DisLog – A Disjunctive Deductive Database Prototype

Dietmar Seipel

University of Würzburg
Am Hubland, D - 97074 Würzburg
seipel@informatik.uni-wuerzburg.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 database. The non–monotonic operators are semantically founded on generalizations of the well–known closed–world–assumption and the negation–as–failure concept.

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 *clause tree* is used as a data structure that allows for an efficient and transparent evaluation.

The DisLog–system has been developed in Prolog – currently a core part of DisLog is reimplemented in C++. DisLog offers two different types of user interfaces. The user–friendly interfaces allow for quickly experimenting with semantics on example databases – which can even be done remotely through the WWW. The DisLog–library provides various operators that can be loaded into Prolog applications.

1 Introduction

Recently, advanced applications of deductive databases have been investigated which require the extensive analysis of *incomplete information* in the form of *disjunctions* in the data, e.g. diagnostic reasoning, spatial databases (i.e. databases for geometric data), or in probabilistic reasoning, where for instance the analysis of independencies can be done by dealing with disjunctive information. Currently several groups are in the process of building systems for reasoning with disjunctions, cf. for instance [1, 2, 7].

A *disjunctive deductive database* consists of a set of logical deduction rules with disjunctive rule heads, and a set of disjunctive facts. Depending on the use of disjunction and negation, different types of *semantical concepts* are appropriate for reasoning with the database. Like the use of negation in deductive databases, the use of disjunction leads to databases that usually have several models intended by the user. For disjunctive databases without negation, the *minimal models* correctly reflect the user's intended meaning of the database (cf. [5]). In general, other types of semantics have to be used for assigning a proper meaning to a database: The *perfect models* are appropriate for databases with a stratified use of negation. For non–stratified deductive databases, perfect models have been generalized to *stable models* (cf. [3, 6]) – which form the most prominent approach – and partial stable models (cf. [6]), and in several ways to *disjunctive well-founded semantics* (cf. [11, 5]).

The DISLOG–system has been developed as a prototype for reasoning with disjunctive deductive databases. DISLOG supports both model–based semantics and well–founded semantics. It can be used for *studying optimization techniques* for existing methods of efficient reasoning, as well as for gaining *experience* with the nature of different *semantical concepts*. The main part of the DISLOG–system is the DISLOG–library, which is a collection of modules offering various operators dealing with disjunctive databases (cf. [8]). These operators can be loaded into arbitrary PROLOG applications, and thus they can be used as components of other systems. For processing disjunctive databases under various semantics, the DISLOG–system offers several user–friendly interfaces (UFI), which have been built upon the DISLOG–library. The UFI's are good for quickly experimenting with different semantics on different databases – a collection of database examples is coming with the DISLOG system. DISLOG can be accessed even remotely through a special WWW–UFI.

The rest of the paper is organized as follows: In Section 2 we will give a motivating example for disjunctive deductive databases, and in Section 3 we will introduce some basic definitions and notations. Finally, Section 4 gives a flavor of two of DISLOG's interfaces for working with disjunctive deductive databases: a user–friendly interface and the DISLOG–library.

2 A Motivating Example from Medical Diagnosis

As an example we will consider the following disjunctive deductive database for *medical diagnosis*, which deals with the pair of genes that determine a persons blood–group. The basic knowledge is given for the predicates $person$, $parent$, and $equal$. Using the disjunctive rules of the database, which are given below, other information about the persons can be derived.

Every person has got two genotypes, which may be identical. A statement $genotype(P, T)$ means that the person P has the genotype T, where – according to the first rule – the genotype must be one of "a", "b", or "o". The predicates $homozygot$ and $heterozygot$ tell if a person has got two identical or two different genotypes, respectively. One of the genotypes is inherited from the mother, the other is inherited from the father. The rules two, three, and four relate the genotype of a person to the genotypes of the parents. Finally, the $phenotype$, i.e. the person's actual blood–group, is a result of the two genotypes. According to the first rule for $phenotype$, four different phenotypes are possible: "a", "b", "ab", and "o". The rules for determining phenotype "a" are given below.

$$
\begin{aligned}
&genotype(P, a) \vee genotype(P, b) \vee genotype(P, o) \leftarrow person(P).\\
&genotype(Child, T_1) \vee genotype(Child, T_2) \leftarrow parent(Child, Parent) \wedge\\
&\qquad genotype(Parent, T_1) \wedge genotype(Parent, T_2) \wedge not\, equal(T_1, T_2).\\
&genotype(Parent, T_1) \vee genotype(Parent, T_2) \leftarrow parent(Child, Parent) \wedge\\
&\qquad genotype(Child, T_1) \wedge genotype(Child, T_2) \wedge not\, equal(T_1, T_2).\\
&genotype(Child, T) \leftarrow parent(Child, Parent) \wedge\\
&\qquad genotype(Parent, T) \wedge homozygot(Parent).
\end{aligned}
$$

$$heterozygot(P) \leftarrow genotype(P, T_1) \wedge genotype(P, T_2) \wedge not\, equal(T_1, T_2).$$
$$homozygot(P) \leftarrow person(P) \wedge not\, heterozygot(P).$$

$$phenotype(P, a) \vee phenotype(P, b) \vee phenotype(P, ab) \vee phenotype(P, o) \leftarrow person(P).$$
$$phenotype(P, a) \leftarrow genotype(P, a) \wedge genotype(P, o).$$
$$phenotype(P, a) \leftarrow genotype(P, a) \wedge homozygot(P).$$

This disjunctive database is non–stratified,[1] and we would have to draw conclusions from the basic facts and the disjunctive rules by stable model semantics or well–founded semantics.

3 Definitions and Notations

A *disjunctive deductive database* $\mathcal{D}$ consists of logical deduction rules of the form

$$r = A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not C_1 \wedge \ldots \wedge not C_n, \quad (1)$$

where A_i, $1 \leq i \leq k$, B_i, $1 \leq i \leq m$, and C_i, $1 \leq i \leq n$, are positive atoms in a given first order language $\mathcal{L}$, and $k, m, n \in \mathbb{N}_0$. not is the negation–by–default operator. A rule is called a fact if $m = n = 0$. The set of all *ground instances* of the rules in $\mathcal{D}$ is denoted by $gnd(\mathcal{D})$. A database is called *positive–disjunctive* if its rules do not contain any default negation (i.e. $n = 0$).

The result of a DISLOG–operation can be either a set of ground disjunctions, a so–called Herbrand state, or a set of Herbrand interpretations.

(i) The *disjunctive Herbrand base* $D\mathit{HB}_\mathcal{D}$, the *negative Herbrand base* $N\mathit{HB}_\mathcal{D}$, and the *general Herbrand base* $G\mathit{HB}_\mathcal{D}$ of $\mathcal{D}$ are given by

$$\begin{aligned} D\mathit{HB}_\mathcal{D} &= \{ A_1 \vee \ldots \vee A_k \mid A_i \in \mathit{HB}_\mathcal{D},\ 1 \leq i \leq k,\ k \in \mathbb{N}_0 \}, \\ N\mathit{HB}_\mathcal{D} &= \{ \neg A_1 \vee \ldots \vee \neg A_k \mid A_i \in \mathit{HB}_\mathcal{D},\ 1 \leq i \leq k,\ k \in \mathbb{N}_0 \}, \\ G\mathit{HB}_\mathcal{D} &= \{ L_1 \vee \ldots \vee L_k \mid L_i \in \mathit{HB}_\mathcal{D} \cup \neg \mathit{HB}_\mathcal{D},\ 1 \leq i \leq k,\ k \in \mathbb{N}_0 \}. \end{aligned}$$

(ii) An *Herbrand state* is a set S of formulas. It is called *disjunctive*, *negative*, and *general*, if $S \subseteq D\mathit{HB}_\mathcal{D}$, $S \subseteq N\mathit{HB}_\mathcal{D}$, and $S \subseteq G\mathit{HB}_\mathcal{D}$, respectively.

For model–based semantics, DISLOG can compute the set $\mathcal{I}$ of intended models of a database, and it can also derive the Herbrand state that corresponds to $\mathcal{I}$ by means of *sceptical reasoning*. Sceptical reasoning with a set $\mathcal{I}$ of Herbrand interpretations consists in deriving the Herbrand state $\mathit{STATE}(S, \mathcal{I})$ of all disjunctions C in a given universe $S \subseteq G\mathit{HB}_\mathcal{D}$ which are satisfied by all interpretations in $\mathcal{I}$:[2]

$$\mathit{STATE}(S, \mathcal{I}) = \{ C \in S \mid \forall I \in \mathcal{I} : I \models C \}.$$

[1] $genotype$ depends recursively on itself, and there is a negative dependency between $homozygot$ and $heterozygot$ involved in the recursion.

[2] The general Herbrand states $\mathit{STATE}(S, \mathcal{I})$ are S–expanded, since with a general disjunction also all disjunctions that are subsumed by it are satisfied by the Herbrand interpretations, and thus are elements of the states as well.

For the set $\mathcal{MM}_\mathcal{D}$ of *minimal models* of a database, we can derive the well–known concepts of the *minimal model state* $MS_\mathcal{D} = \mathit{STATE}\,(\mathit{DHB}_\mathcal{D}, \mathcal{MM}_\mathcal{D})$ and the *extended generalized closed–world–assumption* $\mathit{EGCWA}_\mathcal{D} = \mathit{STATE}\,(\mathit{NHB}_\mathcal{D}, \mathcal{MM}_\mathcal{D})$, which is a set of *non–monotonic* consequences. E.g. for the database $\mathcal{D}_1 = \{\, c \leftarrow a,\ a \vee b \,\}$ and its set $\mathcal{MM}_{\mathcal{D}_1} = \{\, \{\, a, c \,\}, \{\, b \,\} \,\}$ of minimal models we derive $MS_{\mathcal{D}_1} = \{\, a \vee b,\ b \vee c,\ a \vee b \vee c \,\}$ and $\mathit{EGCWA}_{\mathcal{D}_1} = \{\, \neg a \vee \neg b,\ \neg b \vee \neg c,\ \neg a \vee \neg b \vee \neg c \,\}$.

The *Gelfond–Lifschitz transformation* of a disjunctive database $\mathcal{D}$ w.r.t. an Herbrand interpretation M is obtained from the ground instance $gnd\,(\mathcal{D})$ of $\mathcal{D}$ by removing in the rules all default negated body literals "$not\,A$" such that A is false in M, and subsequently removing all rules from the database which have default negated body literals "$not\,A$" such that A is true in M. M is called a *stable model* of $\mathcal{D}$ if M is a minimal model of $\mathcal{D}^M$ (cf. [3]). The set of all stable models of $\mathcal{D}$ is denoted by $\mathit{STABLE}(\mathcal{D})$. E.g. the database $\mathcal{D}_2 = \{\, a \leftarrow not\,b,\ b \leftarrow not\,a,\ b \vee c \,\}$ has got two stable models, i.e. $\mathit{STABLE}(\mathcal{D}_2) = \{\, \{\, a, c \,\}, \{\, b \,\} \,\}$. The Herbrand interpretation $M = \{\, a, c \,\}$ is a stable model, since M is one of the two minimal models of $\mathcal{D}_2^M = \{\, a,\ b \vee c \,\}$ – the other minimal model is $\{\, a, b \,\}$.

4 The Interfaces to DISLOG

For reasoning with disjunctive databases the DISLOG–system has got several interfaces. The DISLOG–UFI and the DISLOG–library are two PROLOG interfaces. They are invoked by consulting the DISLOG sources in the PROLOG interpreter.[3] The operators in the DISLOG–UFI can be used for easily browsing through different semantics for a given database. The DISLOG–library provides a wide range of predicates on disjunctive databases and Herbrand states, often giving several alternative implementations for the same (abstract) operator.

There exist two additional user–friendly interfaces, which are not described in this paper: one is offered on the WWW, cf. [8], and another interface that is based on Tcl/Tk.

4.1 The DISLOG–UFI

The DISLOG–UFI is a user–friendly interface using the PROLOG interpreter. It allows for evaluating disjunctive databases under various semantics. For all semantics of DISLOG there exists the following common predicate *semantics*:

semantics(?Sem,?Res,+DB_Name) $\leftarrow$
Computes the semantics *Sem* for the database *DB_Name*, and pretty–prints the result, which is of the type *Res*.

semantics $\leftarrow$
Lists the available pairs of *Sem* and *Res*, where *Sem* is one of *minimal*, *perfect*, *stable*, *possible*, *wfs*, *bdwfs*, *gdwfs*, and *Res* is one of *dhb*, *nhb*, *ghb*, *hi*.

For model–based semantics, i.e. for *Sem* $\in \{\, minimal, perfect, stable, possible \,\}$, the results of applying *semantics/3* to a database are the Herbrand states $\mathit{STATE}\,(S, \mathcal{I})$ obtained by sceptical reasoning, where $\mathcal{I}$ is the corresponding set of *Sem*–models, and S is the Herbrand state indicated by

[3] In SWI–PROLOG this can be done by calling "*prolog -f sources/dislog*". It is advised to call PROLOG from a directory, that contains – amoung others – the subdirectories *examples*, *manuals*, *results*, and *sources*. These subdirectories contain the example databases, the DISLOG library manual, temporary files, and the DISLOG source files, respectively.

Res $\in \{$ *dhb*, *nhb*, *ghb* $\}$. In the case *Res* $=$ *hi* the result is the set $\mathcal{I}$ of models itself. For well–founded semantics, DISLOG primarily derives an Herbrand state of ground disjunctions, but it can also compute the corresponding set of minimal Herbrand models of the derived state.

Databases can be stored permanently in files. For this purpose, DISLOG uses a PROLOG–style syntax, where the negation–by–default operator not is denoted by the tilde "~".[4] The following predicates access the *data dictionary* of DISLOG, and give information on the *databases* that are stored in the current *examples* directory of the DISLOG system:

databases(+Pattern) $\leftarrow$
Lists the names of the available databases that match the regular expression *Pattern*.

databases $\leftarrow$
Lists the names of the available databases.

Also the numbers of facts, disjunctive rules, and disjunctive rules with negation, respectively, are given for each of the listed databases. E.g. the call *databases('dd* c*')* lists information on all databases whose names start with "*dd*" or with "*c*". Assume that the file "*ddb3*" contains the disjunctive database $\mathcal{D}_3 = \{\, d \leftarrow a \wedge c,\ a \leftarrow not\, b,\ b \leftarrow not\, a,\ b \vee c \,\}$. The predicate *show_database* pretty–prints a database in a notation that is also used for permanently storing databases in files:

```
?– show_database(ddb3).

d :- a,c.   a :- ~b.   b :- ~a.   b;c.

yes
```

In the following we will explain two options of *reasoning* with a database *under stable model semantics*. We show how the set $\mathit{STABLE}(\mathcal{D}_3) = \{\, \{\, a, c, d \,\}, \{\, b \,\} \,\}$ of *stable models* can be derived, and how we can also derive the corresponding general Herbrand state $\mathit{STATE}\,(\mathit{GHB}_{\mathcal{D}}, \mathit{STABLE}(\mathcal{D}_3)) = \{\, a \vee b,\ b \vee c,\ b \vee d,\ \neg a \vee \neg b,\ \neg b \vee \neg c,\ \neg b \vee \neg d,\ a \vee \neg c,\ a \vee \neg d,\ c \vee \neg d \,\}$, where redundant disjunctions like "$a \vee b \vee c$" or "$a \vee \neg a$" are left out:

```
?– semantics(stable,hi,ddb3).

Semantics = { a,c,d. b. }

yes
```

```
?– semantics(stable,ghb,ddb3).

Semantics = {  a;b. b;c. b;d.  ~a;~b. ~b;~c. ~b;~d.  a;~c. a;~d. c;~d.  }

yes
```

The predicate *semantics* can also be called with either the first or the second argument or both being free variables. In this case DISLOG allows for browsing through all possible semantics and result types that are currently implemented within the system.

[4] Note that in a file a disjunctive database must be preceded by the directive "*:- disjunctive.*" and followed by the directive "*:- prolog.*".

4.2 The DisLog–Library

The DisLog–library offers serveral modules containing a variety of operators for dealing with Herbrand states and databases, and also for computing many of the semantics known from literature. There are modules for the following:

> 1) Herbrand States.[5] 2) Program Transformations. 3) Disjunctive Consequence Operators. 4) Model–Based Semantics. 5) Well–Founded Semantics. 6) Input and Output.

In the following we will briefly talk about 3) to 6). The DisLog library manual, cf. [8], provides a detailed description of all operators with corresponding examples. This manual can also be accessed online by calling *dman* in the Prolog interfaces.

For all operators of the DisLog–library, the arguments are *called by value*, using the following list representation of a rule r of the form (1):

$$[A_1,\ldots,A_k] - [B_1,\ldots,B_m] - [C_1,\ldots,C_n]$$

For the special case of a disjunctive fact we can just write $[A_1,\ldots,A_k]$, i.e. the empty lists for the empty positive and negative body can be left out.

Disjunctive Consequence Operators

For positive–disjunctive databases the concepts of state generation and of model generation are implemented by various disjunctive consequence operators, cf. [9]. The well–known hyperresolution operator $\mathcal{T}^s_{\mathcal{D}}$ on disjunctive Herbrand states S, that is defined by

$$\begin{aligned}\mathcal{T}^s_{\mathcal{D}}(S) = \{\, & \alpha \vee \alpha_1 \vee \ldots \vee \alpha_m \mid \alpha, \alpha_1, \ldots, \alpha_m \in \mathit{DHB}_{\mathcal{D}} \text{ and there is a rule} \\ & \alpha \leftarrow B_1 \wedge \ldots \wedge B_m \in \mathit{gnd}\,(\mathcal{D}) :\ B_i \vee \alpha_i \in S \text{ for } 1 \leq i \leq m \,\},\end{aligned}$$

is incorporated in DisLog as follows:

tps_operator(+Database,+State1,–State2) $\leftarrow$
: Computes the disjunctive Herbrand state *State2* by applying the $\mathcal{T}^s_{\mathcal{D}}$–operator of a database *Database* to an Herbrand state *State1*.

tps_fixpoint_iteration(+Database,–State) $\leftarrow$
: Computes the least fixpoint *State* of the $\mathcal{T}^s_{\mathcal{D}}$–operator of a database *Database* and outputs the intermediate disjunctive Herbrand states.

The notation for the $\mathcal{T}^i_{\mathcal{D}}$–operators *tpi_operator* and *tpi_fixpoint_iteration* for minimal model generation is analogous; the latter operators deal with sets of Herbrand interpretations.

Below we show how the operator *tps_fixpoint_iteration* computes the least fixpoint of the $\mathcal{T}^s_{\mathcal{D}}$–operator of a positive-disjunctive database and displays the intermediate disjunctive Herbrand states:

[5] Including the data structure of clause trees.

?– *tps_fixpoint_iteration(* [[*p(X,Y)*]–[*a(X,Z),p(Z,Y)*],
[*p(X,Y)*]–[*a(X,Y)*], [*a(a,b),a(a,c)*], [*a(b,d)*], [*a(c,d)*]], *State*).

tps_iteration 1: { a(a,b);a(a,c). a(b,d). a(c,d). }

tps_iteration 2: { a(a,b);p(a,c). a(a,c);p(a,b). p(b,d). p(c,d). }

tps_iteration 3: { a(a,b);p(a,d). a(a,c);p(a,d).
p(a,b);p(a,c). p(a,b);p(a,d). p(a,c);p(a,d). }

tps_iteration 4: { p(a,d). }

tps_iteration 5: { }

State = [[a(a,b),a(a,c)], [a(a,b),p(a,c)], [a(a,c),p(a,b)],
[a(b,d)], [a(c,d)], [p(a,b),p(a,c)], [p(a,d)], [p(b,d)], [p(c,d)]]

yes

Model–Based and Well–Founded Semantics

For disjunctive databases most commonly used model–based semantics are implemented, as well as the well–founded semantics of Ross and van Gelder for normal databases without disjunctions [11], and its generalizations to the disjunctive case by Minker and his group [5]:

bdwfs_operator(+Database,-State) ←
Computes the disjunctive well–founded semantics of the database *Database* according to Baral. The result is a general Herbrand state *State*.

For many operators several alternative implementations exist. A complete summary can be found in the DISLOG library manual [8]. Again, the operators for semantics are summarized in a single predicate. The call "*semantics(+Sem,+Res,+Database,-Result)*" computes the semantics *Sem* for the database *Database* and yields a result *Result* of the type *Res*, where *Sem* is one of *minimal*, *perfect*, *stable*, *possible*, *wfs*, *bdwfs*, *gdwfs*, and *Res* is one of *dhb*, *nhb*, *ghb*, *hi*.

Input and Output

There are predicates for loading, storing – from the current examples directory – and pretty–printing DISLOG–objects like Herbrand states, databases, clause trees, etc..

load_database(+DB_Name,–Database) ←
Loads the database *Database* with the name +*DB_Name*.

store_database(+DB_Name,+Database) ←
Stores the database *Database* under the name +*DB_Name*.

dportray(+Mode,+DB_Object) ←
Pretty–prints the DISLOG–object +*DB_Object* in the mode +*Mode*.

E.g. the predicate *show_database* of the DISLOG–UFI is implemented by loading the database with *load_database*, and then pretty-printing it with *dportray* as *DB_Object* with "*Mode = lp*".

5 Final Remarks

In DisLog efficient *reasoning algorithms* have been developed for the most important consequence operators in disjunctive deductive databases. Especially the evaluation of non–monotonic operators can be very complex. The optimizations try to improve the efficiency of each single execution of an operator, as well as its fixpoint computation (e.g. by Δ–iteration techniques).

For representing sets of disjunctive ground clauses a compact data structure called *clause tree* has been used. Based on the clause tree representation of disjunctive information, heuristics for the efficient state generation and model generation have been developed.

The DisLog system is frequently extended by new concepts of non–monotonic reasoning. For ensuring the correctness of the system after modifications, a *test tool* has been built, which can test a collection of operators with a set of critical test examples that have been collected so far.

Recently, a new semantics has been developed, called *(partial) evidential stable model semantics* [10], that extends the original (partial) stable model semantics. For small inconsistent parts of a database, where no (partial) stable models exist, these new semantics select suitable (partial) minimal models as backup models.

References

[1] *C. Aravindan, J. Dix, I. Niemelä:* DisLoP: Towards a Disjunctive Logic Programming System, Proc. Fourth Intl. Conf. on Logic Programming an Non–Monotonic Reasoning (LPNMR'97), Springer LNAI 1265, 1997, pp. 341–352.

[2] *T. Eiter, N. Leone, C. Mateis, G. Pfeifer, F. Scarcello:* A Deductive System for Non–Monotonic Reasoning, Proc. Fourth Intl. Conf. on Logic Programming an Non–Monotonic Reasoning (LPNMR'97), Springer LNAI 1265, 1997, pp. 363–374.

[3] *M. Gelfond, V. Lifschitz:* Classical Negation in Logic Programs and Disjunctive Databases, New Generation Computing, vol. 9, 1991, pp. 365–385.

[4] *J.W. Lloyd:* Foundations of Logic Programming, 2nd edition, Springer, 1987.

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

[6] *T.C. Przymusinski*: Stable Semantics for Disjunctive Programs, New Generation Computing, vol. 9, 1991, pp. 401–424.

[7] *D. Seipel, H. Thöne*: DisLog – A System for Reasoning in Disjunctive Deductive Databases, Proc. Intl. Workshop on the Deductive Approach to Information Systems and Databases, 1994, pp. 325–343.

[8] *D. Seipel*: The DisLog Library Manual, University Tübingen, 1995.
The DisLog Homepage: "http://www-info1.informatik.uni-wuerzburg.de/databases/DisLog".

[9] *D. Seipel, J. Minker, C. Ruiz*: Model Generation and State Generation for Disjunctive Logic Programs, Journal of Logic Programming, vol. 32(1), 1997, pp. 48–69.

[10] *D. Seipel:* Partial Evidential Stable Models For Disjunctive Deductive Databases, to appear in: Proc. Workshop on Logic Programming and Knowledge Representation (LPKR'97) at the International Symposium on Logic Programming 1997 (ILPS'97), 1997.

[11] *A. Van Gelder, K.A. Ross, J.S. Schlipf:* Unfounded Sets and Well–Founded Semantics for General Logic Programs, Proc. Seventh ACM Symposium on Principles of Database Systems (PODS'88), pp. 221–230.