is given by

$$\mathcal{S}_P^{\mathcal{D}}(S) = S \cup (\mathcal{T}_S^D \uparrow \omega, \mathcal{F}_S^D \downarrow \omega).$$

The fixpoint of the negative consequence operator is the maximal set F of atoms A, such that no ground instance of a rule of the program P with the atom A in its head can be used to derive a disjunction containing A.

Definition 12 Given a disjunctive normal logic program P.

(i) The powers of $\mathcal{S}_P^{\mathcal{D}}$ are given by

$$\begin{aligned}\mathcal{S}_P^{\mathcal{D}} \uparrow 0 &= (\emptyset, \emptyset), \\ \mathcal{S}_P^{\mathcal{D}} \uparrow \alpha &= \mathcal{S}_P^{\mathcal{D}}(\mathcal{S}_P^{\mathcal{D}} \uparrow \alpha - 1), \text{ for a successor ordinal } \alpha, \\ \mathcal{S}_P^{\mathcal{D}} \uparrow \alpha &= \bigcup_{\beta<\alpha} \mathcal{S}_P^{\mathcal{D}} \uparrow \beta, \text{ for a limit ordinal } \alpha.\end{aligned}$$

(ii) The *disjunctive well–founded* state pair of P is given by $\mathcal{DWFS}_P = \mathcal{S}_P^{\mathcal{D}} \uparrow \delta$, where δ is the smallest ordinal, such that $\mathcal{S}_P^{\mathcal{D}} \uparrow \delta$ is a fixpoint of $\mathcal{S}_P^{\mathcal{D}}$.

Note, that the least fixpoint of $\mathcal{S}_P^{\mathcal{D}}$ is derived by transfinite fixpoint iteration, cf. [18].

Example 13 Given the disjunctive normal logic program P of Example 9 and given $S_0 = (\emptyset, \emptyset)$. The least fixpoint $lfp(\mathcal{S}_P^{\mathcal{D}}) = \mathcal{DWFS}_P$ is derived in a single step:

$$\mathcal{DWFS}_P \;=\; \mathcal{S}_P^{\mathcal{D}}(S_0) \;=\; (\{a \vee b\}, \emptyset).$$

Thus, $\mathcal{S}_P^{\mathcal{D}}(S_0)$ is weaker than $\mathcal{S}_P^{\mathcal{E}}(S_0)$. But, from

$$S_1 \;=\; \mathcal{S}_P^{\mathcal{E}}(S_0) \;=\; (\{a \vee b, a \vee q, b \vee t, e \vee f\}, \{a \wedge q, b \wedge t, e \wedge f, p, q \wedge t\})$$

we can derive further information:

$$S_2 \;=\; \mathcal{S}_P^{\mathcal{D}}(S_1) \;=\; (\{a \vee b, a \vee q, b \vee t, e \vee f\}, \{a \wedge q, b \wedge t, e, p, q \wedge t\}).$$

The information that e is *false* was not derived by $\mathcal{S}_P^{\mathcal{E}}$. By applying $\mathcal{S}_P^{\mathcal{D}}$ to S_2, we can derive that f is *true*:

$$S_3 \;=\; \mathcal{S}_P^{\mathcal{D}}(S_2) \;=\; (\{a \vee b, a \vee q, b \vee t, f\}, \{a \wedge q, b \wedge t, e, p, q \wedge t\}).$$

sd_operator(*+Program*, *+State_Pair1*, *?State_Pair2* **)**
computes *State_Pair2* from the disjunctive normal logic program *Program* and *State_Pair1* by $\mathcal{S}_P^{\mathcal{D}}$.

dwfs_operator(*+Program*, *?State_Pair* **)**
computes the disjunctive well–founded state pair *State_Pair* of the disjunctive normal logic program *Program*.

2.6 The Generalized Disjunctive Well–Founded Semantics

The generalized disjunctive well–founded semantics combines logical reasoning with both forms of non–monotonic reasoning (closed–world–assumptions and negation–as–failure). As shown in Example 13, alternating applications of $\mathcal{S}_P^{\mathcal{E}}$ and $\mathcal{S}_P^{\mathcal{D}}$ yield a stronger result than separate applications.

Definition 14 Given a disjunctive normal logic program P.

(i) The operator $\mathcal{S}_P^{\mathcal{ED}}$ is defined for a state pair S as

$$\mathcal{S}_P^{\mathcal{ED}}(S) \;=\; \mathcal{S}_P^{\mathcal{E}}(\mathcal{S}_P^{\mathcal{D}}(S)).$$

(ii) The powers of $\mathcal{S}_P^{\mathcal{ED}}$ are state pairs given by

$$\begin{array}{rcl} \mathcal{S}_P^{\mathcal{ED}} \uparrow 0 & = & (\emptyset, \emptyset), \\ \mathcal{S}_P^{\mathcal{ED}} \uparrow \alpha & = & \mathcal{S}_P^{\mathcal{ED}}(\mathcal{S}_P^{\mathcal{ED}} \uparrow \alpha - 1), \text{ for a successor ordinal } \alpha, \\ \mathcal{S}_P^{\mathcal{ED}} \uparrow \alpha & = & \bigcup_{\beta < \alpha} \mathcal{S}_P^{\mathcal{ED}} \uparrow \beta, \text{ for a limit ordinal } \alpha. \end{array}$$

(iii) The *generalized disjunctive well–founded* state pair of P is given by $\mathcal{GDWFS}_P = \mathcal{S}_P^{\mathcal{ED}} \uparrow \delta$, where δ is the smallest ordinal, such that $\mathcal{S}_P^{\mathcal{ED}} \uparrow \delta$ is a fixpoint of $\mathcal{S}_P^{\mathcal{ED}}$.

Thus, the least fixpoint *lfp*($\mathcal{S}_P^{\mathcal{ED}}$) is derived by transfinite fixpoint iteration.

Example 15 For the disjunctive normal logic program P of Example 9, the operators $\mathcal{S}_P^{\mathcal{D}}$ and $\mathcal{S}_P^{\mathcal{E}}$ derive the state pairs $S_{2i} = \mathcal{S}_P^{\mathcal{ED}} \uparrow i$ and $S_{2i+1} = \mathcal{S}_P^{\mathcal{D}}(\mathcal{S}_P^{\mathcal{ED}} \uparrow i)$ $(i \in I\!N_0)$:

$$\begin{array}{rcl} S_0 & = & (\emptyset, \emptyset), \\ S_1 & = & (\{a \vee b\}, \emptyset), \\ S_2 & = & (\{a \vee b, a \vee q, b \vee t, e \vee f\}, \{a \wedge q, b \wedge t, e \wedge f, p, q \wedge t\}), \\ S_3 & = & (\{a \vee b, a \vee q, b \vee t, e \vee f\}, \{a \wedge q, b \wedge t, e, p, q \wedge t\}), \\ S_4 & = & (\{a \vee b, a \vee q, b \vee t, f\}, \{a \wedge q, b \wedge t, e, p, q \wedge t\}), \\ S_n & = & S_4, \textit{ for } n \geq 5. \end{array}$$

sed_operator(*+Program***,***+State_Pair1***,***?State_Pair2***)**
computes *State_Pair2* from the disjunctive normal logic program *Program* and *State_Pair1* by $\mathcal{S}_P^{\mathcal{ED}}$.

gdwfs_operator(*+Program***,***?State_Pair***)**
computes the generalized disjunctive well–founded state pair *State_Pair* of the disjunctive normal logic program *Program*.

The generalized disjunctive well–founded semantics uses two concepts for non–monotonic reasoning: the extended generalized closed–world–assumption and the negation–as–failure operator $\mathcal{F}_S^D$. This operator only derives *false* atoms. Baral [2] introduces several *generalized negation–as–failure* operators, which derive conjunctions as *false* where none of the atoms in the conjunction can be derived as *false*.

Example 16 Given the disjunctive normal logic program P, which is a subset of the program of Example 9:

q :- $\neg a$.
t :- $\neg b$.
$a \vee b$.

Baral's semantics is able to derive that the conjunction $a \wedge b$ is *false*. For this, only the third rule is considered. Then from the first and second rule, and from $\neg(a \wedge b)$ we can derive the *true* disjunction $q \vee t$. Thus Baral's semantics yields the state pair

$$(\{a \vee b, q \vee t\}, \{a \wedge b, q \wedge t\}).$$

The conjunction $a \wedge b$ neither was derivable by the extended generalized closed–world–assumption from the disjunctive transformation of P, nor by the $\mathcal{F}_S^D$–operator.

3 An Efficient Data Structure for Resolution

In the previous section, we presented the operators of DisLog for logical and non-monotonic reasoning based on Herbrand states. To represent the Herbrand states with common subclauses in DisLog, Seipel [23] has introduced the so-called clause tree. Every path from the root of a clause tree to a node represents a clause and the paths from the root to the leaves represent the Herbrand states. For instance, Figure 2 shows the clause tree for the disjunctive Herbrand state $T = \{a \vee b \vee d, a \vee b \vee e, a \vee c\}$. The disjunctive clause $a \vee b$ is a shared common subclause of $a \vee b \vee d$ and $a \vee b \vee e$.

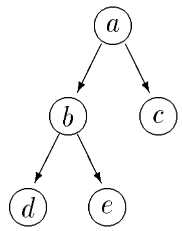

Figure 2: Clause Tree

Definition 17 (Clause Tree) Given an Herbrand state D. Let $\mathcal{A}_D$ denote the set of all atoms occurring in the clauses of D. A *clause tree* $\mathcal{T} = (V, E)$ with a node labeling $l : V \to 2^{\mathcal{A}_D}$ *represents* D if the following conditions hold:

(i) Each leaf $v \in V$ represents a clause $c(v)$ as follows: $c(v)$ is the union of the sets $l(v')$ which label the nodes v' on the path from the root to v.

(ii) The clauses $c(v)$ of the leaves $v \in V$ are in D, and every clause $C \in D$ is represented by exactly one leaf v with $c(v) = C$ of the clause tree.

Lobo, Minker and Rajasekar [18] presented another abstract data structure, the so–called *model tree*. It represents the set of all minimal models of a disjunctive Herbrand state D instead of the clauses of D as in the clause tree. Both types of trees can be used for an efficient resolution on Herbrand states, cf. [23].

Definition 18 Given a disjunctive logic program P, a disjunctive Herbrand state $T \subseteq DHB_P$ and a conjunction $E = A_1 \wedge \ldots \wedge A_k \in CHB_P$.

(i) The *state resolvent* $Res^S(T, E)$ of T w.r.t. E is defined as

$$Res^S(T, A_1 \wedge \ldots \wedge A_k) \;=\; \{\; C_1 \vee \ldots \vee C_k \mid C_i \vee A_i \in T,\ 1 \leq i \leq k \;\}.$$

(ii) The *state reduction* $Red^S(T, E)$ of T w.r.t. E is defined as

$$Red^S(T, A_1 \wedge \ldots \wedge A_k) \;=\; \{\; C \in T \mid\ \not\exists A_i, 1 \leq i \leq k :\ A_i \text{ occurs in } C \;\}.$$

Let us illustrate the state resolution and state reduction on clause trees.

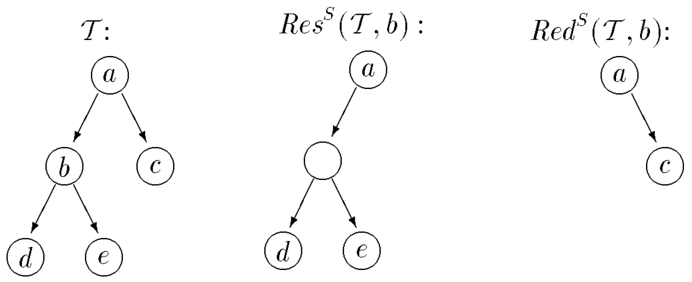

Figure 3: Tree–Resolvents and Tree–Reductions

Example 19 For the disjunctive Herbrand state $T = \{a \vee b \vee d, a \vee b \vee e, a \vee c\}$, we get $Res^S(T, b) = \{a \vee d, a \vee e\}$ and $Red^S(T, b) = \{a \vee c\}$. For the corresponding clause tree $\mathcal{T}$, we get $Res^S(\mathcal{T}, b)$ and $Red^S(\mathcal{T}, b)$ as depicted in Figure 3.

4 Evaluation Techniques

During the various derivation steps in the computation of the generalized disjunctive well–founded semantics of a disjunctive normal logic program, a sequence of monotonically growing state pairs $S = (T_S, F_S)$ is generated by alternating applications of the operators $\mathcal{S}_P^{\mathcal{E}}$ and $\mathcal{S}_P^{\mathcal{D}}$. Thus, we can apply Δ–iteration techniques for the described consequence operators. We will describe the Δ–iteration technique for the disjunctive consequence operator $\mathcal{T}_P^S$.

For deriving the minimal model state and the extended generalized closed–world–assumption of disjunctive logic programs, we can use the state resolvent and the state reduction operations.

4.1 Delta–Iteration

Semi–naive or Δ–iteration was originally defined for definite logic programs. For extending it to the fixpoint computation of a disjunctive logic program P, we need the Δ–version of the disjunctive consequence operator $\mathcal{T}_P^S$.

Definition 20 Given a disjunctive logic program P. The *disjunctive Δ–consequence operator* $\mathcal{T}_P^S : 2^{DHB_P} \times 2^{DHB_P} \to 2^{DHB_P}$ is defined as

$$\mathcal{T}_P^S(T, \Delta T) \;=\; \{\; C \vee C_1 \vee \ldots \vee C_m \mid C \text{ :- } B_1, \ldots, B_m \text{ is in } Ground(P),\; and$$
$$(\; n = 0 \;\; or \;(\; B_i \vee C_i \in T \cup \Delta T,\; 1 \le i \le m, \text{ and some } B_i \vee C_i \text{ is in } \Delta T \;)\;)\;\}.$$

The algorithm Δ–ITERATION of Figure 4 formulates the disjunctive Δ–iteration based on the Δ–version of $\mathcal{T}_P^S$. When applied to the disjunctive Herbrand state T which consists of the disjunctive facts in P, it yields the least fixpoint of $\mathcal{T}_P^S$.

```
Algorithm  Δ–ITERATION ( P, T )
{ initialization }
ΔT := T;
{ iteration }
while  ( |ΔT| > 0 )
     T1   :=  T_P^S(T, ΔT);
     T2   :=  T1 \_s T;
     ΔT   :=  can(T2);
     T    :=  can(T ∪ ΔT);
end
```

Figure 4: Disjunctive Δ–ITERATION

Experiments on the efficiency of Δ–ITERATION are described in [23].

4.2 State–Resolvents and State–Reductions

The minimal model state $\mathcal{MS}_P$ of a disjunctive logic program P can be derived by repeated computations of resolvents of derived disjunctive Herbrand states with the ground conjunctions formed by the bodies of the ground rules in $Ground(P)$, since

$$\mathcal{T}_P^S(T) \;=\; \bigcup_{C \text{:-} E \;\in\; Ground(P)} \{C\} \vee Res^S(T, E),$$

where the body E of a disjunctive rule is considered as a conjunction.

Here, $Res^S(T, E) = \bigvee_{i=1}^m Res^S(T, B_i)$ for $E = \bigwedge_{i=1}^m B_i$. The formula can be simplified by applying the state reduction operation Red^S to the state resolvents $Res^S(T, B_i)$ and the conjunctions $E_i = \bigwedge_{j=1}^{i-1} B_j \wedge \bigwedge_{j=i+1}^m B_j$:

$$\mathcal{T}_P^S(T) \;=\; T \;\cup \bigcup_{\substack{C \text{:-} B_1, \ldots, B_m \\ \in\; Ground(P)}} \{C\} \vee \bigvee_{i=1}^m Red^S(Res^S(T, B_i), E_i).$$

The extended generalized closed–world–assumption $\mathcal{EGCWA}_P$ can be derived from the resolvents $Res^S(\mathcal{MS}_P, E)$, which are called support–for–negation sets in [18], of the minimal model state $\mathcal{MS}_P$ with the conjunctions $E \in CHB_P$:

$$\mathcal{EGCWA}_P \;=\; \{\; E \in CHB_P \mid Res^S(\mathcal{MS}_P, E) \setminus_s \mathcal{MS}_P = \emptyset \;\}.$$

5 Conclusion

We have developed a system called DisLog for the efficient evaluation of disjunctive normal logic programs. Among other things, DisLog is designed as a test tool for disjunctive database applications under different semantics. DisLog is implemented as a meta–interpreter in Sicstus–Prolog.

Computing the semantics of a disjunctive normal logic program P, such as the generalized disjunctive well–founded semantics $\mathcal{GDWFS}_P$, requires the repeated derivation of the minimal model state and the extended generalized closed–world–assumption w.r.t. certain disjunctive transformations of P. The basic operators for computing minimal model states are the disjunctive consequence operator $\mathcal{T}_P^S$, the Δ–operator $\setminus_s$ with subsumption and the canonization operator *can* for disjunctive Herbrand states. Tree–based versions of these operators are applied in a Δ–iteration for computing $\mathcal{MS}_P$ and $\mathcal{EGCWA}_P$. Currently, we are working on an integration of the well–known *magic–sets* method into fixpoint–computation to answer queries posed to a disjunctive deductive database.

The minimal model state $\mathcal{MS}_P$ and the the extended generalized closed–world–assumption $\mathcal{EGCWA}_P$ of a program are *views* derived from the disjunctive facts in the database. The Δ–iteration techniques for the $\mathcal{T}_P^S$–operator and certain update formulas for state resolvents, cf. [24], allow for maintaining these views under updates to these facts.

References

[1] *A. Abdelmoty, M. Williams, N. Paton:* Deduction and Deductive Databases for Geographic Data Handling, Proc. Intl. Symposium on Advances in Spatial Databases 1993 (SSD'93), pp. 443-464.

[2] *C. Baral:* Generalized Negation As Failure and Semantics of Normal Disjunctive Logic Programs, Proc. Intl. Conference on Logic Programming and Automated Reasoning 1992 (LPAR'92), pp. 309-319.

[3] *C. Baral, J. Lobo, J. Minker:* Generalized Well–Founded Semantics for Logic Programs, Proc. Intl. Conference on Automated Deduction 1989, pp. 102-116.

[4] *F. Bancilhon, R. Ramakrishnan:* An Amateur's Introduction to Recursive Query Processing Strategies, Proc. ACM SIGMOD 1986, pp. 16-52.

[5] *C. Baral, V.S. Subrahmanian:* Dualities between Alternative Semantics for Logic Programming and Non–monotonic Reasoning, Proc. Intl. Conference on Logic Programming and Non–monotonic Reasoning 1991, pp. 69-86.

[6] *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.

[7] *W. Chen, D.S. Warren:* A Goal–Oriented Approach to Computing Well–Founded Semantics, Proc. Joint Intl. Conference and Symposium on Logic Programming 1992, pp. 589-603.

[8] *W. Chen, D.S. Warren:* Query Evaluation under the Well–Founded Semantics, Proc. ACM PODS 1993.

[9] *S.K. Das:* Deductive Databases and Logic Programming, Addison–Wesley, 1992.

[10] *J. Dix:* Classifying Semantics of Disjunctive Logic Programs, Proc. Joint Intl. Conference and Symposium on Logic Programming 1992, pp. 798-812.

[11] *U. Fuhrbach:* Computing Answers for Disjunctive Logic Programs, Proc. ILPS Workshop on Disjunctive Logic Programs 1991.

[12] *J.A. Fernández, J. Minker:* Theory and Algorithms for Disjunctive Deductive Databases, Programmirovanie J. 1993, Academy of Sciences of Russia.

[13] *M. Fitting:* The Family of Stable Models, Journal of Logic Programming, vol. 17(3,4), 1993, pp. 197-225.

[14] *M. Gelfond, V. Lifschitz:* The Stable Model Semantics for Logic Programming, Proc. Intl. Conference and Symposium on Logic Programming 1988, pp. 1070-1080.

[15] *U. Güntzer, W. Kießling, R. Bayer:* On the Evaluation of Recursion in (Deductive) Database Systems by Efficient Differential Fixpoint Iteration, Proc. Intl. Conference on Data Engineering 1987, pp. 120-129.

[16] *G. Köstler, W. Kießling, H. Thöne, U. Güntzer:* The Differential Fixpoint Operator with Subsumption, Proc. Intl. Conference on Deductive and Object–Oriented Databases 1993 (DOOD'93), pp. 35-48.

[17] *J.W. Lloyd:* Foundations of Logic Programming, Springer, second edition, 1987.

[18] *J. Lobo, J. Minker, A. Rajasekar:* Foundations of Disjunctive Logic Programming, MIT Press, 1992.

[19] *V.W. Marek, M. Truszczyński:* Nonmonotonic Logic — Context–Dependent Reasoning, Springer, 1993.

[20] *J. Pearl:* Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, 1988.

[21] *T.C. Przymusinski*: Stationary Semantics for Disjunctive Logic Programs and Deductive Databases, Proc. North–American Conference on Logic Programming 1990, pp. 42-59.

[22] *R. Reiter*: On Closed World Data Bases, In *H. Gallaire, J. Minker*: Logic and Databases, Plenum Press, 1978, pp. 55-76.

[23] *D. Seipel*: Tree–Based Fixpoint Iteration for Disjunctive Logic Programs, Proc. Workshop on Logic Programming with Incomplete Information, Intl. Symposium on Logic Programming 1993, pp. 177-189.

[24] *D. Seipel*: An Efficient Computation of the Extended Generalized Closed World Assumption by Support–for–Negation Sets, Proc. Intl. Conference on Logic Programming and Automated Reasoning 1994 (LPAR'94).

[25] *D. Seipel, H. Thöne*: An Application of Disjunctive Logic Programming with Incomplete Information, Proc. Intl. Conference on Expert Systems for Development 1994, pp. 176-181.

[26] *J.D. Ullman*: Principles of Database and Knowledge-Base Systems, Volume I,II, Computer Science Press, 1988/89.

[27] *A. Van Gelder, K.A. Ross, J.S. Schlipf*: The Well–Founded Semantics for General Logic Programs, JACM, vol. 38(3), 1991, pp. 620-650.

Appendix

A DisLog–Objects

Pretty–Printing of DisLog–Objects

There exist pretty–printers for DisLog–objects called by

dportray(*+Type*,*+Object*)
a DisLog–object *Object* of the type *Type* is pretty–printed.

where the logical disjunction '$\vee$' is shown as a semicolon ';', the logical conjunction '$\wedge$' is shown as a comma ',' and the logical negation '$\neg$' is shown as a tilde '~'.

Input/Output of DisLog–Objects

DisLog–programs can be stored in and consulted from files.

dconsult(*+Example*,*?Program*)
loads a program *Program* from the file *Example*.

dstore(*+Program*,*+Example*)
stores the program *Program* into the file *Example*.

B Sample Session with a Disjunctive Logic Program

```
| ?- dconsult(example1,Program),
     dportray(lp,Program).

Program =
{ a;d :- c.
  c :- a.
  a;b;c.
  a;e. }

yes

| ?- dconsult(example1,Program),
     mm_operator(Program,Models),
     dportray(chs,Models).

Models = { a,c. b,e. c,d,e. }

yes

| ?- dconsult(example1,Program),
     mms_operator(Program,Model_State),
     dportray(dhs,Model_State).

Model_State = { a;b;d. a;e. b;c. c;e. }

yes

| ?- dconsult(example1,Program),
     egcwa_operator(Program,Egcwa),
     dportray(chs,Egcwa).

Egcwa = { a,b. a,d. a,e. b,c. b,d. }

yes
```

C Sample Session with a Disjunctive Normal Logic Program

```
| ?- dconsult(example2,Program),
     dportray(lp,Program).

Program =
{ e :- p, ~f.
  f :- ~e.
  p :- q,t.
  q :- ~a.
  t :- ~b.
  a;b. }

yes

| ?- dconsult(example2,Program),
     dwfs_operator(Program,State_Pair),
     dportray(sp,State_Pair).

State_Pair = ( { a;b. }, { } )

yes

| ?- dconsult(example2,Program),
     gdwfs_operator(Program,State_Pair),
     dportray(sp,State_Pair).

State_Pair = ( { a;b. a;q. b;t. f. }, { a,q. b,t. e. p. q,t. } )

yes
```