Disjunctive Deductive Databases

By

Hwang Hoi Yee Cothan

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Philosophy

(Computer Science and Engineering)

at the

CHINESE UNIVERSITY OF HONG KONG

JUNE 1996

Abstract

In this thesis, we introduce the data-disjunctive logic programs and concentrate on the data-disjunction. We define the model-theoretic semantics and fixpoint semantics of the data-disjunctive logic program. The minimal data-disjunctive model state is presented which contains the definite and indefinite information of the data-disjunctive logic program. The contingency model is defined which stores the uncertainty information about the atoms. We develop a bottom-up evaluation method to compute the model of this logic program and show how to use the computed result to answer query. We prove that any disjunctive deductive database can be transformed to a data-disjunctive deductive database and show that the expressive power of the data-disjunctive logic program is the same as the disjunctive logic program. The concept of standard-form facts is presented and with this concept, a disjunctive deductive database can be transformed into the corresponding data-disjunctive deductive database where the bottom-up evaluation method can be applied. A rule rewriting technique similar to the magic sets techniques is proposed to solve query for the data-disjunctive deductive databases. With this technique, it becomes possible to constrain the computations to the relevant information in the database for a given query. We have built a prototype for performance evaluation of the bottom-up evaluation procedure and the rule rewriting method and we find that the procedure using the rule rewriting method can solve a query efficiently.

Acknowledgement

I want to thank Prof. Ada Fu and Prof. Ho-Fung Leung for their kindness and encouragement in my two years of postgraduate study. Without their care and direction, I am certain that this thesis would not have been completed. I am extremely fortunate to have been their student.

I thank my thesis committee, Prof. Jimmy Lee, Prof. Man-Hon Wong for being my markers and giving comments in my thesis.

Thank go to all my friends in the department, Terry Lau for lending references and helping me in using the Unix system. Edward Tam for his good resource of valuable CD-ROMs. Alice, Frank and Henry for the kindness in listening my boring presentations throughout the two years. Special thank to the Terry and Phyllis for their aid in using the Latex. I also like to thank Johnson, Ah Shun, Tin Si and the members of the logic programming group Wan-Lung Wong, Hon-Wing Won and Chung-Kan Chiu for their encouragement and joyfulness.

I am deeply grateful to my family for their love and support during the past two years. Lastly, I must thank for my sister and sister-in-law who offer me a room in their house and encouraging me throughout my six-years of graduate study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Objectives of the Thesis

The study of disjunctive databases started around early 80's [26]. Minker and Grant [27] first address the problem of computing answers to queries in disjunctive databases. Since then different methods and procedures have been presented [5, 16, 19, 36, 37]. Different data structures have been proposed to represent indefinite information [18, 19] and a data structure called minimal model tree is introduced in [11] as a structure-sharing approach to represent indefinite information which captures the semantics of disjunctive deductive databases.

In this thesis, we consider disjunctive deductive databases, which do not contain negative literals in the body of rules. We develop a new model semantics for the disjunctive logic program with the concept of **data-disjunction** and **data-disjunctive logic program**. We define the model-theoretic and fixpoint semantics of this kind of disjunctive logic program. In our model, definite information and indefinite information can be deduced.

Consider the program

$$Go(swim) \vee Go(barbecue).$$
$$Go(swim).$$

Under the generalized closed world assumption (GCWA) [26] and the extended generalized closed world assumption (EGCWA) [36], the atom $Go(barbecue)$ is assumed false. But

from the program, we know that it is possible that the person go to barbecue. This information about uncertainty can be viewed as the *maybe information* which was first addressed in [19] but the paper only concerns about maybe information in relational databases. As we want to be able to give such information about uncertainty in the disjunctive deductive databases and use it to answer query, our new model contains a **contingency model** which expresses this kind of information. The contingency model can be viewed as the union of the set of true ground atoms and the set of possibly true ground atoms defined by the Possible World Semantics (PWS) [6] and we deduce an efficient algorithm to compute this contingency model.

A data-disjunction is a disjunction of atoms with the same predicate. This kind of disjunction is a general form of *restricted* disjunction which has been argued to occur more often in practice [17]. A data-disjunctive clause is a clause of the form

$$A(x_1) \vee \cdots \vee A(x_n) \leftarrow B(y_1), \ldots, B(y_m).$$

where the atoms at the head of the clause have the same predicate. A data-disjunctive deductive database consists of a set of data-disjunctive clauses.

The model of the data-disjunctive logic program, which is called the **minimal data-disjunctive model** MM_P^{DD}, consists of two components: The **minimal data-disjunctive model state** and the contingency model. The minimal data-disjunctive model state contains the logical consequences with a single predicate of the program.

Another reason for introducing data-disjunction is for efficiency in query evaluation. Consider the program

$Bus(1) \leftarrow Go(swim).$
$Bus(2) \leftarrow Go(barbecue).$
$Go(swim) \vee Go(barbecue).$

where the first two clauses indicate the bus to be taken to go to a certain place, and the last clause states that one can either go to swim or barbecue. There are simply $2^2 = 4$

logical consequences: [1]

1. $Go(swim) \vee Go(barbecue)$.
2. $Go(swim) \vee Bus(2)$.
3. $Bus(1) \vee Go(barbecue)$.
4. $Bus(1) \vee Bus(2)$.

The minimal data-disjunctive model state of the program only contains the data-disjunctions $Go(swim) \vee Go(barbecue)$ and $Bus(1) \vee Bus(2)$. In general, for the data-disjunctive logic program, only disjunctions with a single predicate are concerned.

According to the expectation of the user, a program can be transformed to another program such that the minimal data-disjunctive model state of transformed program contains those disjunctions corresponding to the disjunction with more than one predicate in the original program.

Take the last example on the relation Bus and Go. The program can be transformed to:[2]

$T(bus, 1) \leftarrow T(go, swim)$.
$T(bus, 2) \leftarrow T(go, barbecue)$.
$T(go, swim) \vee T(go, barbecue)$.

where in the transformed program, $T(bus, x)$, $T(go, y)$ corresponds to $Bus(x)$, $Go(y)$ of the original program respectively. Then the minimal data-disjunctive model state of the transformed program is

$\{\ T(go, swim) \vee T(go, barbecue), T(go, swim) \vee T(bus, 2),$
$T(bus, 1) \vee T(go, barbecue), T(bus, 1) \vee T(bus, 2)\ \}$

where each disjunction in this model corresponds to a logical consequence of the original program.

We introduce the concept of standard-form fact, which provides a strategy to transform the original program to a data-disjunctive program. The concept of standard-form fact

[1] Only those ground clauses not subsumed by other logical consequence are counted
[2] detail of transformation can be found in chapter 5

is used in two different ways. Firstly, it is used for the transformation of a non-data-disjunctive deductive database to a data-disjunctive deductive database. For example, for the program

$$Father(X,Y) \vee Mother(X,Y) \leftarrow Parent(X,Y).$$

A fact of the form $Father(\cdot) \vee Mother(\cdot)$ will be a standard-form fact. The program will be transformed where a new predicate, say S, is introduced such that a standard form fact is corresponding to a data-disjunction in the transformed program. The transformed program will be

$$Father(X,Y) \leftarrow S(X,Y,1).$$
$$Mother(X,Y) \leftarrow S(X,Y,2).$$
$$S(X,Y,1) \vee S(X,Y,2) \leftarrow Parent(X,Y).$$

Moreover, the concept of standard form fact can be used by the user to indicate which kind of disjunction is useful and meaningful. Consider the previous program on Bus and Go. If the user expect that the information of the form $Bus(x) \vee Go(y)$ is useful, he can insert the clause

$$Bus(\eta) \vee Go(\eta).^3$$

into the original program. With this clause, the fact $Bus(\cdot) \vee Go(\cdot)$ will be in standard-form and the system will transform the program to the mentioned program on relation T consequently.

In disjunctive logic program, we expect the answer to a query is not always a simple 'yes' or 'no' form. Consider the program on Bus and Go. Under the traditional definition of answer [1, 11], the set of answers for a query Q is defined to be

$$\{\ A_1 \vee \cdots \vee A_k : k > 0 \ and\ M \models A_1 \vee \cdots \vee A_k \ and\ \forall i, A_i \Rightarrow \exists Q\ \}$$

If the query ?-Go(swim) is given, the query reduces to a 'yes' or 'no' question as only $Go(swim)$ implies $Go(swim)$. Although $Go(swim)$ is not a logical consequence of program, we do not want to give a simple answer 'no' to the query since we know that

[3] η is a default constant.

the person probably go to swim. The answer to the query should at least express this uncertainty. For example, an answer can be

Probable: $Go(swim)$

Alternatively, in a probably better way, the answer could express relevant knowledge to the query, for example, the answer can be

$Go(swim) \vee Go(barbecue)$

which expresses more information than the former case.

In the scope of data-disjunctive deductive databases, we define the answer for a query Q, which we call the **relevant answer set**, to be the set of ground clause

$$\{A(x_1) \vee \cdots \vee A(x_k) \in M \mid \exists i,\ A(x_i) \Rightarrow \exists Q\ \}$$

where M is the minimal data-disjunctive model state. With this definition, the query ?-$Go(swim)$ yields the answer

$Go(swim) \vee Go(barbecue)$.

Consider another query ?-$Bus(1)$ for the program. Under the traditional definition of answer, the answer for this query is an empty set which means that $Bus(1)$ is not a logical consequence of the program. However, the answer $Bus(1) \vee Bus(2)$ will be given under the new definition.

One can see that all tuples in the answer set under the new definition are data-disjunctions. There are cases when the user expect the answer contain some logical consequences which are disjunctions of more than one predicate. For example, the user may find that the answer $Bus(1) \vee Go(barbecue)$ is useful when the query $Bus(1)$ is posted. In this case, the program should be first transformed to the program with predicate T mentioned above and then $Bus(1) \vee Go(barbecue)$ will constitute an answer to the query as the corresponding disjunction $T(bus, 1) \vee T(go, barbecue)$ is in the minimal data-disjunctive model state of the transformed program.

The set of minimal models are used to characterize the semantics of a disjunctive logic program in the GCWA [26] and the model tree method [10, 11]. For the model tree method, each branch of the model tree is one-to-one corresponding to a minimal model of the disjunctive program.

Consider the program

$$B(1) \vee B(2) \vee \cdots \vee B(m).$$
$$B(m+1) \vee B(m+2) \vee \cdots \vee B(2m).$$
$$\cdots$$
$$B(m(n-1)+1) \vee \cdots \vee B(nm).$$

With these n disjunctions, the program will have m^n minimal models. In general, the number of minimal models for a program can be exponential with the number of disjunctions in the program. Hence, it may not be practical to use these semantics, which are characterized by the set of minimal models, in solving a query.

We propose a data-structure, which is called the **derived term**, to represent the indefinite information of the data-disjunctive deductive databases. With this data-structure, we propose a bottom-up evaluation method that can compute the minimal data-disjunctive model state of the program. According to the expectation of the user, a program is transformed to another program such that the evaluation method can be applied and answer is given consequently.

With the presented bottom-up evaluation procedure for the data-disjunctive deductive databases, a rule rewriting method similar to the magic set rewriting method is developed which can be applied to this data-disjunctive deductive database. With this rule rewriting method, the answer for a given query can be obtained in a guided manner.

We have built a prototype for performance evaluation of the proposed bottom-up evaluation procedure and the rule rewriting method for the data-disjunctive deductive databases. We find that, in solving a query, the procedure using the rule rewriting method is much more efficient than the procedure that do not. We also compare our bottom-up evaluation

procedure with the model tree method [10]. We find that, for the tested program, the proposed methods are much more efficient than the model tree method.

1.2 Overview of the Thesis

The thesis is organized as follows: A review of deductive databases and disjunctive deductive database is given in chapter 2. In chapter 3, the notion of data-disjunctive logic program is introduced and we define the model of this data-disjunctive logic program and the model-theoretic and fixpoint semantics in chapter 4. In chapter 5, we prove that any disjunctive deductive database can be transformed into a data-disjunctive deductive database, and hence we can use those operational procedures to compute the logical consequences of the disjunctive deductive database and use them for query answering. Rule rewriting algorithm, which is similar to the idea of magic set on deductive databases, is presented in chapter 6. Experimental results will be given in chapter 7 which is followed by the conclusions in chapter 8.

Chapter 2

Background and Related Work

This chapter gives a review of the field of deductive databases together with an overview of the development in disjunctive deductive databases. As state in [9], the field of deductive databases is generally considered to be started in 1978 when the book *Logic and Data Bases*, edited by Gallaire and Minker, was published. The field of disjunctive deductive databases started around early 80's with the appearance of the paper by Minker [26].

2.1 Deductive Databases

During the early 80's, a number of papers about deductive database systems where published [12, 13, 14, 21, 22]. These database systems take first-order logic as the foundation. The foundation of logic programming can be found in [20]. A deductive database can be viewed as the union of two sets EDB and IDB, where the EDB is a set of ground atomic formulae and IDB is a set of definite rules of the form

$$A \leftarrow B_1, \ldots, B_n$$

where A and the B_i, $1 \leq i \leq n$ are atomic formulae, and $n \geq 0$.

With respect to the EDB and IDB, the set of predicate symbols is partitioned into two disjoint sets: the set of base predicates and the set of virtual predicates. The deductive database is assumed to be function free which ensures that answers to queries are finite.

Analogous to the development of normal logic program from definite logic program, the field of deductive databases was extended to allow negated atoms in the body of an IDB clause and the clause of this extended deductive database has the form

$$A \leftarrow B_1, \ldots, B_n, \neg D_1, \ldots, \neg D_m$$

where A, B_i and D_j,$1 \leq i \leq n$,$1 \leq j \leq m$, are atoms and this extended database is called normal deductive database.

In the clause of a normal deductive database, a unary operator $\neg$ is used to indicate a default rule of negation. This default rule of negation serves as a mechanism to compute negative information in the deductive databases which does not explicitly contain negated data. The close world assumption (CWA) and the negation as failure (NAF) are two commonly used rules of negation. The close world assumption states that the negation of an atom may be assumed if one cannot prove the atom. The negation as failure states that if an atom is in the SLD finite failure set of the program P, then the negation of the atom is inferred [7].

A restricted class of normal deductive databases, which is known as stratified databases, were studied [2, 15]. In this restricted class of program, recursion through negation is not allowed. With this restriction, one can divide the logic program (a deductive database can be viewed as a logic program) into layers such that answers to queries over the program can be obtained by execution over those layers consecutively. There was a unique model that characterizes the meaning of this stratified deductive database and this model was shown to be the perfect model of the program [30].

Different optimization techniques for query evaluation has been proposed [3, 4, 29, 32, 34]. Among these optimization techniques, magic sets is considered to be the most general and important one [31]. Magic-sets is a technique that rewrites the rules for each query form, so that both the advantages of top-down and bottom-up methods are achieved [3, 4]. The advantage of top-down evaluation method such as Prolog evaluation procedure is that during the search, only relevant goals to the query are traversed. On the other hand, the

looping-freedom, easy termination testing, and efficient evaluation are the advantages of the bottom-up evaluation method.

2.2 Disjunctive Deductive Databases

The clause of a disjunctive deductive database is a generalization of the clause of the deductive database where disjunction of atoms is allowed in the head of a clause. In deductive databases, there is a single minimal model that characterizes the database. A disjunctive deductive database does not necessarily have a unique minimal model. Hence even when there are no negated atoms allowed in the clauses of the disjunctive deductive database, we may not have a Herbrand model that could characterize the semantics of the database. Take the disjunctive program $\{a \vee b\}$ as example. The program has three models $\{a\}$,$\{b\}$ and $\{a, b\}$. Both models $\{a\}$ and $\{b\}$ are minimal and hence there is no unique minimal model for the program. Though there is not necessarily a unique minimal model for a disjunctive deductive database, there is a set of minimal models which can characterize the disjunctive deductive database. The model-state semantics was introduced [23] which captures the disjunctive logical consequences from a disjunctive logic program as a set of models. This model-state semantics will be described in chapter 4. A state is a set of disjunctive ground clauses and this concept of state was used as the domain of a fixpoint operator T_P that characterizes the disjunctive logic programs where the fixpoint is a model state that contains the set of minimal models of the program [28].

A number of papers [5, 25, 33, 36] have studied the issue of how to deduce negated information, or in particular, solving negated queries in disjunctive deductive databases. The closed world assumption (CWA), which is a default rule of negation for deductive databases, is not sufficient to answer queries for the case when disjunction is allowed. Minker [26] provides a model theoretic definition of negation called generalized closed world assumption (GCWA), which states that negation of an atom is assumed true if the atom is not in any minimal model. Yahya and Henschen [36] extends the GCWA such that negation of conjunction of ground atoms can be determined. This default rule of

negation is termed as extended generalized closed world assumption (EGCWA).

The query answering issue in disjunctive deductive databases has been studied for nearing a decade, which can be traced back to mid 80's [5, 10, 11, 16, 19, 27, 37]. Grant and Minker [27] study the query answering problem in disjunctive databases, which is a restricted case of disjunctive deductive databases in which all predicates in the databases are in EDB. Maybe information was introduced in [19] where the relational model was generalized and the data structure M-table are used to represent the disjunctive data. The relational algebra is generalized to operate on the M-tables but this relational algebra is not complete.

2.3 Model tree for disjunctive deductive databases

The concept of model tree was introduced in [11, 10]. A model tree for a disjunctive deductive database is a tree structure in which the nodes are labeled by the atoms of the Herbrand base of the deductive database and each branch of the tree represents the models of the deductive database.

Consider the following program P:

$a.$

$b \vee c.$

$d \vee e.$

One of model trees is shown in Figure 2.1.[1]

As there may be more than one minimal model for a disjunctive deductive database, the minimal model tree is defined for which there is a one-to-one correspondence between the minimal models and the branches of the minimal model tree. The paper [11] shows how one can compute sound and complete answers to queries in hierarchical disjunctive deductive databases, where recursion is not allowed. Algorithms for the incremental computation on model trees were introduced. Clauses are classified into three groups:

[1]The model tree formed is not unique as it depends on the order of the clauses processed.

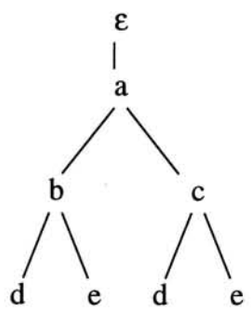


Figure 2.1: A model tree for P

1. Positive disjunctive ground clauses.
2. Clauses with only one atom in their body.
3. Clauses with more than one atom in their body.

A positive disjunctive ground clause is called a disjunctive tuple. It is to be added during the computation of the model tree associated with the extensional part of the database. Those clauses with only one atom in their body are called selection rules, where these clauses only need one atom to be true in a model in order to deduce the conclusion. The clauses with more than one atom in their body are called join rules and more than one atom need to be true in the same model for the conclusion to be true. An algorithm is built for the incremental computation on the model tree for each kind of these clauses.

In [10], Fernandez *et al.* develop a fixpoint operator over model trees to capture the meaning of a disjunctive deductive database that allows recursion. The algorithm of constructing the minimal model tree is represented in this two papers but the minimal model tree constructed is not unique as different model tree will be built by changing the order in which the clauses are processed. In [35], Yahya *et al.* introduce the concept of ordered minimal model trees as the normal form for disjunctive deductive database. An order is imposed on the atoms of the Herbrand base of the database and is used to define the ordered minimal model tree, which is a special form of minimal model tree. With this concept, all minimal model equivalent databases will correspond to a uniquely ordered minimal model tree and the algorithms for constructing and performing operations on ordered minimal model trees are presented.

Chapter 3

Preliminary

3.1 Disjunctive Logic Program

For completeness, we quote some definitions from [24].

A **disjunctive logic program clause** is a program clause of the form:

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_m$$

with $k \geq 1$ and $m \geq 0$, where $A_1, \ldots, A_k, B_1, \ldots, B_m$ are atoms, defined using a FOL λ. When $m = 0$, it is called a **disjunctive assertion**. When $k = 1$, it is called a **definite clause**. Throughout the context of the thesis, we assume λ contain no function symbols.

A **disjunctive logic program** is a program with a finite set of disjunctive logic program clauses.

Definition 3.1 A ground clause $A = A_1 \vee \cdots \vee A_n$ is a **subclause** of a ground clause $B = B_1 \vee \cdots \vee B_m$ if for each $A_i, 1 \leq i \leq n$, there is a B_j such that $A_i = B_j$. The clause A is said to be a **proper subclause** of B if it is a subclause of B and there is a $B_j, 1 \leq j \leq m$, such that B_j is not in A. □

As stated in the previous chapter, a disjunctive logic program does not necessarily have a unique minimal model. For the program that consists of a single clause $\{a \vee b\}$, there are two minimal models $\{a\}$ and $\{b\}$. From this example, firstly, we can see that there is no unique minimal model. Secondly, we can see that the model intersection property

of definite logic programs does not in general apply to disjunctive logic programs as the intersection of model $\{a\} \cap \{b\}$ is $\emptyset$ which is not a model of the program.

Though there is no unique minimal model for disjunctive logic program, the logical consequences of a disjunctive logic program can be characterized by the set of minimal models of the program as stated by the following theorem derived in [26]

Theorem 3.1 Let P be a disjunctive logic program. A positive ground clause C is a logical consequence of P if and only if C is true in every minimal Herbrand model of P.

Given a disjunctive logic program, the **disjunctive Herbrand base** of P, DHB_P, is the set of all disjunctive ground clauses which can be formed with distinct ground atoms from the Herbrand base of P, such that no two logically equivalent clauses are in the set.

Definition 3.2 Let P be a disjunctive logic program. Let S be a set of positive ground clauses in P. The **expansion** of S, denoted by $exp(S)$, is defined as follows:

$$exp(S) = \{C \in DHB_P \mid \exists\ C' \in S \textit{ such that } C' \textit{ is a subclause of } C\} \qquad \square$$

Definition 3.3 Let A and B be two ground clauses, A is said to be **subsumed by** B, or B **subsumes** A, iff B is a proper subclause of A. $\square$

Example 1 Let $Q_1 = A(1) \vee B(1)$ and $Q_2 = A(1) \vee B(1) \vee C(1)$ be two ground clauses, then Q_1 is a proper subclause of Q_2. And we say that Q_1 subsumes Q_2, or Q_2 is subsumed by Q_1.

3.2 Data-disjunctive Logic Program

With the definitions of terminologies of disjunctive logic programming, we now describe the syntax of data-disjunctive logic program.

Definition 3.4 A **data-disjunctive clause** is a clause of the form

$$C(X_{11}, \ldots, X_{1r}) \vee \cdots \vee C(X_{n1}, \ldots, X_{nr}) \leftarrow B_1, \ldots, B_m$$

where $r \geq 1, n \geq 1, m \geq 0$ and each X_{ij} is either a constant or a variable which appears in some atom $B_k, 1 \leq k \leq m$. The clause is represented in the form:

$$C(Y) \mid Y \in \{y_1, \ldots, y_n\} \leftarrow B_1, \ldots B_m$$

where y_i's are tuples and $y_i = (X_{i1}, \ldots, X_{ir}), 1 \leq i \leq n$. And the above clause is said to be defining the predicate C. □

A **data-disjunctive logic program** is a finite set of data-disjunctive clauses and definite clauses.

Example 2 The program

$A(X,1) \vee A(X,2) \leftarrow B(X).$

$B(3) \vee B(4).$

is data-disjunctive where the program

$A(X,1) \vee B(X,1) \leftarrow C(X).$

$C(3) \vee C(4).$

is not a data-disjunctive program as $A(X,1) \vee B(X,1) \leftarrow C(X).$ is not a data-disjunctive clause.

Definition 3.5 A **ground data-disjunctive clause** is a data-disjunctive clause formed with ground atom. A ground instance $C(y_1) \vee C(y_2) \vee \cdots \vee C(y_n) \leftarrow B_1, \ldots, B_m$ where $m \geq 0, n \geq 1, 1 \leq i \leq n$, is represented by

$$C(Y) \mid Y \in \{y_1, \ldots, y_n\} \leftarrow B_1, \ldots, B_m$$

where each B_j,$1 \leq j \leq m$, is a ground atom and each y_i is a tuple of constants. □

We use $C[\{r_1, \ldots, r_n\}]$ to represent the ground clause $C(r_1) \vee \cdots \vee C(r_n)$, where r_i's are tuples of ground terms and we call this a **data-disjunctive fact**.

Given a data-disjunctive logic program P, we can form the dependency graph on the predicates of P. The definition of dependency graph for data-disjunctive logic program is similar to that of a normal program [8].

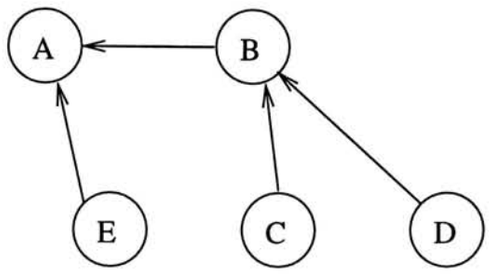


Figure 3.1: The dependency graph for the example

Definition 3.6 The **dependency graph** of a data-disjunctive logic program P has a node for each predicate symbol occurring in P and a directed edge from the node for predicate Q to the node for predicate P whenever predicate Q is in the body of some clause and P is in the head of the same clause. The length of a cycle in a dependency graph is the number of edges occurring in the cycle. □

Example 3 The dependency graph of the program

$$A(X,1) \vee A(X,2) \leftarrow B(X).$$
$$A(X,Y) \leftarrow E(X,Y).$$
$$B(X) \leftarrow C(X), D(X).$$

is as shown in Figure 3.1.

A data-disjunctive logic program is **directly recursive** if its dependency graph does not contain a cycle of length greater than one. As any data-disjunctive logic program can be transformed to a directly recursive data-disjunctive logic program,[1] we make the assumption that the data-disjunctive logic program P is directly recursive throughout the thesis. With this assumption, we can establish an evaluation order of the clause efficiently and the bottom-up evaluation procedure will be shown in Chapter 4.

[1]This can be seen from Theorem 5.1

Chapter 4

Semantics of Data-disjunctive Logic Program

4.1 Model-theoretic semantics

In viewing that a data-disjunctive logic program is a restricted form of disjunctive logic program where only one single predicate is allowed in the head of a clause, the model-theoretic semantics of a data-disjunctive logic program has some relation with that of a disjunctive logic program. Note that in the model state semantics of a disjunctive logic program [23]

$$MS_P = \{C \in DHB_P \mid C \text{ is a logical consequence of } P\}$$

a logical consequence C can be any disjunctive ground clause. In a data-disjunctive logic program, only data-disjunctive facts are concerned. Hence, we define the **data-disjunctive model state semantics** MS_P^{DD} to be

$$MS_P^{DD} = \{C = A(x_1) \vee \cdots \vee A(x_n) \in DHB_P \mid C \text{ is a logical consequence of } P\}$$

i.e. the data-disjunctive model state semantics is a subset of the model state semantics.

For each program P, there is a corresponding model state semantics. Suppose $A \in MS_P^{DD}$ is a logical consequence of the program, we know that all the ground clauses subsumed by A are also logical consequence of the program. This leads to the definition of **minimal**

data-disjunctive model state semantics, denoted by MMS_P^{DD}, where

$$MMS_P^{DD} = \{C \in MS_P^{DD} \mid \nexists\, C' \text{ in } MS_P^{DD} \text{ s.t. } C' \text{ subsumes } C\}$$

Consider a program with relations Bus and Go as example, suppose P has the clause

$Bus(1) \leftarrow Go(swim).$

$Go(swim) \vee Go(barbecue).$

When we treat the program as a data-disjunctive logic program and use the data-disjunctive model state semantics to answer the query, no relevant information can be found in the model state semantics and an answer 'no' will be given for the query. Although $Bus(1) \vee Go(barbecue)$ is not an expected answer for the query, it contains the information that $Bus(1)$ *may* hold. We use a model, which we call contingency model, to express this kind of information.

We give the recursive definition of contingently support and then define the contingency model CM_P.

Definition 4.1 An ground atom A is said to be **contingently supported** by a clause C if either there is a ground instance of C,

$$A_1 \vee \cdots \vee A_k \leftarrow$$

where A equals to some $A_i, 1 \leq i \leq k$, or there is a ground instance of C,

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \cdots, B_m$$

where A equals to some A_i and each B_j is contingently supported by some clause in the program. □

The **contingency model** CM_P is defined by

$$CM_P = \{C \in HB_P \mid C \text{ is } \textit{contingently supported} \text{ by some clause } \textit{in } P\}$$

Definition 4.2 Given a data-disjunctive logic program P, the **minimal data-disjunctive model** of P, MM_P^{DD}, is defined by the following ordered set:

$$MM_P^{DD} = \langle MMS_P^{DD}, CM_P \rangle \qquad \square$$

Theorem 4.1 Given a data-disjunctive logic program P. The minimal data-disjunctive model of P, MM_P^{DD}, is unique. $\square$

Proof To prove the uniqueness of MM_P^{DD}, we have to prove that both MMS_P^{DD} and CM_P are unique given a data-disjunctive program P.

uniqueness of CM_P

Given the program P, we transform P to P' by replacing each clause

$$A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m$$

by

$$A_1 \leftarrow B_1, \ldots, B_m$$
$$A_2 \leftarrow B_1, \ldots, B_m$$
$$\ldots$$
$$A_n \leftarrow B_1, \ldots, B_m$$

Then P' will become a definite logic program and from the definition of contingently support and contingency model, we have

$$CM_P = M_{P'}$$

hence CM_P is unique by the uniqueness property of model of definite logic program.

uniqueness of MMS_P^{DD}

It has been shown that the model state semantic, MS_P, of a given program P is unique [24]. The corresponding data-disjunctive model state semantic MS_P^{DD} is unique since it is the subset of MS_P and is the set of all data-disjunction in MS_P. For all x in DHB_P, $x \in MMS_P^{DD}$ iff $x \in MS_P^{DD}$ and x is not subsumed by y, where $y \in MS_P^{DD}$. Since the relation 'subsume' form a partial order on DHB_P, the collection of elements in MMS_P^{DD} is determined and unique. $\square$

4.2 Fixpoint semantics

As state in chapter 2, we mention that all data-disjunctive logic program P can be transformed to be a directly recursive program. With this property, we have a partial order ($\prec$) on the predicates where for two distinct predicates A and B, we have $A \prec B$ iff there is a path from B to A in the dependency graph. For each predicate, we define a rank for them. A predicate with no child in the dependency graph is assigned rank 0, where the rank of other predicates, $rank(A)$ is defined by

$$rank(A) = \max_{B \prec A}(rank(B)) + 1$$

And we assume that the predicates in the program P rank form 0 to q.

In order to define the fixpoint semantics, we introduce two data structures, **condition term** and **derived term**.

Definition 4.3 A **condition term** has the form $C\{r_1, \ldots, r_n\}$, where $r_1, \ldots, r_n$ are terms called arguments and one of the argument r_i is being marked. □

We underline the marked argument in a condition term, i.e. a condition term will have the form $C\{r_1, \ldots, \underline{r_i}, \ldots, r_n\}$.

Definition 4.4 Let $C_1, \ldots, C_n$ be n condition terms. They are said to be **complementary** if each C_j has the same predicate and arguments $r_1, \ldots, r_n$ and for each $r_i, 1 \leq i \leq n$, there is a uniquely corresponding $C_j, 1 \leq j \leq n$ such that r_i is marked for C_j. The terms $C_1, \ldots, C_n$ are called complementary condition terms. □

Example 4 Let $C_1 = C\{\underline{1}, 2, 3\}, C_2 = C\{1, \underline{2}, 3\}, C_3 = C\{1, 2, \underline{3}\}$, then C_1, C_2 and C_3 are complementary condition terms. □

A condition term $C\{r_1, \ldots, \underline{r_i}, \ldots, r_n\}$ is said to be true if $C(r_i)$ is true. A set of condition terms is said to be true if each condition term in the set is true.

Definition 4.5 A **derived term** has the form $C[\{r_1, \ldots, r_n\}, \langle Cond \rangle]$, where $\langle Cond \rangle$ is a set of condition terms. □

Intuitively, the derived term $C[\{r_1, \ldots, r_n\}, \langle Cond \rangle]$ contains the information that $C[\{r_1, \ldots, r_n\}]$ is true if $\langle Cond \rangle$ is true.

Example 5 Let $C[\{1, 2\}, \langle B\{\underline{1}, 2\}\rangle]$ be a derived term, then its intuitive meaning is that if $B(1)$ is true, we have $C(1) \vee C(2)$ is true. □

Definition 4.6 For a program P, a derived term

$$C[\{r_1, \ldots, r_n\}, \langle A_1\{\ldots, \underline{s}_1, \ldots, \}, \ldots, A_m\{\ldots, \underline{s}_m, \ldots\}\rangle]$$

is *true under* P if the following statement holds:

$C[\{r_1, \ldots, r_n\}$ is a logical consequence of $P \cup \{s_1, \ldots, s_m\}$,
where $s_i, 1 \leq i \leq m$, are the marked arguments in $\langle Cond \rangle$.

Definition 4.7 For a program P, a set of derived terms is true under P if all the derived terms in the set are true under P.

By the definition of derived term, it is obvious that a data-disjunctive fact is a derived term with no condition terms and vice versa. Hence, a derived term $C[\{r_1, \ldots, r_n\}, \langle Cond \rangle]$ with no condition term is a data-disjunction $C[\{r_1, \ldots, r_n\}]$ and vice versa. Here, we extend the definition of subsumption for the derived term. A derived term $C[\{r_1, \ldots, r_n\}, \langle Cond \rangle]$ is said to be **subsumed by** $C[\{s_1, \ldots, s_m\}]$ iff

$$C(s_1) \vee \cdots \vee C(s_m) \text{ is a proper subclause of } C(r_1) \vee \cdots \vee C(r_n)$$

We make use of the directly recursive property of the program to establish an evaluation order for rules in the databases so that each non-recursive clause needs to be evaluated only once during the computation of the fixpoint operators. The rank of the predicates is used to establish this evaluation order and correspondingly, we define the rank of condition term and derived term.

Definition 4.8 Let $C\{r_1, \ldots, \underline{r}_i, \ldots, r_n\}$ be a condition term, then its rank is defined by

$$rank(C\{r_1, \ldots, \underline{r}_i, \ldots, r_n\}) = rank(C)$$

□

Definition 4.9 Let $\langle Cond \rangle$ be a set of condition terms. Suppose $C_1, \ldots, C_n$ are predicates that appear in $\langle Cond \rangle$. Then the rank of $\langle Cond \rangle$ is defined by

$$rank(\langle Cond \rangle) = \max_{i \in \{1,\ldots,n\}} rank(C_i)$$

□

Definition 4.10 Let $C[\{r_1, \ldots, r_n\}, \langle Cond \rangle]$ be a derived term, then its rank is defined by

$$rank(C[\{r_1, \ldots, r_n\}, \langle Cond \rangle]) = rank(C)$$

□

Given a data-disjunctive logic program P, the minimal data-disjunctive model of P, MM_P^{DD} defined by

$$MM_P^{DD} = \langle MMS_P^{DD}, CM_P \rangle$$

has two parts: one is the minimal data-disjunctive model state and the other is the contingency model. For each of these two components, there are corresponding operators.

4.2.1 Fixpoint operators corresponding to the MMS_P^{DD}

We present two fixpoint operators, the F operator and the B operator, to compute the minimal data-disjunctive model state.

The F and B operators

$M_{q+1} = \emptyset$ (empty set).

For $i = q, \ldots, 0$

$F_i^1 = M_{i+1}$

$F_i^{k+1} = F(F_i^k) = \Big\{ C[\{x_1, \ldots, x_n\}, \langle Cond \rangle] \mid C$ is rank i and

$\exists$ non-recursive data-disjunctive clause

$$C \mid Y \in \{y_1, \ldots, y_n\} \leftarrow B_1, \ldots, B_m. \;\; n \geq 1, m \geq 0.$$

with ground instance

$$C[\{x_1, \ldots, x_n\}] \leftarrow B_1(z_1), \ldots, B_m(z_m)$$

where $\forall j, 1 \leq j \leq m$, either $B_j[\{z_j\}, \langle Cond_j \rangle] \in F_i^k$, ($Cond_j$ may be empty)

or $B_j[\{z_j, z'_1, \ldots, z'_s\}, \langle Cond'_j \rangle] \in F_i^k$, ($Cond'_j$ may be empty) where in this case

set $Cond_j = B_j\{\underline{z}_j, z'_1, \ldots, z'_s\} \cup Cond'_j$

and $Cond = Cond_1 \cup \cdots \cup Cond_m$ $\Big\}$

$\cup$

$\Big\{ C[\{r\}, \langle Cond \rangle] \mid C$ is a predicate of rank i and

(1). $\exists$ recursive data-disjunctive clause

$$C \mid Y \in \{y_1, \ldots, y_n\} \leftarrow B_1, \ldots, B_m. \;\; n \geq 1, m \geq 1.$$

and S is the index set where $\forall s \in S, B_s$ is a literal with predicate C.

(2). $\forall s \in S, C[\{v_{s1}, \ldots, v_{sh_s}, \ldots, v_{sw_s}\}, \langle Cond_s \rangle] \in F_i^k$ and

$\exists$ a ground instance of the clause

$$C[\{x_1, \ldots, x_n\}] \leftarrow B_1(z_1), \ldots, B_m(z_m), \text{ with } z_j = v_{jh_j}, \; \forall j \in S$$

(3). $\forall j \notin S$, either $B_j[\{z_j\}, \langle Cond_j \rangle] \in F_i^k$, ($Cond_j$ may be empty)

or $B_j[\{z_j, z'_1, \ldots, z'_t\}, \langle Cond'_j \rangle] \in F_i^k$, ($Cond'_j$ may be empty) where in this case

set $Cond_j = B_j\{\underline{z}_j, z'_1, \ldots, z'_t\} \cup Cond'_j$

(4). $r_s = \{v_{s1}, \ldots, v_{sw_s}\} \setminus \{v_{sh_s}\}$

$r = \bigcup_{s \in S} r_s \cup \{x_1, \ldots, x_n\}$

$Cond = Cond_1 \cup \cdots \cup Cond_m$ $\Big\}$ $\cup$ F_i^k

$B_{i+1,i}^1 =$ *fixpoint of* F_i^k.

For $j, i+1 \leq j \leq q$

$B_{j,i}^{k+1} = B(B_{j,i}^k) = \Big\{ C[\{r\}, \langle Cond \rangle] \mid C$ is rank i and

$\forall s, 1 \leq s \leq m, \exists\, C[\{r_{s1}, \ldots, r_{sn_s}\}, \langle Cond_s \rangle] \in B_i^k$ with $rank(\langle Cond_s \rangle) = j$ and

$\exists$ rank j complementary condition terms in these m $Cond_s$, and

$$Cond = \bigcup_{s=1}^{m} Cond_s \setminus \text{ the complementary condition terms and } r = \bigcup_{s=1}^{m} \{r_{s1}, \ldots, r_{sn_s}\} \Big\} \cup B_{j,i}^k$$

$B_{j+1,i}^1 = fixpoint\ of\ B^\omega(B_{j,i}^1)$

$C_i = Fixpoint\ of\ F_i^k \cup \{\text{derived terms in } B_{q+1,i}^1 \text{ with no condition term}\}$

$M_i = \left\{C \in C_i \mid \nexists\, C' \in C_i \text{ s.t. } C' \text{ subsumes } C\right\}$

The F operator is used to deduce those derived terms when a clause is applied using the previously deduced derived terms. The F operator can be partitioned into two components, the first component applies on the non-recursive clauses and the other component applies on the recursive clauses. The rank 'i' in the F operator is used to restrict which clauses are applied by the F operator. The set of derived terms deduced reach a fixpoint point for a certain rank i as the F operator act on the clauses recursively. The B operator is then used to deduce new information by 'combining' the derived terms according to information of their condition terms.

Example 6 To illustrate the F operator on recursive clauses, consider the program:

$A(X) \leftarrow B(X).$

$B(1) \vee B(2) \vee B(3).$

In the program, the rank of A is 0, the rank of B is 1 and

$F_0^1 = M_1 = C_1 = \{\ B(1) \vee B(2) \vee B(3)\ \}.$

Applying the F operator, we have the following three derived terms

$F_0^2 = \left\{A[\{1\}, \langle B\{\underline{1}, 2, 3\}\rangle],\ A[\{2\}, \langle B\{1, \underline{2}, 3\}\rangle],\ A[\{3\}, \langle B\{1, 2, \underline{3}\}\rangle]\right\}$

which is the fixpoint of the F operator. With this three derived facts, applying the B operator yields the data-disjunctive fact $A(1) \vee A(2) \vee A(3)$.

Example 7 To illustrate the F operator on recursive clauses, consider the program

$T(X,Y) \vee T(X,Z) \leftarrow A(X,Y,Z).$

$T(X,Y) \leftarrow T(X,Z), E(Z,Y).$

$A(1,2,3).$

$E(2,4).$

$E(3,6).$

T is rank 0, E and A are rank 1. $F_0^1 = M_1 = C_1 = \{ A(1,2,3), E(2,4), E(3,6) \}$. Applying the F operator, from the first clause, we get $T[\{(1,2),(1,3)\}]$ and is added to F_0^1 and gives F_0^2, apply the F operator again, as $T(1,4) \leftarrow T(1,2), E(2,4)$ is a ground instance of the second clause and $T[\{(1,2),(1,3)\}]$ in F_0^2, we get

$$T[\{(1,3),(1,4)\}]$$

Similarly, since $T(1,6) \leftarrow T(1,3), E(3,6)$ is a ground instance of the first clause, we get

$$T[\{(1,2),(1,6)\}]$$

and F_0^3 becomes

$$\{ T[\{(1,2),(1,3)\}],\ T[\{(1,2),(1,6)\}],\ T[\{(1,3),(1,4)\}],\ A(1,2,3),\ E(2,4),\ E(3,6) \}$$

Similarly, by the ground instance $T(1,4) \leftarrow T(1,2), E(2,4)$ and $T[\{(1,2),(1,6)\}]$ in F_0^3, we get

$$T[\{(1,4),(1,6)\}]$$

this disjunctive fact is also deduced from the ground instance $T(1,6) \leftarrow T(1,3), E(3,6)$ and $T[\{(1,3),(1,4)\}]$ in F_0^3. Hence, F_0^4 becomes

$$\{ T[\{(1,2),(1,3)\}],\ T[\{(1,2),(1,6)\}],\ T[\{(1,3),(1,4)\}],$$
$$T[\{(1,4),(1,6)\}],\ A(1,2,3),\ E(2,4),\ E(3,6) \}$$

An example is also shown in section 5.4 which illustrates the use of the above operators.

4.2.2 Fixpoint operator corresponding to the contingency model, CM_P

Define a fixpoint operator $T_{CM} : 2^{HB} \rightarrow 2^{HB}$ *where* HB is the Herbrand base.

$$T_{CM}(S) = \Big\{ p \mid p \vee q_1 \vee \cdots \vee q_n \leftarrow s_1, \ldots, s_m \text{ is a ground instance of a clause}$$
$$\textit{and } \forall i, 1 \leq i \leq m, s_i \in S \Big\}$$

Define the powers of T_{CM} as follows:

$T_{CM} \uparrow 0 = \{p \mid p \vee q_1 \vee \cdots \vee q_n \leftarrow$ is a ground instance of a clause of $P\}$

$T_{CM} \uparrow (i+1) = T_{CM}(T_{CM} \uparrow (i))$

$T_{CM} \uparrow \omega = lub\{T_{CM} \uparrow (i) \mid i < \omega\}$

4.3 Equivalence between the model-theoretic and fixpoint semantics

In this section, several theorems on the fixpoint operators will be derived. We first identify the underlining semantics of the derived terms. A derived term

$$C[\{r_1, \ldots, r_s\}, \langle A_1\{a_{11}, \ldots, \underline{a}_{1i_1}, \ldots, a_{1n_1}\}, \ldots, A_m\{a_{m1}, \ldots, \underline{a}_{mi_m}, \ldots, a_{mn_m}\}\rangle]$$

contains the information that if $A_1(a_{1i_1}), \ldots, A_m(a_{mi_m})$ are true, then $C[\{r_1, \ldots, r_s\}]$ is true.

Theorem 4.2 The F operator is sound, i.e. suppose a set of derived terms F_i^k is true under P, then $F(F_i^k)$ is true under P. □

Proof Assume F_i^k is true under P. We want to show that the derived terms computed by the F operator is true under P. We prove only the case in which the clause is a non-recursive data-disjunctive clause, other cases are similar.

Let $C[\{x_1, \ldots, x_n\}] \leftarrow B_1(z_1), \ldots, B_m(z_m)$ be a ground instance of a non-recursive data-disjunctive clause in the program P. If we have $B_j[\{z_j, z'_1, \ldots, z'_s\}, \langle Cond'_j\rangle]$, we know that $B_j[\{z_j, z'_1, \ldots, z'_s\}]$ is true if $\langle Cond'_j\rangle$ is true. Hence $B_j[\{z_j\}]$ is true if $\langle B_j\{\underline{z}_j, z'_1, \ldots, z'_s\}\rangle$ and $\langle Cond'_j\rangle$ are true. Let $\langle Cond_j\rangle$ be $\langle B_j\{\underline{z}_j, z'_1, \ldots, z'_s\}\rangle \cup \langle Cond'_j\rangle$, then $B_j\{z_j\}$ is true if $\langle Cond_j\rangle$ is true. Therefore,

$$B_i\{z_i\} \textit{ is true if} \langle Cond_i\rangle \textit{ is true}, \forall i, 1 \leq i \leq m.$$

By $C[\{x_1, \ldots, x_n\}] \leftarrow B_1(z_1), \ldots, B_m(z_m)$, we have $C[\{x_1, \ldots, x_n\}]$ is true if $\langle Cond_i\rangle$ is true, $\forall i, 1 \leq i \leq m$. Hence

$$C[\{x_1, \ldots, x_n\}, \langle Cond\rangle], \text{ where } \langle Cond\rangle = \bigcup_{i=1}^{m} \langle Cond_i\rangle$$ □

Lemma 4.1 Let S be a set of derived terms. Define a function $Q : 2^S \rightarrow \{true, false\}$ where

$Q(S)$ is true iff for all derived term

$C[\{r_1, \ldots, r_t\}, \langle A_1\{a_{11}, \ldots, \underline{a}_{1i_1}, \ldots, a_{1n_1}\}, \ldots, A_m\{a_{m1}, \ldots, \underline{a}_{mi_m}, \ldots, a_{mn_m}\}\rangle] \in S$,

for any $s, 1 \leq s \leq m$, either

$A_s[\{a_{s1}, \ldots, a_{sn_s}\}]$ is true or

$A_s[\{a_{s1}, \ldots, a_{sn_s}\}]$ is true if $A_j(a_{ji_j})$ are true, $\forall j$ where A_j's rank is larger than A_s's rank.

We have $Q(F_i^k)$ and $Q(M_i)$ are true, $\forall k$, and $\forall i, 0 \leq i \leq q$. □

Theorem 4.3 The B operator is sound, i.e. suppose a set of derived terms B_j^i is true under P, then $B(B_j^i)$ is true under P.. □

Proof Assume $B_{j,i}^k$ is sound. We want to show that derived terms computed by the B operator is sound. Suppose we have $C[\{r_{s1}, \ldots, r_{sn_s}\}, \langle Cond_s\rangle] \in B_{j,i}^k, 1 \leq s \leq m$. We know that $C[\{r_{s1}, \ldots, r_{sn_s}\}]$ is true if $\langle Cond_s\rangle$ is true. Suppose $\langle R_s\rangle = \langle R\{x_1, \ldots, \underline{x}_s, \ldots, x_m\}\rangle$ are complementary condition terms in $\langle Cond_s\rangle$ respectively, $1 \leq s \leq m$. Let $\langle Cond'_s\rangle = \langle Cond_s\rangle \setminus \langle R_s\rangle$, then $R[\{x_1, \ldots, x_m\}]$ is true if $\langle Cond_s\rangle \setminus \langle R_s\rangle$ is true (by the lemma), for any particular s. Hence if $\langle Cond_s\rangle \setminus \langle R_s\rangle$ is true for all $s, 1 \leq s \leq m$, we have

$C[\{r_{s1}, \ldots, r_{sn_s}\}]$ is true if $R(x_s)$ is true and $R[\{x_1, \ldots, x_m\}]$

Hence $C[r]$ is true, $r = \bigcup_{s=1}^m \{r_{s1}, \ldots, r_{sn_s}\}$, if $\bigcup_{s=1}^m \langle Cond'_s\rangle$ is true. □

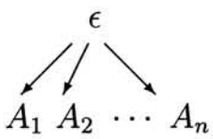
The following theorem states the correspondence of the model-theoretic semantics and the fixpoint semantics.

Theorem 4.4 Let M be the set of derived terms in M_0 computed from the fixpoint operators with no condition terms. Then $M = MMS_P^{DD}$. □

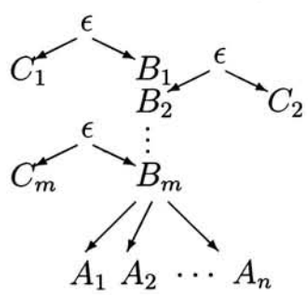
Proof The soundness part of the theorem follows from Theorem 4.2 and 4.3. So, to prove Theorem 4.4, it suffices to show that the fixpoint operator is complete in the sense that $\forall x \in MMS_P^{DD}$, x can be deduced by the fixpoint operator.

For a definite program, we know that a derivation tree can be built for each atom x which is a logical consequence. For disjunctive logic program, the logical consequences are disjunctions of atoms. To prove the theorem, we first have to show that a similar tree (we call it a disjunctive derived graph) can be built for the logical consequence of the disjunctive program.

First, a ground fact $A_1 \vee \cdots \vee A_n$ in the program is represented by the derivation graph:



We follow the immediate consequence operator of disjunctive logic program to build the derivation graph. Suppose $B_1 \vee C_1, \ldots, B_m \vee C_m$ are logical consequences of the program and $A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m$ is a ground instance of a clause, by the immediate consequence operator, $A_1 \vee A_2 \vee \cdots \vee A_n \vee C_1 \vee C_2 \vee \cdots \vee C_m$ will be a logical consequence of the program[1]. This operation can be represented by the graph:

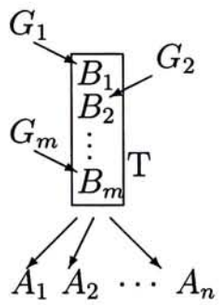


In the graph, there are two kinds of endnodes, the endnodes labelled with ϵ are called the root nodes, where all other endnodes are called leaf nodes. Hence, the consequence of the immediate consequence operator corresponds to the disjunction of leafnodes of the above graph.

We can build the derivation graph recursively by following the immediate consequence operator. Suppose $G_1, \ldots, G_m$ are derivation graph for $B_1 \vee C_1, \ldots, B_m \vee C_m$ respectively and if $A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m$ is a ground instance of a clause, we build a new derivation graph based on $G_1, \ldots, G_m$ by attaching each B_i of G_i, $1 \leq i \leq m$, to a trunk

[1] with removal of duplicate atoms

T as follows:



where trunk T is the block containing $B_1, \ldots, B_m$, and we call the set of nodes $A_1, \ldots, A_n$ a branching point. By the structure of the data-disjunctive logic program, all atoms appear in a particular branching point must have the same predicate. Since all logical consequences can be deduced by the immediate consequence operator, we can assume that each logical consequence of the disjunctive logic program has a corresponding derivation graph. Here we want to prove that all logical consequences which are data-disjunctive facts can be deduced by the fixpoint operator. We prove by induction from M_q to M_0.

In the fixpoint operator, when $i = q$, the F operator is reduced to the immediate consequence operator since when the rank of the predicate is q, the clause can only be of the following form:

$$A(X_1) \vee \cdots \vee A(X_n) \leftarrow A(Y_1), \ldots, A(Y_m).$$

where there is only one predicate in each clause and the body of the clause could be empty.

As the F operator reduces to the immediate consequence operator, there are no derived terms with non-empty condition terms deduced by the F operator, and hence the B operator is not used and M_q contains all logical consequences in MMS_P^{DD} with rank q.

Suppose M_{k+1} contains all logical consequences in MMS_P^{DD} with rank $\geq k+1$, we want to prove M_k contains all logical consequences in MMS_P^{DD} with rank $\geq k$.

For any $x \in DHB_P$, if $x \in MMS_P^{DD}$, a derivation graph can be formed for x where there are m leafnodes for the graph and each leafnode is $C\{Y_i\}$, where $\{Y_1\} \cup \cdots \cup \{Y_m\} = \{x_1, \ldots, x_n\}$. Let x be rank k. The atoms at each branching point in the derivation graph

must have the same predicate. We can see that each leafnode $C\{Y_i\}$ is deducible by the fixpoint operator and represents by the derived term $C[\{Y_i\}, \langle Cond_i \rangle]$ where $Cond_i$ is the set of condition terms which indicate the branching points of the branch of leafnode $C\{Y_i\}$. These derived terms are combined by the B operator to form a graph in which the set of leafnodes are the same as the derived graph.

Since in the fixpoint operator, during the computation of M_i from C_i, some derived terms are removed if they are subsumed by a data-disjunctive fact in C_i. We have to prove that this removal of derived terms will not lead to the inability in deducing some data-disjunctive facts in MMS_P^{DD}.

Suppose not, let x be a data-disjunctive fact in MMS_P^{DD} where the derivation graph for x has nodes at some branching point which are removed in the fixpoint operator. Without loss of generality, let $A\{x_1, \ldots, x_n\}$ be this set of nodes at the branching point. Since it is removed by the fixpoint operator, we have some data-disjunctive fact $A\{z_1, \ldots, z_m\} \in MS_P$, where $\{z_1, \ldots, z_m\}$ is a proper subset of $\{x_1, \ldots, x_n\}$. Hence there is another graph which replace this derivation graph by removing those branches $x_i \notin \{z_1, \ldots, z_m\}$ and the resulting graph will represents x or some x' which subsumes x. Hence the removal of derived terms by the fixpoint operator will not lead to the inability to deduce those data-disjunctive facts in MMS_P^{DD}. □

Theorem 4.5 $T_{CM} \uparrow \omega$ is equal to CM_P, the contingency model of P. □

Proof Follows directly from the definitions of *contingently support* and T_{CM} operator. □

4.4 Operational Semantics

When we are dealing with disjunctive deductive databases, the fixpoint semantics becomes operational as the termination of the operations is guaranteed. We can view the operations of the fixpoint semantics as the bottom-up naive evaluation procedures.

4.5 Correspondence with the I-table

In the context of indefinite and maybe information, a data structure called I-table is used which is capable of representing definite, indefinite and maybe information [19].

Given a program P, once the minimal data-disjunctive model MM_P^{DD} is found, the set of maybe information can be identified accordingly. For a relation (or predicate) Q, the I-table of Q is represented by $T = \langle T_D, T_I, T_M \rangle$ where T_D is the definite component of the I-table and consists of tuples that refer to as definite tuples. T_I is the indefinite component of the I-table and consists of the indefinite tuple sets. Each indefinite tuple is represented by a set of atoms, hence T_I is a set of atoms and is corresponding to the set of disjunctions. T_M is the maybe component of the I-table and consists of tuples which is referred to as maybe tuples. Each of these components can be computed identified easily once we get the minimal data-disjunctive model. Let $MM_P^{DD} = \langle MMS_P^{DD}, CM_P \rangle$ and suppose S be the set of atoms in CM_P which is not appeared in MMS_P^{DD}. Then the set of definite facts of Q in MMS_P^{DD} will be equal to T_D, the definite tuples. The set of disjunctions of Q in MMS_P^{DD} corresponds to T_I and the set of atoms of Q in S is corresponding to the maybe tuples T_M.

For example, suppose we have the following program:

$Q(1)$.
$Q(1) \vee Q(2)$.
$Q(3) \vee Q(4)$.

then we have $T_D = \{Q(1)\}$, $T_I = \{\{Q(3), Q(4)\}\}$ and $T_M = \{Q(2)\}$.

It should be easily seen that the data-disjunctive model state semantics is

$$MS_P^{DD} = \{Q(1),\ Q(1) \vee Q(2),\ Q(3) \vee Q(4)\}$$

After removing the disjunction which are assumed by others, we get the minimal data-disjunctive model state semantics

$$MMS_P^{DD} = \{Q(1),\ Q(3) \vee Q(4)\}$$

and the contingency model CM_P will be $\{Q(1), Q(2), Q(3), Q(4)\}$. Hence the minimal data-disjunctive model is

$$MM_P^{DD} = \langle MMS_P^{DD}, CM_P \rangle = \langle \{Q(1),\ Q(3) \vee Q(4)\}, \{Q(1), Q(2), Q(3), Q(4)\} \rangle$$

With this data-disjunctive model state semantics, we can see that the set of non-disjunctive facts in MMS_P^{DD} corresponds to component T_D, the set of disjunctive facts in MMS_P^{DD} corresponds to the component T_I and $S = \{Q(2)\}$ is the set of maybe tuples T_M.

The minimal data-disjunctive model extends the I-table in the sense that it contain the definite, indefinite and maybe information of a data-disjunctive deductive database rather than a relational database. Those non-disjunctive ground facts in MMS_P^{DD} are the definite information, the disjunctive ground facts in MMS_P^{DD} are the indefinite information and all the other atoms in CM_P constitute the maybe information.

Chapter 5

Disjunctive Deductive Databases

5.1 Disjunctions in deductive databases

For each query posted for a disjunctive deductive database, a set of answer should be deduced accordingly. For a non-disjunctive databases DB, the set of answers to a query Q can be defined by the set of ground atoms,

$$\{ A \in M_{DB} : A \Rightarrow \exists Q \}$$

where M_{DB} denotes the unique minimal model of DB. For disjunctive deductive databases, there are different definitions of answer. In tradition, the set of answers for a query is defined to be [1, 11]

$$\{ A_1 \vee \cdots \vee A_k : k > 0 \text{ and } \forall \textit{ minimal model } M, M \models A_1 \vee \cdots \vee A_k \textit{ and } \forall i, A_i \Rightarrow \exists Q \}$$

We call this set of answer the **traditional answer set**. Under this definition of answer, given a query ?-$A(1, X)$, and knowing that the ground clauses $A(1, a) \vee A(1, b)$, $A(1, a) \vee A(2, a)$ are logical consequences of the databases, the clause $A(1, a) \vee A(1, b)$ constitutes answer to the query as both $A(1, a)$ and $A(1, b)$ imply $\exists X, [A(1, X)]$. The ground clause $A(1, a) \vee A(2, a)$ does not constitute an answer because $A(2, a)$ does not implies $\exists X, [A(1, X)]$.

As disjunction is considered in disjunctive logic program, we expect that the answer to a query is not always in a simple 'yes' or 'no' form. Suppose we have a program consisting of one single ground clause

$Colour(car, blue) \vee Colour(car, red)$

which states that the car is either blue or red in colour. If the query ?-$Colour(car, blue)$ is given, under the previous definition of answer, the query reduces to a 'yes' or 'no' question as only $Colour(car, blue)$ implies $Colour(car, blue)$. Although $Colour(car, blue)$ is not a logical consequence of the program, we do not want to give a simple answer 'no' to the query since we know that the car is probably blue in colour. The answer to the query should at least express this uncertainty (or possibility) of the car being blue. For example, an answer could be

Probable: $Colour(car, blue)$

Alternatively, in a probably better way, the answer could express relevant knowledge to the query, for example, the answer can be

$Colour(car, blue) \vee Colour(car, red)$

which expresses more information than the former case.

In the scope of data-disjunctive deductive databases, we define the answer for a query Q to be the set of ground clause

$$\{A(x_1) \vee \cdots \vee A(x_k) \in MMS_P^{DD} \mid \exists i,\ A(x_i) \Rightarrow \exists Q\ \}$$

and this answer set is called the **relevant answer set**.

Consider the program

$Bus(1) \vee Bus(2)$.

$Bus(1)$.

For the query ?-$Bus(2)$, both the traditional answer set and the relevant answer set are not able to express the possibility that $Bus(2)$ is true. Hence, in addition to the relevant answer set, we define the **contingency answer set** for the query Q to be

$$\{A(x) \in CM_P \mid A(x) \Rightarrow \exists Q\ \}$$

This contingency answer set contains all those atoms which 'match' with the query and is contingency support by the program.

5.2 Relation between predicates

In order to compute the set of answers with relevant knowledge to a given query, one can naively compute the set of all logical consequences of the deductive database and project those relevant ones from these logical consequences. The set of all logical consequences can be unconditionally large. For example, if an atom A is a logical consequence of the deductive databases, any ground clause subsumed by A will also be a logical consequence. So it is not reasonable to compute all the logical consequences of a deductive database. For simplicity, we only need to compute those logical consequences which are not subsumed by others. In this way, we can reduce the number of logical consequences to be computed. Another way to reduce the size of computed logical consequences is by restricting the possible form of disjunctive ground clauses. Suppose we have the following deductive database:

$$Father(X,Y) \vee Mother(X,Y) \leftarrow Parent(X,Y)$$
$$Grandfather(X,Y) \vee Grandmother(X,Y) \leftarrow Parent(X,Z), Father(Z,Y)$$
$$Parent(terri, peter)$$
$$Parent(joe, terri)$$

The logical consequence of the program is $exp(M)$, where M is

$$\{\ Parent(terri, peter),$$
$$Parent(joe, terri),$$
$$Father(terri, peter) \vee Mother(terri, peter),$$
$$Father(joe, peter) \vee Mother(joe, peter),$$
$$Grandfather(joe, peter) \vee Grandmother(joe, peter) \vee Mother(terri, peter)\}$$

Then the query ?-$Mother(terri, peter)$ will yield the two answer

1. $Father(terri, peter) \vee Mother(terri, peter)$,
2. $Grandfather(joe, peter) \vee Grandmother(joe, peter) \vee Mother(terri, peter)$

On the other hand, it is unnatural to have the answer

$$Grandfather(joe, peter) \vee Grandmother(joe, peter) \vee Mother(terri, peter)$$

for the query. This is because, from the program, we do not normally expect that a fact in the form

$$Grandfather(\cdot) \vee Grandmother(\cdot) \vee Mother(\cdot)$$

to be deduced. On the other hand, in viewing of the clause

$$Father(X, Y) \vee Mother(X, Y) \leftarrow Parent(X, Y)$$

we would likely expect that some disjunctive facts of the form $Father(\cdot) \vee Mother(\cdot)$ will be deduced. Similarly, from the clause

$$Grandfather(X, Y) \vee Grandmother(X, Y) \leftarrow Parent(X, Z), Father(Z, Y)$$

we would expect that the disjunctive facts of the form $Grandfather(\cdot) \vee Grandmother(\cdot)$ will be deduced if the conditions in the body of the clause are satisfied. This investigation leads to the definition of standard-form disjunctive fact.

Definition 5.1 Given a disjunctive logic program P, a fact G:

$$A_1 \vee \cdots \vee A_n, \ n \geq 1$$

is called a **standard-form fact** if there exists a clause C in P such that for all predicate A, A appear in G if A is in the head of the clause C. □

Example 8 Suppose one of the clauses of a program is

$$A(X) \vee B(X) \leftarrow C(X)$$

then all the facts $A(1) \vee B(1)$, $A(1) \vee B(1) \vee B(2)$, $A(1) \vee A(2) \vee A(3) \vee B(1)$ are in standard form. □

If we have the program

$A_1 \vee B_1$.

$B_2 \leftarrow B_1$.

$B_3 \leftarrow B_2$.

$\ldots$

$B_n \leftarrow B_{n-1}$.

Then the query ?-A_1 will yield the answers

$\{ \quad A_1 \vee B_1,$

$A_1 \vee B_2,$

$A_1 \vee B_3,$

$\ldots$

$A_1 \vee B_n \ \}$

We can see that all the elements in the answer are deduced from the fact $A_1 \vee B_1$, and note that the only element in standard form is $A_1 \vee B_1$, hence if we make the assumption that all interesting facts are in standard form, we can reduce the answer to $\{A_1 \vee B_1\}$ only.

Naturally, we have to sacrifice something when we assume that the only interesting facts are in standard form. For, example, consider the program

$Mother(X, Y) \vee Father(X, Y) \leftarrow Parent(X, Y)$

$Grandfather(X, Z) \leftarrow Father(X, Y), Parent(Y, Z)$

$Grandmother(X, Z) \leftarrow Mother(X, Y), Parent(Y, Z)$

$Parent(terri, peter)$

$Parent(peter, mary)$

Then when the query is ?-$Grandmother(terri, mary)$, we cannot get the answer

$Grandfather(terri, mary) \vee Grandmother(terri, mary)$

since it is not in standard form. In fact, this problem can be solved by adding a simple clause to the program. As we expect that the facts of the form $Grandfather(\cdot) \vee Grandmother(\cdot)$ is interesting, we add a clause

$Grandfather(\eta) \vee Grandmother(\eta)$.

to the program, where η is a default constant. In this way, the fact

$$Grandfather(terri, mary) \vee Grandmother(terri, mary)$$

will be in standard form.

In fact, the concept of 'standard-form fact' provides a strategy for the system to identify which kinds of disjunction are interesting and expected by the user. This strategy is used in the transformation from the original program to the program which the evaluation method can be applied.

5.3 Transformation of Disjunctive Deductive Databases

Theorem 5.1 Given a disjunctive logic program P, we can transform it to a data-disjunctive logic program such that all logical consequences of the program P will be logical consequences of the transformed program.

Proof Given any disjunctive logic program P, a trivial transformation can be done as follows:

Let $P_1, \ldots, P_n$ be the predicates[1] of P. As for each predicate, it has a unique number of arguments. Let k be the maximum number of arguments of the predicates in P. We introduce a new predicate, say T, with $k+1$ arguments, and a new constant, say η. We transform the program P by replacing[2] each m-ary term $P_i(X_1, \ldots, X_m)$ by $T(X_1, \ldots, X_m, \eta, \eta, \ldots, \eta, i)$ in all clauses of P and add the clause

$$P_i(X_1, \ldots, X_m) \leftarrow T(X_1, \ldots, X_m, \eta, \eta, \ldots, \eta, i)$$

for each predicate P_i. □

From the proof, we note two things: first, we can see that any disjunctive logic program, even if it has cycles of length more than one in the dependency graph, can be transformed

[1] when we deal with disjunctive deductive database, only the predicates defined in IDB is considered
[2] for disjunctive deductive database, those terms with predicates defined in EDB is not replaced

to a directly recursive data-disjunctive logic program. Hence we see that the expressive power of the directly recursive data-disjunctive logic program is the same of the disjunctive logic program. Secondly, we use the trivial transformation to show that any disjunctive logic program can be transformed to a data-disjunctive logic program. But it should be noted that many different transformations are possible to transform a disjunctive logic program into a data-disjunctive logic program. For example, let P be

$$Male(X) \vee Female(X) \leftarrow Human(X)$$
$$Human(terri)$$

Then P can be transformed into the following directly recursive data-disjunctive logic program P':

$$Male(X) \leftarrow T(X, 1)$$
$$Female(X) \leftarrow T(X, 2)$$
$$T(X, Y) \mid Y \in \{1, 2\} \leftarrow Human(X)$$
$$Human(terri)$$

In facts, the aim of the transformation is to replace a clause with more than one predicates in the head by a data-disjunctive clause. We can see that the performance of the evaluation method depends on the transformation chosen. A general strategy is: do not use a new predicate to replace too many predicates if it is unnecessary. As the last example, we found that the clause

$$Male(X) \vee Female(X) \leftarrow Human(X)$$

has two different predicates in its head, hence a new predicate T must be used to replace the predicate $Male$ and $Female$, but we do not use this predicate T to replace the predicates $Human$ since it is not necessary to do so. This is one of the strategies that can be used for choosing the transformation but discussion on this issue is beyond the scope of this thesis.

Once we get the transformed data-disjunctive program, we can apply the procedure

of the operational semantics to the transformed program to find the minimal data-disjunctive model state. For the last example, we found that the logical consequence $Male(terri) \vee Female(terri)$ of the original program P corresponds to the data-disjunctive fact $T(terri, 1) \vee T(terri, 2)$ of the data-disjunctive program P'. The correspondence of logical consequence between the original program and the transformed program is stated by the following theorem:

Theorem 5.2 Given a disjunctive deductive database P and a transformed data-disjunctive deductive database P'. Any logical consequence of P which is in standard form corresponds to a data-disjunctive fact of P' in the following sense:

Suppose

$$A_1(x_1) \vee \cdots \vee A_n(x_n)$$

is a standard-form fact which is a logical consequence of the original program P, it corresponds to a data-disjunctive fact

$$T(x_1, \eta, \ldots, \eta, s_1) \vee \cdots \vee T(x_n, \eta, \ldots, \eta, s_n)$$

where the clause

$$A_i(X_i) \leftarrow T(X_i, \eta, \ldots, \eta, s_i) \text{ is in } P', \forall i,\ 1 \leq i \leq n. \qquad \square$$

Proof As discussed above.

From now on, we shall use the terms data-disjunctive deductive databases and deductive databases interchangeably.

5.4 Query answering for Disjunctive Deductive Databases

In this section, we give an example of disjunctive deductive databases to illustrate how the evaluation procedure compute the data-disjunctive facts which are logical consequences of the transformed program.

Let the deductive database P be

$$Mother(X,Y) \vee Father(X,Y) \leftarrow Parent(X,Y)$$
$$Grandfather(X,Z) \leftarrow Father(X,Y), Parent(Y,Z)$$
$$Grandmother(X,Z) \leftarrow Mother(X,Y), Parent(Y,Z)$$
$$Grandparent(X,Y) \leftarrow Grandfather(X,Y)$$
$$Grandparent(X,Y) \leftarrow Grandmother(X,Y)$$
$$Parent(terri, peter)$$
$$Parent(peter, mary)$$

First, we transform P to a data-disjunctive logic program P', where P' is

$$Mother(X,Y) \leftarrow T(X,Y,1)$$
$$Father(X,Y) \leftarrow T(X,Y,2)$$
$$T(Z) \mid Z \in \{(X,Y,1),(X,Y,2)\} \leftarrow Parent(X,Y)$$
$$Grandfather(X,Z) \leftarrow Father(X,Y), Parent(Y,Z)$$
$$Grandmother(X,Z) \leftarrow Mother(X,Y), Parent(Y,Z)$$
$$Grandparent(X,Y) \leftarrow Grandfather(X,Y)$$
$$Grandparent(X,Y) \leftarrow Grandmother(X,Y)$$
$$Parent(terri, peter)$$
$$Parent(peter, mary)$$

After transforming the program, we compute the rank of the predicates of the transformed program.

rank	Predicate
0	Grandparent
1	Grandfather, Grandmother
2	Father, Mother
3	T
4	Parent

we have q, the maximum rank of the predicates, equals to 4.

$C_4 = \{\ Parent(terri, peter), Parent(peter, mary)\}$

$M_4 = C_4,\ F_3^1 = M_4$

$B_{4,3}^1 = F_3^2 = \Big\{T[\{(terri, peter, 1), (terri, peter, 2)\}], T[\{(peter, mary, 1), (peter, mary, 2)\}]\Big\} \cup F_3^1$

$C_3 = F_3^2,\ M_3 = C_3,\ F_2^1 = M_3$

$F_2^2 = \Big\{\ Mother[\{terri, peter\}, \langle T\{\underline{(terri, peter, 1)}, (terri, peter, 2)\}\rangle],$

$Father[\{terri, peter\}, \langle T\{(terri, peter, 1), \underline{(terri, peter, 2)}\}\rangle],$

$Mother[\{peter, mary\}, \langle T\{\underline{(peter, mary, 1)}, (peter, mary, 2)\}\rangle],$

$Father[\{peter, mary\}, \langle T\{(peter, mary, 1), \underline{(peter, mary, 2)}\}\rangle]\Big\} \cup F_2^1$

$B_{4,2}^1 = B_{3,2}^1 = F_2^2,\ C_2 = F_2^2,\ M_2 = C_2,\ F_1^1 = M_2$

$F_1^2 = \Big\{\ Grandmother[\{terri, mary\}, \langle T\{\underline{(terri, peter, 1)}, (terri, peter, 2)\}\rangle],$

$Grandfather[\{terri, mary\}, \langle T\{(terri, peter, 1), \underline{(terri, peter, 2)}\}\rangle]\Big\} \cup F_1^1$

$B_{4,1}^1 = B_{3,1}^1 = B_{2,1}^1 = F_1^2,\ C_1 = F_1^2,\ \ M_1 = C_1,\ F_0^1 = M_1$

$F_0^2 = \Big\{\ Grandparent[\{terri, mary\}, \langle T\{\underline{(terri, peter, 1)}, (terri, peter, 2)\}\rangle],$

$Grandparent[\{terri, mary\}, \langle T\{(terri, peter, 1), \underline{(terri, peter, 2)}\}\rangle]\Big\} \cup F_0^1$

$B_{3,0}^1 = B_{2,0}^1 = B_{1,0}^1 = F_0^2$

$B_{3,0}^2 = \{\ Grandparent(terri, mary)\} \cup B_{3,0}^1$

$C_0 = F_0^2 \cup \{Grandparent(terri, mary)\}$

$M_0 = \{Grandparent(terri, mary)\} \cup M_1$

$CM_P = \{\ Parent(peter, mary),\ Parent(terri, peter),$

$T(peter, mary, 1),\ T(peter, mary, 2),$

$T(terri, peter, 1),\ T(terri, peter, 2),$

$Mother(peter, mary),\ Mother(terri, peter),$

$Father(peter, mary),\ Father(terri, peter),$

$Grandmother(terri, mary),\ Grandfather(terri, mary),$

$Grandparent(terri, mary).\ \}$

Suppose the query ?-$Father(terri, peter)$ is posted. As we found that the term $Father(X, Y)$ is replaced by $T(X, Y, 2)$ during the transformation. The query ?-$Father(terri, peter)$ of the original program is corresponding to the query ?-$T(terri, peter, 2)$ to the transformed program. As the data-disjunctive fact $T(terri, peter, 1) \vee T(terri, peter, 2)$ is found in M_0 and $T(terri, peter, 1)$ corresponds to $Mother(terri, peter)$, we get the answer $Father(terri, peter) \vee Mother(terri, peter)$.

For the query ?-$Grandparent(terri, mary)$, we found that the fact $Grandparent\ (terri, mary)$

is in M_0, hence $Grandparent(terri, mary)$ holds. For the query ?-$Grandfather\ (terri, mary)$, we cannot find any relevant disjunctive fact in M_0 but as Grandfather(terri,mary) is found in CM_P, we get

Probable : $Grandfather(terri, mary)$

which indicates that Terri maybe a grandfather of Mary.

Note that we can get the fact $Grandfather(terri, mary) \vee Grandmother(terri, mary)$ if we transformed the program in a different way by replacing the clauses

$$Grandfather(X, Z) \leftarrow Father(X, Y), Parent(Y, Z)$$
$$Grandmother(X, Z) \leftarrow Father(X, Y), Parent(Y, Z)$$

by

$$S(X, Z, 1) \leftarrow Father(X, Y), Parent(Y, Z)$$
$$S(X, Z, 2) \leftarrow Father(X, Y), Parent(Y, Z)$$
$$Grandfather(X, Y) \leftarrow S(X, Y, 1)$$
$$Grandmother(X, Y) \leftarrow S(X, Y, 2)$$

This shows that the number of logical consequences deduced by the evaluation method depends on the transformation of the program. As in the last example, we have the knowledge that the disjunction of atoms with relation Grandfather and Grandmother is meaningful and useful, we should transformed the program in the later way.

Chapter 6

Magic for Data-disjunctive Deductive Database

In this chapter, we propose a rule rewriting method that can be applied to the data-disjunctive deductive database such that when a query is posted, the answer set can be computed efficiently with the usage of the constraint propagated from the query to the subgoals as the search tree is traversed.

We assume the query is atomic throughout this chapter, which means that there is only one term in the query. We make use of the magic set method for the definite deductive database in the rule rewriting algorithm for the data-disjunctive deductive database and details of this magic set method can be found in [4].

6.1 Magic for Relevant Answer Set

As stated in the previous chapter, we define the answer for a query Q for the disjunctive deductive database to be the set of ground clause

$$\{A_1 \vee \cdots \vee A_k \in MMS_P^{DD} \mid \exists i,\ A_i \Rightarrow \exists Q\ \}$$

This definition is different from the traditional definition of answer, but clearly, when the query is atomic, the set of answers yielded by the new definition is a superset of the traditional answer set. Hence, if we can develop a rule rewriting method and bottom-up

evaluation method to find the relevant answer set, we can easily get the traditional answer set also.

For a query ?-$Q(X_1, \ldots, X_k)$ where X_i is either a constant or variable, the answer can be found by solving the query ?-$Q(Y_1, \ldots, Y_k)$, where all Y_i's are variables and find those relevant answers from this set of answers obtained. If we evaluate the query in this way, the information known in the value of X_i will not be used in guiding the searching progress. This information is used in projecting relevant tuples from a larger set only. We want to develop a query evaluation procedure similar to the magic set method of definite deductive database such that the searching process is constrained by the knowledge on the bindings of the query.

Use a simple example to illustrate the searching process in disjunctive deductive database, suppose P is a program with the following clauses:

$A(X) \leftarrow B(X)$

$B(1) \vee B(2)$

$B(2) \vee B(3)$

$\ldots$

$B(9) \vee B(10)$

A simple query ?-$A(2)$ should yield the answer $\{ A(1) \vee A(2),\ A(2) \vee A(3) \}$. If the query evaluation method finds the answer for ?-$A(Y)$ first, the set of answer $\{ A(i) \vee A(i+1) : 1 \leq i \leq 9 \}$ is computed and the answer for the query ?-$A(2)$ is projected from this set. In fact, the information of Y binding to the value 2 in the query can be used in the evaluation process. Firstly, as in SLD resolution, $A(2)$ is unified with the first clause with the variable X binded to 2. Then, the evaluation process continues to solve the subquery ?-$B(2)$. Only the ground clauses $B(1) \vee B(2)$ and $B(2) \vee B(3)$ contain atom $B(2)$ which match with the subquery ?-$B(2)$. Applying the bottom-up evaluation algorithm on these two ground clauses the answer $\{ A(1) \vee A(2),\ A(2) \vee A(3) \}$ is obtained.

In the example, we should note that the query answering procedure can be viewed as two parts: A top-down procedure and a bottom-up procedure. The top-down part contributes

to the passing of information obtained by the query to the subquery. In this procedure, those relevant ground clauses ($B(1) \vee B(2)$ and $B(2) \vee B(3)$ in the example) are marked. In the bottom-up part, we make use of those ground clause that are marked relevant to derive the answer tuples for the query.

In the definite deductive database, the magic set method rewrites the clauses of the program such that the information passing procedure can be done in a bottom-up manner. We would like to develop a rewriting method applicable to disjunctive deductive database such that the answer of a query can be computed by the proposed bottom-up evaluation algorithm. As disjunction is considered, we do not expect that the magic set rewriting method can be applied directly. Before introducing the rewriting method for the data-disjunctive deductive database, we define some terms first.

Definition 6.1 Given a program P and a query ?-$T(x_1, \ldots, x_n)$. The set of **relevant clauses** of T is defined recursively as follows:

1. All clauses defining predicate T are relevant clauses.
2. Let P be a predicate that appears in a relevant clause of T, all clauses defining predicate P are relevant clauses. □

Definition 6.2 All predicates appear in the relevant clauses are defined as **relevant predicates** of T. □

6.1.1 Rule rewriting algorithm

INPUT: A set of clauses and a query ?-$T(X_1, \ldots, X_n)$ (or T in short), where X_i is either a variable or a constant.

OUTPUT: A set of clauses such that applying the bottom-up evaluation procedures yields answer to the query.

Details: Let the output set of clauses be S, then the program is rewritten in two sets of clauses, S_1 and S_2, where $S_1 \cup S_2 = S$. The clauses of S_1 correspond to the original magic set rewriting method while S_2 is specialized in data-disjunctive deductive database.

A. Assume all relevant clauses have single atom at the head part

Generation of S_1:

The set of clauses is generated by the magic set rewriting method [4].

Generation of S_2:

1. Add $T(X) \leftarrow ST(X).$ into S_2.
2. For all relevant predicates P_i in IDB, add

   $mg_{SP_i}(X) \leftarrow SP_i(X).$
3. For all relevant clauses

   $A \leftarrow B_1, \ldots, B_m.$

add

   $SA \leftarrow SB_1, \ldots, SB_m.$

4. $\forall k$, $1 \leq k \leq m$, suppose SB_j, $1 \leq j \leq m$ and $j \neq k$ are the predicates connected to SB_k in the clause, then for all clauses define SB_j of the form

   $SB_j(X) \leftarrow D_{j1}, \ldots, D_{js}.$

add

   $SB_j(X) \leftarrow mg_{SB_k}(Y), E_{j1}, \ldots, E_{js}$

where

   (a) $E_{jl} = D_{jl}$ if predicate of D_{jl} is in EDB.

   (b) $E_{jl} = SD_{jl}$ if predicate of D_{jl} is in IDB.

   (c) The list of terms Y is defined by the knowledge on the connected terms between SB_k and SB_j in the clause such that the corresponding connected terms in X and Y have the same variables.

B. General case of data-disjunctive clauses

The above algorithm is used when the clauses of the program have only one atom in the head. In general, for a given predicate P, the clauses defining P may have more than one atom in the head as the form

$$P(X_1) \vee \cdots \vee P(X_n) \leftarrow B_1, \ldots, B_m$$

Both the magic set rewriting method and the specialized algorithm defined above have to be modified for these predicates.

Generation of S_1:

As in the magic set method, there are two modes of information passing in the evaluation of a query. The first mode is identified as unification. For deductive database, the unification between goal and head of predicate is well-defined as there is only one atom in the head of a clause. However, this unification differs in the case of disjunctive deductive database since for a query $P(Y)$, as in unification, $P(Y)$ maybe "unify" with *anyone* of $P(X_i), 1 \leq i \leq n$. So the clause should be viewed as n distinct clauses

$$P(X_i) \leftarrow B_1, \ldots, B_m$$

When applying the magic set rewriting method, a corresponding clause

$$P(X_i) \leftarrow \text{a list of magic predicates}, B_1, \ldots, B_m.$$

will be generated and we rewrite these n corresponding clauses by replacing the head part $P(X_i)$ by the original head $P(X_1) \vee \cdots \vee P(X_n)$.

Generation of S_2:

1. Add $T(X) \leftarrow ST(X).$ into S_2.
2. For all relevant predicates P_i in IDB, add

   $$mg_{SP_i}(X) \leftarrow SP_i(X).$$

3. For all relevant clauses

   $$A(X_1) \vee \cdots \vee A(X_n) \leftarrow B_1, \ldots, B_m.$$

add

$$SA(X_1) \vee \cdots \vee SA(X_n) \leftarrow SB_1, \ldots, SB_m.$$

4. $\forall k$, $1 \leq k \leq m$, suppose SB_j, $1 \leq j \leq m$ and $j \neq k$ are the predicates connected to SB_k in the clause, then for all clauses define SB_j of the form

$$SB_j(X_1) \vee \cdots \vee SB_j(X_n) \leftarrow D_{j1}, \ldots, D_{js}.$$

add n clauses with variable t, $1 \leq t \leq n$

$$SB_j(X_1) \vee \cdots \vee SB_j(X_n) \leftarrow mg_{SB_k}(Y_t), E_{j1}, \ldots, E_{js}$$

where

(a) $E_{jl} = D_{jl}$ if predicate of D_{jl} is in EDB.

(b) $E_{jl} = SD_{jl}$ if predicate of D_{jl} is in IDB.

(c) The list of terms Y is defined by the knowledge on the connected terms between SB_k and SB_j in the clause such that the corresponding connected terms in X_t, $1 \leq t \leq n$, and Y have the same variables.

6.1.2 Bottom-up evaluation

Once the set of clauses S is generated, the bottom-up evaluation is used to compute the answer for the query with the following rules:

1. When a ground fact $P(x_1) \vee \cdots \vee P(x_n)$ is used in producing any atom the stimulated fact $SP(x_1) \vee \cdots \vee SP(x_n)$ is added.

2. The magic predicates are used for constraining the search tree. We do not store the disjunction of atoms of magic predicate. Suppose $mg_P(x_1) \vee \cdots \vee mg_P(x_n)$ is deduced, then $mg_P(x_i), 1 \leq \imath \leq n$, is assumed.

6.1.3 Examples

Example 9 Program:

$$T(X, Z) \leftarrow A(X, Y), B(Y, Z).$$

$A(1,3) \vee A(2,3).$

$B(3,4).$

Query:

?-$T(1,X)$

Apply the rule rewriting algorithm, the program is rewritten into:

S_1:

clause 1: $mg_T(1).$

clause 2: $T(X,Z) \leftarrow mg_T(X), A(X,Y), B(Y,Z).$

S_2:

clause 3: $T(X,Y) \leftarrow ST(X,Y).$

clause 4: $ST(X,Z) \leftarrow SA(X,Y), SB(Y,Z).$

clause 5: $mg_{SA}(X,Y) \leftarrow SA(X,Y).$

clause 6: $SB(X,Y) \leftarrow mg_{SA}(U,X), B(X,Y).$

clause 7: $mg_{SB}(X,Y) \leftarrow SB(X,Y).$

clause 8: $SA(X,Y) \leftarrow mg_{SB}(Y,V), A(X,Y).$

Evaluation procedure:

1. $mg_T(1).$
2. From clause 2, we get $T[(1,4), \langle A\{\underline{(1,3)}, (2,3)\}\rangle]$
   stimulate $A[(1,3),(2,3)]$, so add $SA[(1,3),(2,3)]$
   from clause 5, get $mg_{SA}(1,3)$ and $mg_{SA}(2,3)$.
   from clause 6, get $SB(3,4)$.
3. From clause 4, deduce $ST[(1,4), \langle A\{\underline{(1,3)}, (2,3)\}\rangle]$ and $ST[(2,4), \langle A\{(1,3), \underline{(2,3)}\}\rangle]$
   deduce $ST[(1,4),(2,4)]$
4. Deduce $T[(1,4), \langle ST\{\underline{(1,4)}, (2,4)\}\rangle]$ and $T[(2,4), \langle ST\{(1,4), \underline{(2,4)}\}\rangle]$ from clause 3
   which can deduce $T[(1,4),(2,4)]$ by the bottom-up evaluation procedure.

Example 10 For the program

$$T(X,Y) \vee T(Y,Z) \leftarrow A(X,Y), B(Y,Z).$$

$$?\text{-}T(1,X)$$

is rewritten into:

S_1:

$$mg_T(1).$$

$$T(X,Y) \vee T(Y,Z) \leftarrow mg_T(X), A(X,Y), B(Y,Z).$$

$$T(X,Y) \vee T(Y,Z) \leftarrow mg_T(Y), A(X,Y), B(Y,Z).$$

S_2:

$$T(X,Y) \leftarrow ST(X,Y).$$

$$ST(X,Y) \vee ST(Y,Z) \leftarrow SA(X,Y), SB(Y,Z).$$

$$mg_{SA}(X,Y) \leftarrow SA(X,Y).$$

$$SB(X,Y) \leftarrow mg_{SA}(U,X), B(X,Y).$$

$$mg_{SB}(X,Y) \leftarrow SB(X,Y).$$

$$SA(X,Y) \leftarrow mg_{SB}(Y,V), A(X,Y).$$

Example 11 For the program

$$T(X,Y) \leftarrow P(X,Y).$$

$$P(X,Y) \vee P(Y,Z) \leftarrow A(X,Y), B(Y,Z).$$

$$?\text{-}T(1,X)$$

is rewritten into:

S_1:

$$mg_T(1).$$

$$T(X,Y) \leftarrow mg_T(X), P(X,Y).$$

$$mg_P(X) \leftarrow mg_T(X).$$

$$P(X,Y) \vee P(Y,Z) \leftarrow mg_P(X), A(X,Y), B(Y,Z).$$

$$P(X,Y) \vee P(Y,Z) \leftarrow mg_P(Y), A(X,Y), B(Y,Z).$$

S_2:

$T(X,Y) \leftarrow ST(X,Y).$

$ST(X,Y) \leftarrow SP(X,Y).$

$mg_{SP}(X,Y) \leftarrow SP(X,Y).$

$SP(X,Y) \vee SP(Y,Z) \leftarrow SA(X,Y), SB(Y,Z).$

$mg_{SA}(X,Y) \leftarrow SA(X,Y).$

$SB(X,Y) \leftarrow mg_{SA}(U,X), B(X,Y).$

$mg_{SB}(X,Y) \leftarrow SB(X,Y).$

$SA(X,Y) \leftarrow mg_{SB}(Y,V), A(X,Y).$

6.1.4 Discussion on the rewriting algorithm

From the discussions, it is obvious that we only need to consider those relevant clauses and predicates when we solve the query. When the program is not disjunctive, the transformation degenerates into the original magic-set method for deductive database. Hence, the answer can be obtained accordingly with a bottom-up evaluation. When the program contains disjunctions, the idea of the evaluation method is as follow: we first solve the query in a top-down manner. At some point, a disjunctive fact is found which is partially supporting the atom in the query or subquery. We then have to check if this disjunctive fact could deduce other atom with the same predicate as in the query. For example, suppose the program has the clause:

$B(X) \leftarrow A(X)$

and if the program has the fact $A(1) \vee A(2)$. For the query ?-$B(1)$, when we solve the query in a top-down manner, the fact $A(1) \vee A(2)$ is reached and is partially supporting the fact $B(1)$. At this point, we have to check if $A(2)$ could deduce other atoms in the form $B(x)$, where we could deduce $B(1) \vee B(x)$ for this case. Hence in our rule-rewriting method proposed, the rules generated in S_2 are used to deduce the derived terms which are supported by the disjunctive facts reached when solving the query in the first stage.

A point has to be noted for the rule rewriting algorithm. By the definition of relevant answer set, an answer which is subsumed by another fact should not be deduced. For

example, suppose $A(1) \vee A(2)$ and $A(2)$ are ground facts of a program with the clause:

$$B(X) \leftarrow A(X).$$

the program has only one minimal model $\{ A(1), B(1) \}$. For our rule rewriting algorithm for relevant answer set query, when the query ?-$B(1)$ is posted, the rewritten program will be:

$mg_B(1)$
$B(X) \leftarrow mg_B(X), A(X).$
$B(X) \leftarrow SB(X).$
$SB(X) \leftarrow SA(X).$
$mg_{SA}(X) \leftarrow SA(X).$[1]

By the bottom-up evaluation procedure, clause 2 will stimulate the disjunction $SA(1) \vee SA(2)$ and consequently $SB(1) \vee SB(2)$ is deduced by clause 4. We then get $B(1) \vee B(2)$ as an answer by clause 3. We can see that the answer $B(1) \vee B(2)$ is sound although it is subsumed by $B(2)$.

By the definition of the relevant answer set, the data-disjunctive fact $B(1) \vee B(2)$ should not be deduced as it is not in MMS^P_{DD} since it is subsumed by $B(2)$. However, the answer $B(1) \vee B(2)$ can be viewed as an additional information that related to the query posted.

One can obtain the relevant answer set by the following procedure. For each answer $T_1 \vee T_2 \vee \ldots \vee T_n$ obtained by the rewriting algorithm, we post a query ?-T_i for each T_i which does not match the original query. We then check and remove the answer of the original answer set if it is proper subsumed by the answer of this query. The relevant answer set will be obtained after this removal procedure.

As for the last example, when we get $B(1) \vee B(2)$ as answer, we post the query ?-$B(2)$, and get $B(2)$ by the bottom-up evaluation procedure. This answer $B(2)$ subsumes the disjunctive fact $B(1) \vee B(2)$ of the original answer set and hence $B(1) \vee B(2)$ is removed. Therefore, the relevant answer set for the query ?-$B(1)$ is empty set as expected.

[1] this clause is removed as mg_{SA} is not used in other clauses

Generally, much more computation will be needed for this subsumption checking and removal procedure. In practice, as the answer set obtained are sound and related to the query, this subsumption checking and removal procedure is not necessary to be applied.

6.2 Alternative algorithm for Traditional Answer Set

We have developed the rule rewriting method for the data-disjunctive deductive databases. As argued before, the traditional answer set can be obtained by this rule rewriting method also as the relevant answer set contain the traditional answer set. However, if only the traditional answer set is needed, a simpler rewriting method can be applied and no effort will be wasted to solve for the whole relevant answer set.

6.2.1 Rule rewriting algorithm

In the rule rewriting method for the relevant answer set, the set of rules in S_2 are responsible in the deduction of those atoms which can be generated by the disjunctive facts that are partially supporting an atom match with the query. For the traditional answer set, all atoms in an answer match with the query, hence the deduction of these atoms can be solved by the rules in S_1. Therefore, the set of rules generated in S_2 can be removed and the rules generated are as follow:

INPUT: A set of clauses and a query ?-$T(X_1, \ldots, X_n)$ (or T in short), where X_i is either a variable or a constant.

OUTPUT: A set of clauses such that applying the bottom-up evaluation procedures yields answer to the query.

Let S be the output set of clauses. This set is similar to the output of the rewriting method for relevant answer except that the set S_2 is removed. Hence the set S is generated as follows:

For all relevant clauses

$$P(X_1) \vee \cdots \vee P(X_n) \leftarrow B_1, \ldots, B_m.$$

the clause is viewed as n distinct clauses

$$P(X_i) \leftarrow B_1, \ldots, B_m,\ 1 \leq i \leq m$$

When applying the magic set rewriting method, a corresponding clause

$$P(X_i) \leftarrow \text{a list of magic predicates}, B_1, \ldots, B_m.$$

will be generated and we rewrite these n corresponding clauses by replacing the head part $P(X_i)$ by the original head $P(X_1) \vee \cdots \vee P(X_n)$.

6.2.2 Examples

Example 12 For the program

$$T(X,Y) \vee T(Y,Z) \leftarrow A(X,Y), B(Y,Z).$$
$$?\text{-}T(1,X)$$

is rewritten into:

$$mg_T(1).$$
$$T(X,Y) \vee T(Y,Z) \leftarrow mg_T(X), A(X,Y), B(Y,Z).$$
$$T(X,Y) \vee T(Y,Z) \leftarrow mg_T(Y), A(X,Y), B(Y,Z).$$

Example 13 For the program

$$T(X,Y) \leftarrow P(X,Y).$$
$$P(X,Y) \vee P(Y,Z) \leftarrow A(X,Y), B(Y,Z).$$
$$?\text{-}T(1,X)$$

is rewritten into:

$$mg_T(1).$$
$$T(X,Y) \leftarrow mg_T(X), P(X,Y).$$
$$mg_P(X) \leftarrow mg_T(X).$$
$$P(X,Y) \vee P(Y,Z) \leftarrow mg_P(X), A(X,Y), B(Y,Z).$$
$$P(X,Y) \vee P(Y,Z) \leftarrow mg_P(Y), A(X,Y), B(Y,Z).$$

6.3 Contingency answer set

As stated in Section 5.1, the Contingency answer set for a query Q of a program P is defined by

$$\{A(x) \in CM_P \mid A(x) \Rightarrow \exists Q \}$$

This contingency answer set provides additional information about the uncertainty of those atoms 'match' with the query. From theorem 4.1, the contingency model of a program P can be found by transforming P to P' by replacing each clause

$$A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m.$$

by

$$A_1 \leftarrow B_1, \ldots, B_m.$$
$$A_2 \leftarrow B_1, \ldots, B_m.$$
$$\ldots$$
$$A_n \leftarrow B_1, \ldots, B_m.$$

and then compute the model of this definite program P'. Hence, in order to compute the contingency answer set for a query Q, we can apply the magic set rule rewriting method to the program P' and the query Q to obtain a rewritten program and those atoms in the contingency answer set can be found by a bottom-up evaluation procedure on this rewritten program.

Chapter 7

Experiments and Comparison

We have proposed the data-disjunctive deductive databases which is general enough to express all disjunctive deductive databases with no negated literals. In the presented bottom-up evaluation, the data structure *derived term* is used. This derived term is used to represent the indefinite information and the conditions where the disjunction is supported.

The model tree method uses a tree structure to represent the set of minimal models of the deductive database. On solving a query, the model tree method has to traverse all clauses within the cluster and build the whole model tree. The model tree method does not make use of the information of the query and those propagated by the incremental solving of the subgoals as the magic set rule rewriting method do. Hence, for the model tree method, the complexity of building the model tree for the query ?-$Q(a)$ will be the same as building the model tree for the query ?-$Q(X)$ where all the terms are variables.

7.1 Experimental Results

We have built a prototype for performance evaluation of the proposed evaluation procedures and the magic-set method for the data-disjunctive deductive databases. This prototype is written by C programming language and run on the SPARCstation 20 and SPARCcenter 2000 workstation. The evaluation procedure are tested for the program of transitive closure which have the clauses:

$P(1,2).$	$P(1,3).$	$P(2,4).$	$P(2,5).$	$P(3,6).$
$P(3,7).$	$P(4,8) or P(4,9).$	$P(4,10).$	$P(5,11) or P(5,12).$	$P(5,13).$
$P(6,14).$	$P(6,15).$	$P(7,16) or P(7,17).$	$P(8,18).$	$P(9,19) or P(10,19).$
$P(10,20).$	$P(11,21).$	$P(12,22) or P(12,23).$	$P(13,24) or P(14,24).$	$P(15,25) or P(15,26).$
$P(15,27).$	$P(16,28).$	$P(17,29).$	$P(18,30).$	$P(19,31) or P(20,31).$
$P(21,32).$	$P(22,33).$	$P(26,34).$	$P(28,35).$	$P(29,36).$

Table 7.1: The sample ground facts for relation P

$$T(X,Y) : -P(X,Y).$$
$$T(X,Y) : -T(X,Z), P(Z,Y).$$

This program of transitive closure is chosen because it is the most common recursive predicate and the performance of evaluating information on recursive predicate is critical for deductive databases.

We view the predicate P as the *parent* relation where the predicate T represents the *ancestor* relation. The sample ground facts for the predicate P is listed in Table 7.1 which can be represented by the family tree in Figure 7.1.[1]

We test the performance of the evaluation procedure and the magic-set method for four different cases.[2] These four cases are:

1. Traditional answer set with magic-set method
2. Traditional answer set without magic-set method
3. Relevant answer set with magic-set method
4. Relevant answer set without magic-set method

7.1.1 Results for the Traditional answer set

We first run the experiments on the traditional answer set. For each query, we ask for the descendant of a particular node in the tree. Table 7.2 shows the running time for solving these queries with and without using the magic-set method. In the case without

[1] Jointed arrows represent the corresponding disjunctions

[2] The experiments for both traditional and relevant answer set were run on a SPARCstation 20 machine with 128M memory.

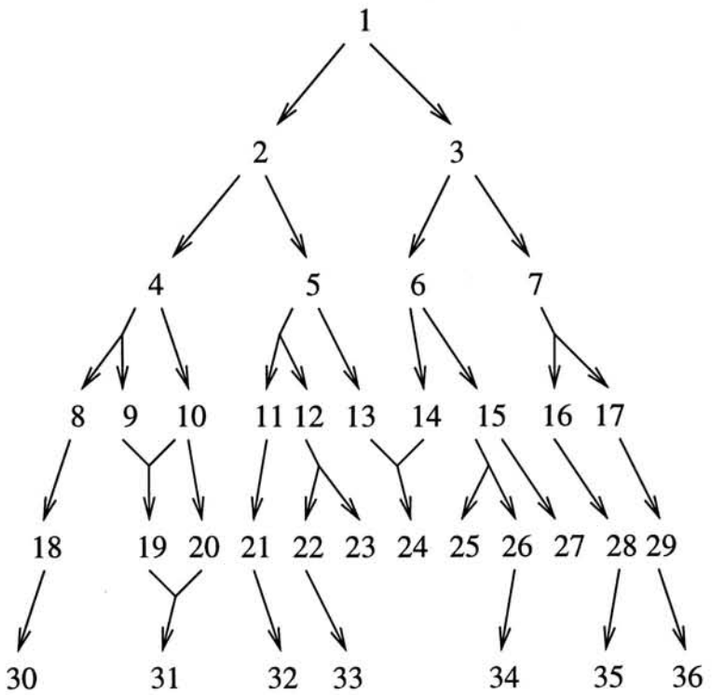


Figure 7.1: Family tree representing the ground facts for relation P

using magic-set method, we use the information on the query to remove those irrelevant derived terms before applying the B operator, i.e. suppose the query is ?-$T(1, Y)$, after applying the F operator we can get:

$T[(1,8), \langle P\{\underline{(4,8)}, (4,9)\}\rangle]$

$T[(4,9), \langle P\{(4,9), \underline{(4,9)}\}\rangle]$

Though the above two terms can be 'joined' to deduce the disjunctive fact $T[(1,8),(4,9)]$, the derived term $T[(4,9), \langle P\{(4,9), \underline{(4,9)}\}\rangle]$ can be removed as the term $T(4,9)$ does not match the query.

The figures in Table 7.2 show that the program using the magic-set method work better than that without. In the magic-set method, we found that the lower the level of n, the shorter the running time for solving the query. This characteristic is found as the magic-set method use the magic predicate in guiding the search space, where when the level of n is lower, less derived terms will be formed and the computation time is shorter accordingly.

Query ?-$T(n, Y)$	with magic (in sec)	without magic (in sec)
n=1	0.12	0.45
n=2	0.06	0.41
n=3	0.03	0.40
n=4	0.02	0.40
n=5	0.02	0.40
n=6	0.02	0.39
n=7	0.02	0.39
n=8	0.01	0.39
n=9	0.01	0.39
n=10	0.01	0.39
n=11	0.01	0.39
n=12	0.01	0.39
n=13	0.01	0.39
n=14	0.01	0.39
n=15	0.01	0.39
n=16	0.01	0.39
n=17	0.01	0.39

Table 7.2: Comparison for Traditional answer set with and without using magic-set method

Generally, the parent relation would store the facts for more than one family tree. In the next two experiments, we study the effect of increasing the number of family trees on the performance of the evaluation method. For simplicity, we assume that the structure of different family trees are identical. In this way, another family tree can be viewed as the same tree as in Figure 7.1 but with different label $A1, A2, \ldots, A36$. We test for four sets where Set 1 is the original case, Set 2 has two such trees, Set 3 has three and Set 4 has four.

Table 7.3 and Table 7.4 show that the running time for solving a query increases when the number of family trees increases. We found that the order of increasing is much smaller for the program that use the magic-set method. This happens because the program with magic-set method use the constant in the query to guide the search space and would not form any derived terms using the facts in the other irrelevant family trees. For the program without using magic-set method, all facts in all the family trees are used in forming the derived terms and the computation take much more time as the number of family trees is increased.

Query ?-$T(n, Y)$	Set 1 (in sec)	Set 2 (in sec)	Set 3 (in sec)	Set 4 (in sec)
n=1	0.12	0.15	0.19	0.22
n=2	0.06	0.08	0.10	0.12
n=3	0.03	0.05	0.06	0.07
n=4	0.02	0.03	0.05	0.06
n=5	0.02	0.03	0.05	0.06
n=6	0.02	0.02	0.03	0.05
n=7	0.02	0.03	0.04	0.04

Table 7.3: Effect of no. of family trees with magic-set method for traditional answer set

Query ?-$T(n, Y)$	Set 1 (in sec)	Set 2 (in sec)	Set 3 (in sec)	Set 4 (in sec)
n=1	0.45	1.54	3.32	5.89
n=2	0.41	1.51	3.31	5.87
n=3	0.40	1.48	3.30	5.84
n=4	0.40	1.49	3.29	5.83
n=5	0.40	1.49	3.29	5.84
n=6	0.39	1.49	3.27	5.82
n=7	0.39	1.49	3.30	5.83

Table 7.4: Effect of no. of family trees without magic-set method for traditional answer set

7.1.2 Results for the Relevant answer set

The following experiments test the running time of solving query for the relevant answer set. Similar to the case for the traditional answer set, for each query, we ask for the descendant of a particular node in the tree. Table 7.5 shows the running time of these queries with and without using the magic-set method. For the program using magic-set method, we only apply the bottom-up procedure for the query once and get all related sound answer without applying the subsumption checking and removal procedure.

The figure shows that the program using magic-set method work better than that without using magic-set method. In the magic-set method, we found that the lower the level of n, the shorter the running time for solving the query. This characteristic is found as the magic-set method use the magic predicate in guiding the search space, where when the

Query ?-$T(n, Y)$	with magic (in sec)	without magic (in sec)
n=1	0.79	22.80
n=2	0.47	22.58
n=3	0.08	22.52
n=4	0.07	22.60
n=5	0.07	22.75
n=6	0.03	22.53
n=7	0.03	22.61
n=8	0.01	22.18
n=9	0.03	22.88
n=10	0.02	22.73
n=11	0.01	22.49
n=12	0.03	22.36
n=13	0.02	22.52
n=14	0.01	22.90
n=15	0.02	22.57
n=16	0.01	22.55
n=17	0.02	22.56

Table 7.5: Comparison for Relevant answer set with and without using magic-set method

level of n is lower, less derived terms will be formed and the computation time is shorter accordingly.

Table 7.6 and Table 7.7 show that the running time for solving a query increase when the number of family trees increase. We found that the order of increasing is very smaller for the program that use the magic-set method. This small increment of computation time is due to the increase in time for the matching of the database as the size of database is increased. For the program without using magic-set method, all facts in all the family trees are used in forming the derived terms and the computation take much more time as the number of family trees is increased. Moreover, we can see that the computation time increase to $(0.9) * i^2$ times approximately as there are i family trees. This order of increasing in time for computation is due to the duplication removal process where a derived term need to compare with the other derived terms.

Query ?-$T(n,Y)$	Set 1 (in sec)	Set 2 (in sec)	Set 3 (in sec)	Set 4 (in sec)
n=1	0.79	0.88	1.00	1.09
n=2	0.47	0.54	0.61	0.68
n=3	0.08	0.13	0.16	0.21
n=4	0.07	0.11	0.14	0.18
n=5	0.07	0.10	0.13	0.17
n=6	0.03	0.05	0.09	0.11
n=7	0.03	0.06	0.08	0.10

Table 7.6: Effect of no. of family trees with magic-set method for relevant answer set

Query ?-$T(n,Y)$	Set 1 (in sec) a	ratio a/a	Set 2 b	ratio b/a	Set 3 c	ratio c/a	Set 4 d	ratio d/a
n=1	22.80	1	82.61	3.6	183.87	8.1	320.70	14.1
n=2	22.58	1	82.83	3.7	182.57	8.1	320.55	14.2
n=3	22.52	1	84.02	3.7	184.02	8.2	320.62	14.2
n=4	22.60	1	83.13	3.7	183.46	8.1	320.43	14.2
n=5	22.75	1	82.85	3.6	184.08	8.1	319.87	14.1
n=6	22.53	1	82.98	3.7	183.24	8.1	320.52	14.2
n=7	22.61	1	83.39	3.7	183.96	8.1	320.57	14.2

Table 7.7: Effect of no. of family trees without magic-set method for relevant answer set

7.2 Comparison with the evaluation method for Model tree

The concept of a minimal model tree for disjunctive deductive databases was introduced in [11] as a structure-sharing approach to represent the set of minimal model of the deductive databases.

Consider a simple disjunctive database $\{ a \vee b, c \vee d \}$. There are only two disjunctive facts in the database but there are four minimal models: {a,c},{a,d},{b,c} and {b,d}. Two possible model trees can be built to represent these models as in Figure 7.2.

There are three problems in using the model tree method for answering query in the disjunctive deductive databases. First, it is obvious that the number of minimal models for a disjunctive deductive databases can be exponential in the size of the database. For a simple case, suppose a disjunctive database (without intensional part) is $\{ a_1 \vee b_1, a_2 \vee$

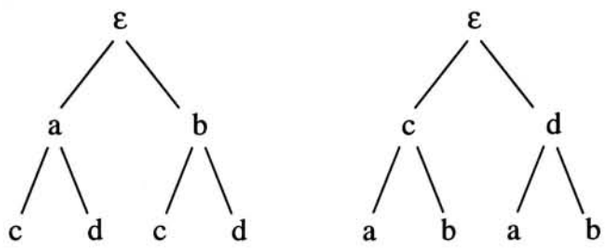


Figure 7.2: Model trees for $\{ a \vee b, c \vee d \}$

$b_2, \ldots, a_n \vee b_n \}$, then the model tree will have 2^n branches. Generation and storage of this model tree will become very inefficient or even impossible as the number of disjunction in the database increase. Second, the model tree method do not make use of the information of the query in restricting the search space. Third, the method incrementally builds the model tree that represents the set of models of the deductive database. Though the model tree keeps all the information on the model of the deductive databases, these information have to be retrieved in order to answer a query. This retrieval of relevant information is inefficient as the number of branches of the model tree increases.

We have tested the performance of the evaluation method using model trees. The model tree method is divided into the following parts:

(a) Build the tree for the ground facts.

(b) Apply the clauses with non-empty body by using the algorithms for the computation on the tree.

(c) Collect the supporting atoms in the model tree to get the answer tree.

(d) Compute the minimal interpretations of this answer tree.

We test the evaluation method on solving query for the traditional answer set and assume that the answer to the query can be obtained once the model tree is obtained. Hence, it is no need to compute and retrieve information to answer the query. The results are shown in Figure 7.8.[3] Since there are eight disjunctive ground facts in one family tree

[3]The program was run on a SPARCcenter 2000 machine with 512M memory.

in our sample, the model tree built for the ground facts for this family tree will have $2^8 = 256$ branches. For the experiment on Set 2, there are $2^{16} = 65536$ branches as two family trees is considered. The experimental result in Figure 7.8 shows the exponential increasing in computation time as the number of branches of the model tree increases. In the experiment of Set 3 and Set 4, the query cannot be solved as memory is not enough for the model tree evaluation method. From the figure, we can also see that the proposed bottom-up evaluation is much more efficient than the model tree method for solving this query.

Method used	Set 1 (in sec)	Set 2 (in sec)	Set 3 (in sec)	Set 4 (in sec)
Model tree method	6.22	9128.19	-	-
Proposed method without magic	0.45	1.54	3.32	5.89
Proposed method with magic	0.12	0.15	0.19	0.22

Table 7.8: Running time for different methods to answer the query ?-$T(1, Y)$

Chapter 8

Conclusions and Future Work

In this thesis, we present the data-disjunctive logic programs. We define the model-theoretic semantic of these data-disjunctive logic programs and present the contingency model which stores the uncertainty information about atoms. We develop a bottom-up evaluation method to compute the model of this logic program and show how to use the computed result in answering queries.

We prove that any disjunctive deductive database can be transformed to a data-disjunctive deductive database and show that the expressive power of the data-disjunctive logic program is the same as the disjunctive logic program. We introduce the concept of standard-form facts which state what kind of facts are interested generally. With the concept of standard-form facts, we can transform a disjunctive deductive database to a data-disjunctive deductive database which the bottom-up evaluation method can be applied. Whenever a query is posted, answer can be computed by means of this evaluation method. In the evaluation procedure, a rank is assigned to every predicate in the program. The data-disjunctive facts can be computed rank by rank by the better structure of the clauses of the data-disjunctive deductive database.

A rule rewriting technique similar to the magic sets techniques is proposed to solve query for the data-disjunctive deductive databases. With this rule rewriting technique, it becomes possible to constrain the computation to the relevant information in the database for a given query. Among our knowledge, this thesis is the first to attempt to make use of the rule rewriting method in the field of disjunctive deductive database.

We have built a prototype for performance evaluation of the proposed bottom-up evaluation procedure and the rule rewriting method for the data-disjunctive deductive databases. We find that, in solving a query, the procedure using the rule rewriting method is much more efficient than the procedure that does not. We also compare our bottom-up evaluation procedure with the model tree method [10]. We find that, for the tested program, the proposed methods are much more efficient than the model tree method.

Future work includes the extension of the data-disjunctive deductive database and evaluation method that can handle negated atom in the body of the clauses. The realization of system based on the rule rewriting method and the bottom-up evaluation procedure is another work to be done. The optimization of the rule rewriting method for the disjunctive deductive databases is also an interesting research topic.

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Database.* Addison-Wesley, 1995.

[2] K.R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148, Washington, D.C., 1988. Morgan Kaufmann Publication.

[3] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 1–15. Cambridge, MA, March 1986.

[4] C. Beeri and R. Ramakishnan. On the power of magic. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 269–283, San Diego, California, March 1987.

[5] G. Bossu and P. Siegel. Saturation, nonmonotonic reasoning and the closed world assumption. *Artificial Intelligence*, 25(1):13–63, January 1985.

[6] Edward P. F. Chan. A possible world semantics for disjunctive databases. *IEEE Transactions on Knowledge and Date Engineering*, 5(2):282–292, April 1993.

[7] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322, New York, 1978. Plenum Press.

[8] S. K. Das. *Deductive Databases and Logic Programming.* Addison-Wesley, 1992.

[9] J. A. Fernandez and J. Minker. Semantics of disjunctive deductive databases. In *International Conference on Database Theory*, pages 21–50, 1992.

[10] J.A. Fernandez, J. Lobo, J. Minker, and V.S. Subrahmanian. Disjunctive LP + integrity constraints = stable model semantics. *Annals of Mathematics and Ariificial Intelligence*, 8(3-4), 1993.

[11] J.A. Fernandez and J. Minker. Bottom-up evaluation of hierarchical disjunctive deductive databases. In K. Furukawa, editor, *Logic Programming Proceedings of the Eighth International Conference.* MIT Press, 1991.

[12] H. Gallaire, J. Minker, and J. M. Nicolas, editors. *Advances in Database Theory, Vol. 1.* Plenum Press, New York, 1981.

[13] H. Gallaire, J. Minker, and J. M. Nicolas, editors. *Advances in Database Theory, Vol. 2.* Plenum Press, New York, 1984.

[14] H. Gallaire, J. Minker, and J. M. Nicolas. Logic and databases: A deductive approach. *Computing Surveys*, 16(2):153–185, 1984.

[15] A. Van Gelder. Negation as failure using tight derivation for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176, Washington, D.C., 1988. Morgan Kaufmann Publication.

[16] L.J. Henschen and H. Park. Compiling the GCWA in Indefinite Databases. *Foundations of Deductive Databases and Logic Programming*, pages 395–438, 1988.

[17] Z. A. Khandaker, J. A. Fernandez, and J. Minker. A tractable class of disjunctive deductive databases. In *Workshop on Deductive Databases, JICSLP*, pages 11–20, 1992.

[18] K. C. Liu and R. Sunderraman. Indefinite and maybe information in relational databases. *ACM Transactions on Databases Systems*, 15(1):1–39, March 1990.

[19] K.L. Liu and R. Sunderraman. On representing indefinite and maybe information in relational databases: A generalization. In *Proceedings of IEEE Data Engineering*, 1990.

[20] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second-edition, 1987.

[21] J. W. Lloyd and R. W. Topor. A basis for deductive database systems. *Logic Programming*, 2(2):93–109, 1985.

[22] J. W. Lloyd and R. W. Topor. A basis for deductive database systems ii. *Logic Programming*, 3(1):55–67, 1986.

[23] J. Lobo, J. Minker, and A. Rajasekar. Extending the semantics of logic programs to disjunctive logic programs. In G. Levi and M. Martelli, editors, *Proc. 6th International Conference on Logic Programming*, pages 255–267, Cambridge, Massachusetts, 1989.

[24] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* The MIT Press, Cambridge, London, England, 1992.

[25] T. C. Przymusinski M. Gelfond, H. Przymusinska. The extended closed world assumption and its relation to parallel circumscription. In *Proceedings of Fifth Symposium on Principles of Database Systems*, pages 133–139, 1986.

[26] J. Minker. On indefinite databases and the closed world assumption. In *Proceedings of 6th Conference on Automated Deduction, New York*, pages 292–308, 1982.

[27] J. Minker and J. Grant. Answering queries in indefinite databases and the null value problem. *Advances in Computing Research*, pages 247–267, 1986.

[28] J. Minker and A. Rajasekar. A fixpoint semantics for disjunctive logic programs. *Logic Programming*, 1990.

[29] F. C. N. Pereira and D. H. D. Warren. Parsing as deduction. In *Proceedings of the Twenty-First Annual Meeting of the Association for Computational Linguistics*, 1983.

[30] T. Przymusinski. On the declarative semantics of deductive databases and logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, Washington, D.C., 1988. Morgan Kaufmann Publication.

[31] R. Ramakrishnan and J. D. Ullman. A survey of deductive database systems. *J. Logic Programming*, 23(2):125–149, May 1995.

[32] J. Rohmer, R. Lescoeur, and J. M. Kerisit. The alexander method - a technique for the processing of recursive axioms in deductive database queries. *New Generation Computing*, 4:522–528, 1986.

[33] C. Sakama. Possible model semantics for disjunctive databases. In *Proceedings of First International Conference on Deductive and Object Oriented Databases*, pages 337–351, 1988.

[34] L. Vieille. Recursive axioms in deductive databases: The query-subquery approach. In *Proceedings of the First International Conference on Expect Database Systems*, pages 179–193, Charleston, SC, 1986.

[35] A. Yahya, J. A. Fernandez, and J. Minker. Ordered Model Trees: A Normal Form for Disjunctive Deductive Databases. *J. Automated Reasoning*, 13(1):117–143, 1994.

[36] A. Yahya and L.J. Henschen. Deduction in Non-Horn Databases. *J. Automated Reasoning*, 1(2):141–160, 1985.

[37] L.Y. Yuan and D.A. Chiang. A sound and complete query evaluation algorithm for relational databases with disjunctive information. In *Proceedings of the Eighth Symposium on Principles of Database Systems*, pages 66–74. ACM Press, March 1989.