The Knowledge Engineering Review, Vol. 4: 4, 1989, 267–304

Deductive database theories*

JOHN GRANT[1] AND JACK MINKER[2]

[1] *Department of Computer and Information Sciences, Towson State University, Towson, Maryland 21204, USA*
[2] *Department of Computer Science and University of Maryland Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742, USA*

Abstract

This paper surveys a variety of deductive database theories. Such theories differ from one another in the set of axioms and metarules that they allow and use. The following theories are discussed: relational, Horn, and stratified in the text; protected, disjunctive, typed, extended Horn, and normal in the appendix. Connections with programming in terms of the declarative, fixpoint, and procedural semantics are explained. Negation is treated in several different ways: closed world, completed database, and negation as failure. For each theory examples are given and implementation issues are considered.

1 Introduction

1.1 Introduction to deductive databases

For nearly a decade after the development of the first widely used database systems in the late 1960s, artificial intelligence and database research followed quite different paths. Artificial intelligence researchers worked on topics such as knowledge representation, natural language processing, and theorem proving, relying primarily on main memory storage and simple file management. Database researchers, on the other hand, concentrated on the efficient storage and retrieval of data in secondary memory, as well as concerns such as concurrency and data security. Within the past decade the area of knowledge engineering emerged, one important component of which is an integration of artificial intelligence and database systems concepts.

A significant aspect in the convergence of artificial intelligence and databases is deductive databases. A history of the subject of deductive databases, going back to 1957, may be found in Minker (1988a). As recounted there, this subject became a separate topic in the mid 1970s with the introduction of logic programming by Colmerauer and Kowalski, the definition of different types of semantics for Horn clause logic in van Emden and Kowalski (1976), and the appearance of major theoretical and practical results in Gallaire and Minker (1978). In the early 1980s Reiter in (Reiter, 1984) proposed formal theories of deductive databases reinterpreting the conventional model-theoretic perspective on databases in purely proof-theoretic form. Also, a comprehensive survey of this field appeared in Gallaire, Minker and Nicolas (1984). Throughout this past decade, various researchers have investigated alternative deductive database theories. Some important recent results about the foundations of deductive databases (and logic programming) appear in Minker (1988).

This paper provides an overview of some deductive database theories and their semantics. Alternative deductive database theories allow different types of statements as axioms for a database. In general, different theories may give different answers to a query and may allow different classes of queries. By knowing which database theory is in use, one may be sure of obtaining precise answers and all the answers to a query. In fact, theoretical developments in deductive databases are

* Research supported by the National Science Foundation under grant numbers IRI-8714544 (both authors) and IRI-8609170 (second author) and by the US Army Research Office under grant number DAAL-03-88-K0087 (second author).

important, because they provide a firm foundation for basic issues involving databases. For each theory discussed, this article provides examples and relevant theorems. A starting point for the implementation of deductive databases is the use of theorem proving techniques. However, typically, more practical implementation techniques are available; there will be some comments about these as the database theories are presented.

1.2 Concepts and definitions from logic

In this paper, databases are formalized within first-order logic. All first-order languages contain a set of infinitely many variables and the standard set of punctuation and logical symbols such as parentheses, comma, connectives, and quantifiers. Different first-order languages are characterized by their non-logical symbols: the constant and predicate symbols. Note that first-order logic allows function symbols also, and these are important in logic programming, but we apply the assumption that deductive databases are function-free. Terms, atomic formulas, and formulas of a language are defined in the standard way.

The meaning of terms and formulas may be defined by means of interpretations. An *interpretation* consists of a domain of objects, D, and an interpretation of the non-logical symbols in D, so that a constant symbol is interpreted as an element of D and a predicate symbol is interpreted as a relation on D. (Function symbols in first-order logic are interpreted as functions on D, but are not considered further in this paper.) Intuitively, an interpretation assigns a meaning to the symbols of the language. In fact, an interpretation implicitly assigns a truth-value to all sentences (formulas without free variables) of the language. Given a set of sentences S, an interpretation I is a *model* of S if every element of S is true in I. A sentence w is a *logical consequence* of a set of sentences S, written $S \models w$, if w is true in all models of S.

An *inference system* (proof theory) for first-order logic is a set of axiom schemas and rules of inference that provide proofs of formulas. $S \vdash w$ is written if there is a proof of w from S. An inference system is *complete* in case $S \vdash w$ iff $S \models w$. Completeness is an important concept equating the capability of a well-defined inference system with the concept of logical consequence that refers to all models. A standard complete inference system uses a set of logical axiom schemas and the two inference rules: modus ponens and generalization. However, most theorem proving procedures are based on the resolution principle, an inference system used for refutation proofs: $S \vdash w$ is shown by deriving the empty clause (contradiction) from the clausal forms of S and $\neg w$. Resolution refutation is also complete. Resolution uses *clauses*, which are disjunctions of literals, where a *literal* is either an atomic formula or its negation. Clauses are usually written in the form

$$\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m,$$

where the A_i and B_j are atoms. An equivalent formulation,

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

is the one used in logic programming and deductive databases. Such a clause can be interpreted in two ways: (1) if $A_1, \ldots, A_n$ are all true, then at least one of $B_1, \ldots, B_m$ is true; (2) to solve for $B_1 \vee \ldots \vee B_m$, solve all of $A_1, \ldots, A_n$. The atoms $A_1, \ldots, A_n$ form the *body* of the clause; $B_1, \ldots, B_m$ form the *head*. A *ground* clause contains no variables. A clause of the form $B_1, \ldots, B_m \leftarrow$ represents a fact; the fact is *definite* if $m = 1$ and *indefinite* if $m > 1$. The empty clause, $\leftarrow$, represents a contradiction. A clause of the form $\leftarrow A_1, \ldots, A_n$ is a *query* clause. While this paper is self-contained, the reader is referred to Enderton (1972) and Mendelson (1978) for background on first-order logic, Chang and Lee (1973) and Loveland (1978) for additional material on resolution theorem-proving, and Lloyd (1987) for a thorough study of logic programming.

In some cases it is convenient to allow negated atoms in the body of a clause. Even if the database does not contain such clauses, the capability is important for queries in general. This paper deals with queries $Q(x_1, \ldots, x_m)$ of the form

$$\exists x_1 \ldots \exists x_m (L_1 \,\&\, \ldots \,\&\, L_n),$$

where the x_i, $1 \leqslant i \leqslant m$, are the variables in Q, and the L_i, $1 \leqslant i \leqslant n$, are the literals. In logic programming such a query is written as $\leftarrow L_1, \ldots, L_n$. Note that the notion of a query clause defined above is less general because there the A_i must all be atoms. More general queries involving universal quantifiers, disjunction, and aggregates are not considered in this paper. A (definite) *answer* to Q has the form $\langle a_1, \ldots, a_m \rangle$ such that $Q(a_1, \ldots, a_m)$ (meaning the substitution of a_i for x_i, $1 \leqslant i \leqslant m$) is provable from the database. A query without free variables represents a yes–no query.

1.3 Contents of this article

Several database theories are explained in the survey paper (Gallaire, Minker and Nicholas, 1984). The paper presented here is a substantially revised and expanded version of (Minker, 1987). The main emphasis is on covering the major alternative theories for deductive databases. Some of the major topics about deductive databases, such as the efficient handling of recursion, view updates, and semantic query optimization, are not discussed here. The 2 volume set (Ullman, 1988) contains much material on deductive databases, particularly on the handling of recursion.

The body of this paper contains explanations of the concepts and the main results concerning the semantics of Horn and stratified databases. These notions and theories are widely accepted and incorporated successfully in many working systems. The Appendix contains similar information on other types of databases that are of great research interest at the present time. The contents of the sections are as follows. Section 2 contains the definition of a deductive database as well as a set of basic axioms for deductive databases. The relational and Horn databases are defined here. Three major semantics: declarative, fixpoint, and procedural, are discussed for Horn theories in section 3. Three ways of dealing with negation: Closed World Assumption, Completed Database, and Negation as Finite Failure, are given in section 4. Stratified databases, an important subclass of normal databases, are the subject of section 5. Section 6 contains the conclusions and summary. The Appendix consists of three sections. Protected databases are discussed in section A1. Section A2 considers semantics, including negation, for disjunctive databases. Section A3 contains briefer discussions of typed databases, extended Horn databases, and the well-founded approach to semantics for normal databases. The references are grouped by topic.

2 Relational and deductive Horn databases

The first description of the proof theoretic and model theoretic approaches to databases appeared in Nicolas and Gallaire (1978). Reiter in Reiter (1984) was the first propose formal theories of databases by reinterpreting the conventional model theoretic perspective on databases in purely proof theoretic terms. In the model theoretic approach a database is a model for axioms (statements). This declarative semantics will be considered further in the next section. In the proof theoretic approach, deductive databases, encompassing relational databases, become special types of theories of first-order logic. This section contains a definition of deductive databases from the proof theoretic point of view. In this framework Horn databases (including the special case of relational databases) are considered.

2.1 Basic concept of deductive databases

A deductive database may be defined as a triple $DB = \langle C, P, I \rangle$. C is a finite set of nonlogical symbols (constant and predicate) that define a specific first-order language. It is assumed that C has at least one constant symbol and at least one predicate symbol. P consists of a finite set of axioms in the language and may contain metarules (that is, rules not expressible in the language) as well. I is a finite set of sentences in the language, the integrity constraints, that must be satisfied by the database. Updates to the database typically involve P only; the updated database must still satisfy I. In first-order logic a set of sentences (in a given language) is also called a theory, hence a specific database may also be referred to as a theory.

Although I contains much of the knowledge of a deductive database, the main concern of this paper is the set P. (See Grant and Minker 1990) for the uses of integrity constraints in deductive databases.) Alternative database theories differ in the types of formulas allowed in P and in the metarules of P. However, there is a basic set of axioms in P: the domain closure axiom, the unique name axioms, and the equality axioms. These axioms assure that the database is finite, all constants can be named, different constant symbols stand for different objects, and equality stands for the identity relation. Some of the theorems presented throughout this article hold under less restrictive assumptions as well, such as without domain closure or with function symbols in the language. Those results will not be specifically mentioned here, but may be found in the references.

The domain closure axiom has the form

$$\forall x(x = c_1 \vee \ldots \vee x = c_n),$$

where $c_1, \ldots, c_n$ are all the constant symbols in C. In words, every element must be named by a constant symbol; no unnamed objects are allowed.

The unique name axioms are

$$\neg c_1 = c_2$$
$$\neg c_1 = c_3$$
$$\vdots$$
$$\neg c_{n-1} = c_n,$$

where, again, $c_1, \ldots, c_n$ are the constant symbols in C. These statements assure that no two constant symbols represent the same object.

The equality axioms are

$\forall x(x = x)$ reflexivity

$\forall x \forall y(x = y \rightarrow y = x)$ symmetry

$\forall x \forall y \forall z(x = y \,\&\, y = z \rightarrow x = z)$ transitivity

$\forall x_1 \ldots \forall y_n(R(x_1, \ldots, x_n) \,\&\, x_1 = y_1 \,\&\, \ldots \,\&\, x_n = y_n \rightarrow R(y_1, \ldots, y_n))$ substitution for all predicate symbols R.

These axioms formalize the standard properties of equality.

As explained in Gallaire, Minker and Nicolas (1984) these axioms are typically not needed for an operational definition of deductive databases where a metarule is given for dealing with negation. Here it is useful to provide these axioms for a better understanding of the underlying theory. With the assumption that the domain closure, unique name, and equality axioms are in P, we concentrate on the rest of the axioms and classify database theories by the types of formulas allowed for them. In fact, from now on the basic axioms will be ignored and P used to stand for the rest of the axioms.

Generally, two examples will be given for each type of database theory. (One theory type may be a special case of another.) The first example will be a practical example, informally presented. The second example will be a small abstract example; the results about it will be worked out. Initially, only query clauses, that is clauses of the form $\leftarrow A_1, \ldots, A_n$ where all the A_i are atoms, will be considered for queries. Queries involving negation will be discussed starting in section 4.

2.2 Relational databases

Relational databases are studied first. Relational databases are a special case of deductive databases that do not allow deductive rules. In this case, P consists of ground atomic formulas only. These formulas represent database facts: tuples in relations (rows in database tables). Practical examples of relational databases are numerous and books on databases contain many examples. Some typical relations include Supplier, Part, Employee, and Department, with appropriate tuples, such as the 4-tuple ⟨Jones H., 31, secretary, 19000⟩ in the Employee relation, indicating an employee's name, age,

title, and salary. No specific example will be given here. The following abstract example is used for illustration.

Example 1 A relational database
C contains the constants a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, the unary predicate symbol R_1, the binary predicate symbol R_2, and the ternary predicate symbol R_3.

P contains

$R_1(a_1) \leftarrow$
$R_1(a_2) \leftarrow$
$R_2(a_1, a_3) \leftarrow$
$R_2(a_2, a_4) \leftarrow$
$R_3(a_1, a_5, a_7) \leftarrow$
$R_3(a_2, a_3, a_4) \leftarrow$
$R_3(a_8, a_1, a_7) \leftarrow$.

The tables in the corresponding relational database are R_1 with the elements a_1 and a_2, R_2 with the pairs $\langle a_1, a_3\rangle$ and $\langle a_2, a_4\rangle$, and R_3 with the triples $\langle a_1, a_5, a_7\rangle$, $\langle a_2, a_3, a_4\rangle$, and $\langle a_8, a_1, a_7\rangle$. Consider now the following queries:

1.1 $\leftarrow R_1(x)$ (Find the x's for which the relation $R_1(x)$ is true.)
There are two answers: a_1 and a_2, since $R_1(a_1)$ and $R_1(a_2)$ are facts.
1.2 $\leftarrow R_2(x_1, x_2), R_3(x_1, x_3, x_2)$
There is one answer: $\langle a_2, a_4, a_3\rangle$, since $R_2(a_2, a_4)$ and $R_3(a_2, a_4, a_3)$ are facts.
1.3 $\leftarrow R_1(a_3)$
The answer is No, since $R_1(a_3)$ is not a fact.

Relational databases are the simplest in the classification. Such theories are very important because of the prominence of commercially available relational database systems, and the development of many sophisticated concepts, both theoretical and applied, for relational databases (Ullman, 1988). While only facts can be represented directly in relational databases, Horn databases allow for the representation of both facts and rules. This yields a significant advantage in expressive power, as rules provide a concise way of expressing knowledge and are particularly useful in knowledge based systems.

2.3 Horn databases

In a *Horn* database, P consists of formulas of the form $B \leftarrow A_1, \ldots, A_n$ where B, $A_1, \ldots, A_n$, are atoms (n may be 0). The non-atomic Horn formulas represent database rules. These rules constitute the deductive part of the database. Horn databases in the sense of this paper, that is, function-free, are also referred to as DATALOG programs (Ullman, 1988).

An example of a Horn database is the example involving family relationships. The predicate parent is given in the form of ground atoms, such as parent(mary,jim) $\leftarrow$. Horn definitions can then be given to define concepts such as grandparent and ancestor.

grandparent(x,y) $\leftarrow$ parent(x,z),parent(z,y)
ancestor(x,y) $\leftarrow$ parent(x,y)
ancestor(x,y) $\leftarrow$ parent(x,z),ancestor(z,y).

In words, a grandparent is a parent of a parent; an ancestor is either a parent or an ancestor of a parent. A definition such as the one for ancestor is called *recursive* because it appears both on the left and right hand side, that is, ancestor is defined partially in terms of itself.

The following abstract example is an illustration of Horn databases.

Example 2 A Horn database
C contains the constants $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$, the binary predicate symbols R_1, R_2, and the ternary predicate symbol R_3.

P contains

$R_1(a_1, a_3) \leftarrow$
$R_1(a_3, a_4) \leftarrow$
$R_1(a_4, a_6) \leftarrow$
$R_2(x, y) \leftarrow R_1(x, y)$
$R_2(x, y) \leftarrow R_1(x, z), R_2(z, y)$
$R_3(a_1, a_2, a_3) \leftarrow$
$R_3(x, y, z) \leftarrow R_1(x, y), R_2(y, z).$

Note that R_2 has a recursive definition and that the contents of R_3 are described in terms of a fact and rule. To make the example more meaningful, one may think of R_1 as the parent predicate, R_2 as the ancestor predicate, and R_3 as a ternary predicate that is a special combination of parent and ancestor.

Consider now the following queries.

2.1 $\leftarrow R_2(x, y)$
There are six answers: $\langle a_1, a_3\rangle, \langle a_3, a_4\rangle, \langle a_1, a_4\rangle, \langle a_4, a_6\rangle, \langle a_3, a_6\rangle, \langle a_1, a_6\rangle$.
2.2 $\leftarrow R_3(a_1, x, y)$
There are three answers: $\langle a_2, a_3\rangle, \langle a_3, a_4\rangle, \langle a_3, a_6\rangle$.
2.3 $\leftarrow R_2(a_2, x), R_1(x, y)$
There is no answer.

3 Declarative, fixpoint, and procedural semantics

Semantics deals with meaning. The previous section presented two specific examples, a relational database and a Horn database. For each database, several questions, along with answers, were given. The answers appear to be correct intuitively. In this section a more formal approach is taken to determine the meaning of a deductive database and the correctness of an answer to a query. Three standard semantics are discussed: declarative, fixpoint, and procedural. The main result of this section is that these differently defined semantics yield identical results for Horn databases. The theorems of this section come from van Emden and Kowalski (1976).

3.1 Declarative semantics for Horn databases

Declarative semantics is based on interpretations, as discussed in section 1. Within the framework of the basic axioms given in the previous section, the domain of every model must contain a distinct element for each constant symbol. But in logic programming in general, or if some of the basic axioms are not included, there may be many different domains for a theory. There is one domain that is particularly useful. The *Herbrand universe* for a set of formulas (axioms) is the set of all symbols built up using the constant symbols (and function symbols, if any). An interpretation whose domain is the Herbrand universe is called a *Herbrand interpretation.* A Herbrand interpretation for a set of sentences which is also a model (that is, in which the sentences are all true) is called a *Herbrand model.* In logic programming the Herbrand universe may be infinite because of a function symbol or infinitely many constant symbols. Using the definition of a deductive database from section 2 including the basic axioms, all models are Herbrand models modulo renaming the elements of the domain.

Another useful concept is the notion of a *Herbrand base*, HB_C: the set of all ground atomic formulas that can be formed using C. HB_C may also be thought of as the set of all possible facts about the database. For a deductive database, with a finite set of constants and predicates, the

Herbrand base is finite. The Herbrand base is always a Herbrand model for a Horn database, because it satisfies the axioms of P including the fundamental axioms. However, the Herbrand base is usually not the intended model: it is too big. In general, we do not intend all possible facts to be true. The idea is to look for small subsets of the Herbrand base that are Herbrand models, in order to make the least number of assumptions concerning what is true in the database. Theorem 2 will justify this restriction.

For Horn databases there is a unique smallest Herbrand model, the minimal model, by the following theorem.

Thorem 1 (van Emden and Kowalski, 1976). The intersection of every (non-empty) set of Herbrand models for a Horn database is a Herbrand model.

Hence it suffices to take the intersection of all Herbrand models, M_P, to obtain the intended meaning of the deductive database. The following theorem provides an important property of M_P.

Theorem 2 (van Emden and Kowalski, 1976). For a Horn database, $M_P = \{A \in HB_C \mid P \models A\}$.

This theorem states that the minimal model contains exactly the atoms logically implied by P.

Now consider Example 1. For this case,

$$M_P = \{R_1(a_1), R_1(a_2), R_2(a_1, a_3), R_2(a_2,a_4), R_3(a_1, a_5, a_7), R_3(a_2, a_3, a_4), R_3(a_8, a_1, a_7)\}.$$

In general, for a relational database, M_P represents exactly the rows in the tables. Now consider the queries 1.1–1.3. Assuming that M_P is the intended meaning of the database, the answers are constants whose substitutions make the queries true in M_P. For instance, considering 1.1, $M_P \models R_1(a_1)$ and $M_P \models R_1(a_2)$, but for any other constant $c \in C$, it is not the case that $M_P \models R_1(c)$. Hence the answers are a_1 and a_2.

Next, consider Example 2. In this case,

$$M_P = \{R_1(a_1, a_3), R_1(a_3, a_4)\ R_1(a_4, a_6), R_2(a_1, a_3), R_2(a_3, a_4), R_2(a_4, a_6), R_2(a_1, a_4),$$
$$R_2(a_1, a_6), R_2(a_3, a_6), R_3(a_1, a_2, a_3), R_3(a_1, a_3, a_4), R_3(a_1, a_3, a_6)\}.$$

So, for 2.2, $M_P \models R_3(a_1, a_2,a_3)$ & $R_3(a_1, a_3, a_4)$ & $R_3(a_1, a_2, a_6)$, but for any other constants c_1, c_2, it is not the case that $M_P \models R_3(a_1, c_1, c_2)$. That is how the three answers are obtained.

3.2 Fixpoint semantics for Horn databases

The second type of semantics is called fixpoint semantics. This type of semantics involves the building of the intended Herbrand model in a step-by-step process using Herbrand interpretations. The starting point is the empty set. At each step the rules of the database imply the addition of new atoms to the existing Herbrand interpretation. For instance, if $A_1, \ldots, A_n$ are in a Herbrand interpretation of P and $A \leftarrow A_1, \ldots, A_n$ is a clause in P, then A should be added to the Herbrand interpretation to obtain a new Herbrand interpretation. When no more additions are needed, a fixpoint is reached. Such a fixpoint is a Herbrand model and the goal of fixpoint semantics is to compute the smallest fixpoint as the intended Herbrand model.

The general concepts involve the mathematical theory of lattices. A *lattice* is a set with a partial ordering (essentially a less than or equal to: $\leqslant$) relation. For a lattice L and a set $X \subseteq L$, $a \in L$ is called an *upper bound* of X if $x \leqslant a$ for all $x \in X$. A *least upper bound* a is an upper bound such that $a \leqslant a'$ for all upper bounds a'. If the least upper bound of X exists, it must be unique and is denoted by *lub*(X). The notions of *lower bound* and *greatest lower bound* are defined in a similar but opposite manner. A lattice L is called *complete* if lub(X) and glb(X) *exist* for every subset X of L.

The set of all subsets of a set S is called the *power set* of S and is written as 2^S. For the application to database semantics the set S should be thought of as HB_C, the Herbrand base. Then 2^S is the set of all Herbrand interpretations. Under the subset relation 2^S is known to be a complete lattice with

lub(X) = $\cup S_i \{S_i \in X\}$ and glb(X) = $\cap S_i \{S_i \in X\}$. The top element is S and the bottom element is $\varnothing$ (the empty set). The crucial aspect of the theory involves properties of transformations between Herbrand interpretations; these are mappings from 2^S to 2^S in the present setup. A mapping T: $2^S \to 2^S$ is *monotonic* if for elements I, J of the lattice I $\subseteq$ J implies T(I) $\subseteq$ T(J), and *continuous* if T(lub(X)) = lub(T(X)) for every directed subset X of 2^S. (A subset of L is *directed* if it contains an upper bound for every finite subset.)

The powers of a monotonic mapping T are defined as follows:

$T \uparrow 0 = \varnothing$ (The 0th power of T is the empty set.)
$T \uparrow i + 1 = T(T \uparrow i)$ (The next power of T is obtained by applying T to the previous power.)
$T \uparrow \omega = \text{lub}\{T \uparrow i \mid i < \omega\}$ (The ωth power of T is obtained by taking the lub of all finite powers.)

An element I $\in 2^S$ is called a *fixpoint* of T if T(I) = I, that is, T does not change I. The *least fixpoint* of T is written as *lfp*(T), and is defined as the fixpoint which is a subset of every fixpoint. In general, lfp(T) need not exist. The following two results are well-known about lattices (see Lloyd, 1987)):

(1) If T is monotonic, then lfp(T) exists.
(2) If T is continuous, then lfp(T) = $T \uparrow \omega$.

As mentioned earlier, the connection between monotonic mappings on a power set and deductive databases is obtained by letting $S = HB_C$, the Herbrand base, in which case 2^S is the set of Herbrand interpretations. The mapping is usually referred to as T_P, where P is the program, which, in this case, is the set of non basic axioms of P.

For a Herbrand interpretation I,

$$T_P(\text{I}) = \{\text{A} \in HB_C \mid \text{A} \leftarrow A_1, \ldots, A_n \text{ is a ground instance of a clause in P and } \{A_1, \ldots, A_n\} \subseteq \text{I}\}.$$

T_P(I) contains all immediate consequences of the rules of P applied to I. The following theorem is a crucial result.

Theorem 3 (van Emden and Kowalski, 1976). If the clauses of P are Horn then T_P is continuous.

Theorem 3 and Result (2) above imply that lfp(T_P) = $T_P \uparrow \omega$. Fixpoint semantics picks lfp(T_P) as the meaning of P. The following computations yield $T_P \uparrow \omega$ for the two examples given in the previous section.

For Example 1,

$T_P \uparrow 0 = \varnothing$
$T_P \uparrow 1 = T_P(T_P \uparrow 0) = \{R_1(a_1), R_1(a_2), R_2(a_1, a_3), R_2(a_2, a_4), R_3(a_1, a_5, a_7), R_3(a_2, a_3, a_4), R_3(a_8, a_1, a_7)\}$
$T_P \uparrow n = T_P \uparrow 1$ for all $n \geqslant 1$
$T_P \uparrow \omega = \text{lub}\{T_P \uparrow i \mid i < \omega\} = T_P \uparrow 1.$

For Example 2,

$T_P \uparrow 0 = \varnothing$
$T_P \uparrow 1 = T_P(T_P \uparrow 0) = \{R_1(a_1, a_3), R_1(a_3, a_4), R_1(a_4, a_6), R_3(a_1, a_2, a_3)\}$
$T_P \uparrow 2 = T_P(T_P \uparrow 1) = T_P \uparrow 1 \cup \{R_2(a_1, a_3), R_2(a_3, a_4), R_2(a_4, a_6)\}$
$T_P \uparrow 3 = T_P(T_P \uparrow 2) = T_P \uparrow 2 \cup \{R_2(a_1, a_4), R_2(a_3, a_6), R_3(a_1, a_3, a_4), R_3(a_3, a_4, a_6)\}$
$T_P \uparrow 4 = T_P(T_P \uparrow 3) = T_P \uparrow 3 \cup \{R_2(a_1, a_6), R_3(a_1, a_3, a_6)\}$
$T_P \uparrow n = T_P \uparrow 4$ for all $n \geqslant 4$
$T_P \uparrow \omega = \text{lub}\{T_P \uparrow i \mid i < \omega\} = T_P \uparrow 4.$

Note that in both cases, $T_P \uparrow \omega = M_P$. In fact, the following theorem shows the equivalence of the declarative and fixpoint semantics.

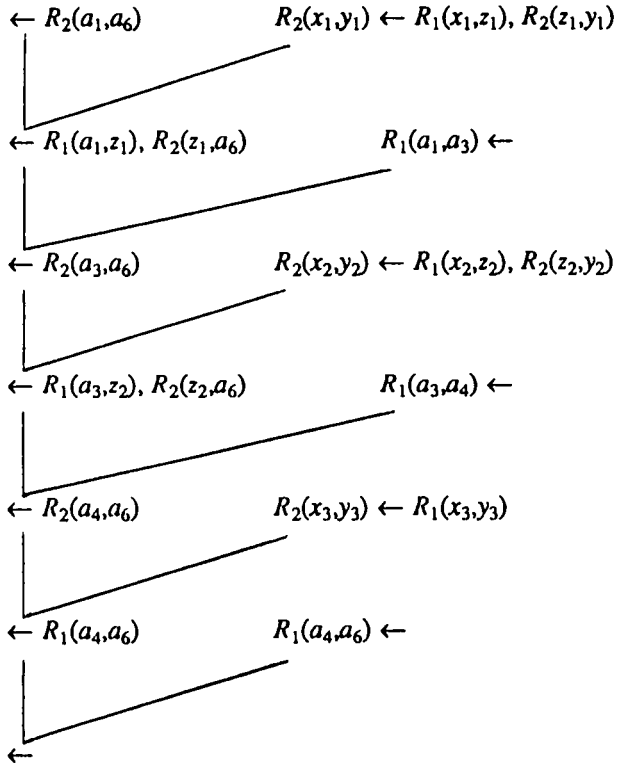


Figure 1

Theorem 4 (van Emden and Kowalski, 1976). For a Horn database, $T_P \uparrow \omega = M_P$.

3.3 Procedural semantics for Horn databases

The third type of semantics, procedural semantics, refers to a computational method for obtaining the meaning of a deductive database. For Horn databases SLD-resolution is used as procedural semantics. In an SLD-refutation, a Horn database P and a goal (clause of the form $\leftarrow B_1, \ldots, B_m$) are given. An SLD-derivation starts with the goal clause as the top clause that is resolved in a linear manner with clauses in P (each clause of P is given new variables not previously used in the derivation). An atom in the present goal, B_i, is selected, and a clause in P is chosen whose head can unify with B_i by a substitution; the new goal is obtained by replacing B_i with the body of the clause after applying the substitution required for the unification. An SLD-refutation is an SLD-derivation that ends with the empty clause: ← (also written as [ ]).

The success set of P, *Succ(P)*, is defined as the set of all $A \in HB_C$ such that $P \cup \{\leftarrow A\}$ has an SLD-refutation. In Example 1 it is easy to see that A can be in Succ(P) if and only if it is an axiom. Now consider Example 2 and take the atom $R_2(a_1, a_6)$. Figure 1 is an SLD-resolution which shows that $R_2(a_1, a_6) \in$ Succ(P).

The following theorem shows the connection between the declarative and procedural semantics.

Theorem 5 (van Emden and Kowalski, 1976). For a Horn database (using SLD-refutation) M_P = Succ(P).

Theorem 6 follows from Theorems 4 and 5.

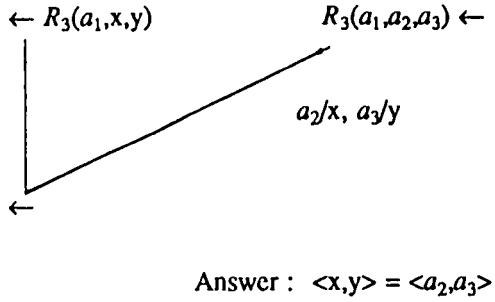
Theorem 6 (van Emden and Kowalski, 1976). For a deductive Horn database $M_P = T_P \uparrow \omega$ = Succ(P).

This theorem shows the equivalence of the declarative, fixpoint, and procedural semantics for the meaning of P.

3.4 Query answering in Horn databases

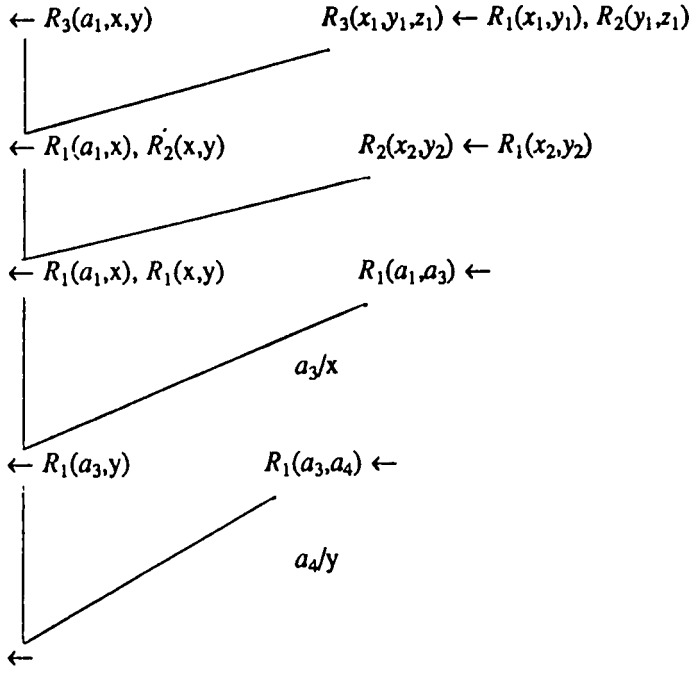
The next objective is to show that these three types of semantics produce identical answers to queries. In the case of the declarative and fixpoint semantics, the equivalence of M_P and $T_P \uparrow \omega$ and the use of logical implication assure identical answers to queries. For procedural semantics we use SLD-resolution to obtain answers. Consider the query 2.2. An answer to this query is obtained by using an SLD-refutation for P starting with $\leftarrow R_3(a_1, x, y)$ as the top clause. The substitutions for x and y provide an answer. In particular, the three SLD-refutations in Figures 2–4 provide the three answers.

The final result of this section shows the query equivalences of the three semantics. It is necessary to take care of the case where an SLD-refutation does not bind some query variables. In such a case all substitutions of constants in C for the variables are considered as solutions. For example, if P contains the clause R(x) ← then the standard SLD-refutation for the query ← R(x) does not produce



Answer : <x,y> = <a_2,a_3>

Figure 2



Answer : <x,y> = <a_3,a_4>

Figure 3

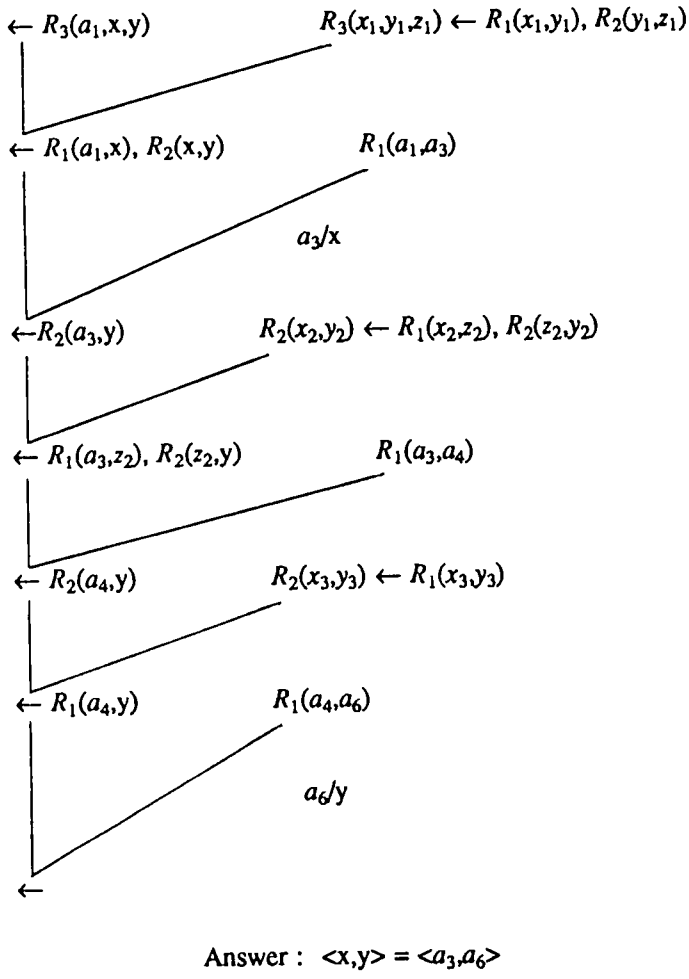


Figure 4

a constant substitution for x. This is taken to mean that all constants in C are solutions. The following theorem should be understood with this proviso.

Theorem 7 (van Emden and Kowalski, 1976). For a Horn database the declarative, fixpoint, and procedural semantics provide identical answers to queries.

Theorems 6 and 7 are highly significant both theoretically and practically. Theoretically these theorems shows the equivalence of three completely differently defined semantics. Declarative semantics is not a practical method for finding the intended Herbrand model because it involves finding all Herbrand models, but it is attractive because it finds the least Herbrand model intuitively. However, fixpoint semantics does provide a practical method, one that in expert systems technology is usually called bottom-up or forward chaining. Finally, procedural semantics is a top-down or backward-chaining technique. So these theorems show the equivalence of a forward chaining and a backward chaining technique for finding the intuitively intended Herbrand model for answering queries in Horn databases.

4 Negation

Negation has been problematic for databases because the volume of negative data is usually much greater than the volume of positive data, so that the storage of negative data might overwhelm a database system. Consequently, databases typically store only positive data. This section explores

the three main methods for dealing with implicit negation in Horn databases: the *Closed World Assumption*, the *Completed Database*, and *Negation as Finite Failure*.

Although storing negative data may not be practical, it is theoretically possible. In this approach, called the *Open World Assumption* (*OWA*) (Reiter, 1978), both positive and negative information must be supplied to the system. Such a theory is based on full first-order logic. So, for a query $Q(x_1, \ldots, x_n)$, $\langle a_1, \ldots, a_n \rangle$ is a (definite) answer if DB $\vdash Q(a_1, \ldots, a_n)$, where provability is within full first-order logic. While conceptually simple, the Open World Assumption is not a practical approach to negation because of the volume of negative data and will not be considered any further here.

4.1 Closed World Assumption

In (Reiter, 1978) an alternative approach, called the *Closed World Assumption* (*CWA*), is proposed. The CWA is a metarule that is used in addition to the axioms of the database. The assumption is that all positive information is known and specified, hence any positive ground atom that cannot be proved from the database is taken to be false. Formally, if A is a ground atom, then $\neg$A is inferred from P if A is not in T where T = {B | B is a ground atom and P $\nvdash$ B}. The semantics discussed in the previous section conform to the CWA. The new queries for Examples 1 and 2 illustrate the application of the CWA.

1.4 $\leftarrow \neg R_1(x)$

By the CWA the answers are a_3, a_4, a_5, a_6, a_7, and a_8, since only $R_1(a_1)$ and $R_1(a_2)$ are facts.

1.5 $\leftarrow R_3(x_1, x_2, x_3), \neg R_1(x_1)$

By the CWA the answer is $\langle a_8, a_1, a_7 \rangle$, since $R_3(a_8, a_1, a_7)$ is a fact and $R_1(a_8)$ is not a fact.

2.4 $\leftarrow \neg R_2(a_1, x)$

By the CWA the answers are a_1, a_2, a_5, a_7, and a_8.

2.5 $\leftarrow \neg R_3(a_4, a_3, a_4)$

The CWA answer is Yes.

The CWA is very powerful because it determines a unique Herbrand model for Horn databases.

4.2 Completed Database

Another approach to the handling of negation is called the *Completed Database* (*CDB*) (Clark, 1978). According to the CDB, the facts and rules of P provide the if parts of the definitions of predicates. However, what is really meant is both the if part (the explicit definition) and the only-if part (implicitly). Recall, for instance, the definition of the grandparent predicate from Section 2.3:

grandparent(x,y) $\leftarrow$ parent(x,z), parent(z,y).

This definition, in English, says that y is a grandparent of x if for some z y is a parent of z and z is a parent of x. However, the definition leaves open the possibility that y may be a grandparent of x for other reasons. What is meant here is that if y is a grandparent of x then there must be a z such that y is a parent of z and z is a parent of x, in other words,

grandparent(x,y) $\rightarrow \exists$z(parent(x,z),parent(z,y)).

The grandparent predicate is defined by a single clause. When a predicate R is defined by several clauses, what is meant is that R holds when (if and only if) the conditions of R's definition are satisfied. Formally, the CDB is defined as follows:

Each Horn clause of the form

$$R(t_1, \ldots, t_n) \leftarrow A_1, \ldots, A_m$$

containing the variables $y_1, \ldots, y_p$, is rewritten as

$$R(x_1, \ldots, x_n) \leftarrow (\exists y_1) \ldots (\exists y_p)(x_1 = t_1 \,\&\, \ldots \,\&\, x_n = t_n \,\&\, A_1 \,\&\, \ldots \,\&\, A_m),$$

where $x_1, \ldots, x_n$ are new variables. For each R, call the right-hand sides of the definitions of R: $E_1, \ldots, E_r$. Then the completed database contains, in addition to P, for each R the formula

$$R(x_1, \ldots, x_n) \rightarrow E_1 \vee \ldots \vee E_r.$$

(In the special case where there are no definitions for R, the CDB yields $\neg R(x_1, \ldots, x_n)$.) A negative query is answered in the CDB by invoking a theorem prover. The CDB is illustrated by listing the clauses of the completed database for Examples 1 and 2.

The CDB for Example 1.

$$R_1(a_1) \leftarrow$$
$$R_1(a_2) \leftarrow$$
$$R_2(a_1, a_3) \leftarrow$$
$$R_2(a_2, a_4) \leftarrow$$
$$R_3(a_1, a_5, a_7) \leftarrow$$
$$R_3(a_2, a_3, a_4) \leftarrow$$
$$R_3(a_8, a_1, a_7) \leftarrow$$
$$R_1(x_1) \rightarrow x_1 = a_1 \vee x_1 = a_2$$
$$R_2(x_1, x_2) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_3) \vee (x_1 = a_2 \,\&\, x_2 = a_4)$$
$$R_3(x_1, x_2, x_3) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_5 \,\&\, x_3 = a_7) \vee (x_1 = a_2 \,\&\, x_2 = a_3 \,\&\, x_3 = a_4) \vee (x_1 = a_8 \,\&\, x_2 = a_1 \,\&\, x_3 = a_7).$$

The CDB for example 2.

$$R_1(a_1, a_3) \leftarrow$$
$$R_1(a_3, a_4) \leftarrow$$
$$R_1(a_4, a_6) \leftarrow$$
$$R_2(\mathrm{x}, \mathrm{y}) \leftarrow R_1(\mathrm{x}, \mathrm{y})$$
$$R_2(\mathrm{x}, \mathrm{y}) \leftarrow R_1(\mathrm{x}, \mathrm{z}), R_2(\mathrm{z}, \mathrm{y})$$
$$R_3(a_1, a_2, a_3) \leftarrow$$
$$R_3(\mathrm{x}, \mathrm{y}, \mathrm{z}) \leftarrow R_1(\mathrm{x}, \mathrm{y}), R_2(\mathrm{y}, \mathrm{z})$$
$$R_1(x_1, x_2) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_3) \vee (x_1 = a_3 \,\&\, x_2 = a_4) \vee (x_1 = a_4 \,\&\, x_2 = a_6)$$
$$R_2(x_1, x_2) \rightarrow R_1(x_1, x_2) \vee \exists y(R_1(x_1, \mathrm{y}) \,\&\, R_2(\mathrm{y}, x_2))$$
$$R_3(x_1, x_2, x_3) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_2 \,\&\, x_3 = a_3) \vee (R_1(x_1, x_2) \,\&\, R(x_2, x_3)).$$

(For this example, some of the existential quantifiers and equalities were eliminated by simplification.)

4.3 Negation as finite failure

The CWA and CDB approaches are computationally complex. Consequently, a more manageable approach, called *Negation as Finite Failure* (*NFF*), was introduced in (Clark, 1978). This metarule states that for any positive ground atom A, $\neg$A is proved if A fails finitely. That is, every attempt to prove A fails in a finite number of steps. The addition of NFF to SLD resolution is called SLDNF resolution. With this method, the solution for $\leftarrow \neg$A is done by attempting to solve for $\leftarrow$ A. If the latter succeeds then the former is said to fail and if the latter fails the former is said to succeed. The NFF does not apply in all cases because of the requirement that A be a ground atomic formula. Consider what the NFF would do for a nonground literal such as 1.4: $\leftarrow \neg R_1(\mathrm{x})$. Since $\leftarrow R_1(\mathrm{x})$ succeeds with $\mathrm{x} = a_1$, 1.4 would fail and no answers would be obtained, even though there are several answers. So, NFF is not applied in such a case; in general, when a negative atom with a free variable must be solved, the query is said to *flounder*.

In order to deal with floundering queries, Chan defined the notion of constructive negation. Using this technique it is possible to solve a negative atom that contains a free variable. The idea of this method in such a situation is to find all answers for the corresponding positive atom and take the negation of all the answers as the answer for the negated atom. Inequalities may be used for this

purpose. For instance, in 1.4 $\leftarrow R_1(x)$ is solved. The solution is $x = a_1 \vee x = a_2$. Then the answer for 1.4 is taken as $x \neq a_1 \,\&\, x \neq a_2$. Constructive negation can be applied whenever a negative atom in a query contains one or more free variables. Constructive negation will not be discussed further in this paper. The reader is referred to Chan (1988) for more details, including a formal definition of SLD-CNF resolution, which generalizes SLDNF resolution by the use of constructive negation.

Now, the negative queries from the beginning of this section will be solved using SLDNF resolution. The limitation of this method is shown immediately in 1.4 where the negative atom is not ground. Figures 5–7 show the attempted SLDNF resolutions for 1.5. Note that in all these cases, $\neg R_1(x_1)$ is ground by the time it is solved. If one started to solve $\neg R_1(x_1)$ before solving $R_3(x_1, x_2, x_3)$, possibly in case the order of these two literals were reversed, NFF could not be applied because of the variable in the negated atom. This shows the importance of the order of evaluation of the literals: the system may have to use a control strategy to reorder the literals in order to obtain the solutions. For 2.4 again, because of the variable in the negated query, NFF does not apply. For 2.5, Figure 8 shows the solution.

Clark proved a soundness result for NFF: If $\neg$A is proved using NFF from P, then CDB $\models \neg$A. However, as shown above, NFF is not complete because of the problem of variables in negated queries. The NFF is complete for ground negated atoms, that is, if CDB $\models \neg$A, then $\neg$A is proved by NFF from P. As far as the strengths of the three methods for handling negation are concerned, the result is as follows:

Theorem 8 (See Shepherdson (1988) for a thorough study of negation in logic programming.) NFF $\Rightarrow$ CDB $\Rightarrow$ CWA. (A negative atom obtained from P by NFF is also obtained using CDB. A negative atom obtained from P by using CDB is also obtained using CWA.)

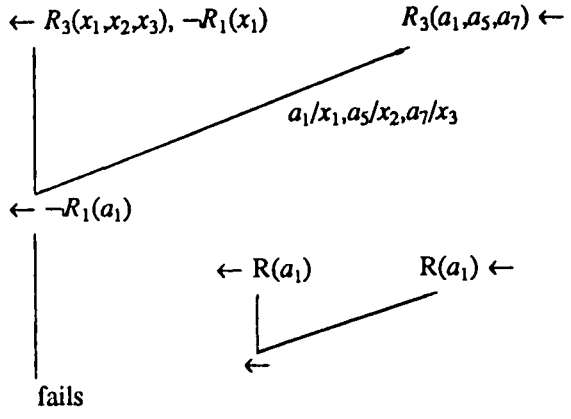


Figure 5

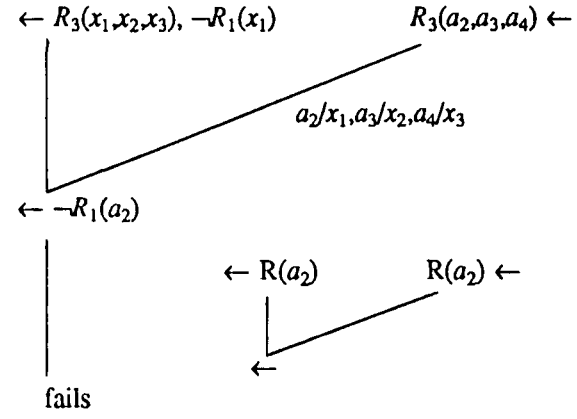
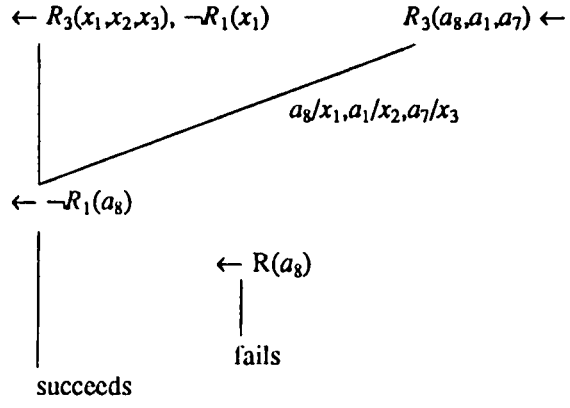
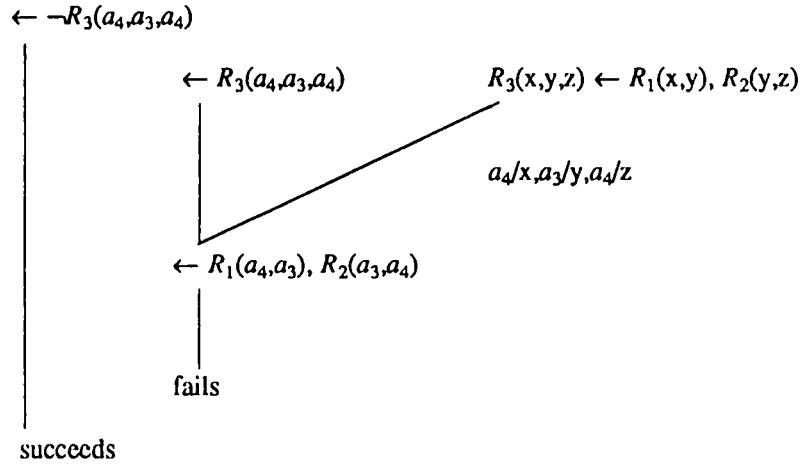


Figure 6



Answer : $\langle x_1,x_2,x_3\rangle = \langle a_8,a_1,a_7\rangle$

Figure 7



Answer : Yes

Figure 8

The following example shows that CWA $\not\Rightarrow$ CDB. Let C = {a,b,R} and P = {R(a) ← R(b), R(b) ← R(a)}. Now $\neg$R(a) is obtained from P using the CWA but not if the CDB is used because the CDB definition for R is

$$R(x) \rightarrow (x = a \;\&\; R(b)) \vee (x = b \;\&\; R(a)).$$

4.4 Implementation of negation

This section ends with some comments about implementations of Horn databases including negation. During the past decade much research has been done on this topic. There are two main methods. First, the logic programming language Prolog is a system that can implement such databases immediately. However, Prolog is not practical for large databases because of its tuple at a time retrieval and its lack of capabilities for handling large disk files. The second method may be divided into two cases. If the axioms are nonrecursive, they can be expressed in terms of the relations of a relational database: this is called the compiled approach (Reiter, 1978). Hence a one step deduction can interface with a relational database system. In the case of recursive axioms, instead of the top-down theorem proving approach (starting with the goal), many bottom-up techniques

(starting with the data and using rules to derive new statements) have been developed for efficient query evaluation. See Ullman (1988) for details.

5 Stratified databases

In the Introduction a clause was written using two formulations:

$$\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m \quad \text{and}$$
$$B_1, \ldots, B_m \leftarrow A_1, \ldots, A_n,$$

with the atoms $A_1, \ldots, A_n$ in the body and $B_1, \ldots, B_m$ in the head. But in some cases it is helpful to use negation in the body of a clause. For instance, a supplier might wish to backorder items that are not in the warehouse. It would be convenient to write

$$\text{Backorder(x)} \leftarrow \text{Item(x)}, \neg \text{Warehouse(x)},$$

but if only positive atoms are allowed in the body, the clause must be written as

$$\text{Backorder(x)}, \text{Warehouse(x)} \leftarrow \text{Item(x)}.$$

This is a disjunctive clause which, while logically equivalent to the former, does not differentiate between the Backorder and Warehouse relations. However, the intent is to backorder an item if it is not in the warehouse, not the reverse.

Stratified databases allow negated atoms in the body of the clause as long as a certain intuitively reasonable restriction is met. This section contains the definition of stratified databases and an example. Semantics are presented for stratified databases including declarative, fixpoint, and procedural semantics. These semantics are natural extensions of the semantics discussed in section 3; however, negation can no longer be treated separately because of the presence of negated atoms in clauses.

5.1 Definition of stratified databases

A *normal* database consists of clauses of the form

$$\text{B} \leftarrow L_1, \ldots, L_n,$$

where each L_i, $1 \leqslant \text{i} \leqslant \text{n}$, is a literal and B is an atom. A Horn database is a special case of a normal database, where each L_i, $1 \leqslant \text{i} \leqslant \text{n}$, is an atom. The terminology "normal database" is not standard, but it follows the terminology "normal program" of (Lloyd, 1987). But when negated atoms are allowed in the body, one must be careful that the meaning of the database is clearly defined. Consider the case of the following two clauses in the language with the constant symbol a and the unary predicate symbols Male and Female:

$$\text{Female(a)} \leftarrow \neg \text{Male(a)}$$
$$\text{Male(a)} \leftarrow \neg \text{Female(a)}.$$

The meaning of this combination of recursion via negation is not clear. Which is true, Male(a) or Female(a)? This database does not have a clear semantics. The notion of stratification was introduced in Apt, Blair, and Walker (1988), Chandra and Harel (1987), Naqvi (1986), and Van Gelder (1988) to avoid this problem. Stratification is defined by the use of a *level* (rank) mapping, r, which is a mapping from the set of predicate symbols of the database to the positive integers, thereby providing a level for each predicate symbol, written as r(R) for the predicate symbol R. For a literal L, r(L) stands for r(R), where R is the predicate symbol of L. A normal database, P, is *stratified* if there is a level mapping r, so that for every clause in P (of the form):

$$\text{B} \leftarrow L_1, \ldots, L_n$$

if L_i is positive, that is, an atom, then $\text{r}(L_i) \leqslant \text{r(B)}$, and if L_i is negative, then $\text{r}(L_i) < \text{r(B)}$, for all i,

$1 \leqslant i \leqslant n$. So, in a stratified database recursion via negation is not allowed. The set of clauses of P are placed into *strata* P_i, where each P_i contains the definitions of predicates of level i. Note that the example given above with C = {a,Male,Female} is not stratified, because any level mapping must have both r(Male) < r(Female) and r(Female) < r(Male), which is impossible. Note also that a stratified database may have many different level mappings, but the results about stratified databases hold for any choice of r. For the rest of this section it will be assumed that the levels are 1, 2, ..., n.

5.2 Examples of stratified databases

The example including the Backorder predicate given earlier in this section, with the assumption of facts for the Item and Warehouse predicates is a stratified database. As another example, consider the representation of a directed graph with the predicates edge and path, where the edge predicate is given by ground atoms such as

Edge(a,b) ←,

and path is defined recursively (without negation) as

Path(x,y) ← Edge(x,y)
Path(x,y) ← Edge(x,z),Path(z,y).

Sometimes it is of interest to consider acyclic paths, where two nodes are connected but do not belong to the same cycle. The definition is

Acyclicpath(x,y) ← Path(x,y),¬Path(y,x),

and again the existing recursion (for Acyclicpath) does not involve negation (on the predicate Acyclicpath). This database is stratified with Edge and Path at a lower level than Acyclicpath because Acyclicpath is defined negatively in terms of Path. Note that although Path is defined in terms of Edge, Edge and Path may be at the same level since the definition is positive.

The following abstract example is used to illustrate the issues involving stratified theories in this section.

Example 3 A stratified database theory.
C contains the constants a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, the unary predicate symbol R_1, the binary predicate symbols R_2, R_3, and the ternary predicate symbol R_4.

P contains

$R_2(a_1, a_3) \leftarrow$
$R_2(a_3, a_4) \leftarrow$
$R_2(a_4, a_6) \leftarrow$
$R_3(x, y) \leftarrow R_2(x, y)$
$R_3(x, y) \leftarrow R_2(x, z), R_3(z, y)$
$R_4(x, y, z) \leftarrow R_2(x, y), \neg R_3(y, z), R_2(y, a_4)$
$R_1(x) \leftarrow R_4(y, z, x), \neg R_4(x, z, x).$

To make this example more meaningful, think of R_2 as Edge, R_3 as Path, R_4 as a special ternary relationship requiring edges from x to y and y to a_4 and no path from y to z. R_1 is then defined in terms of R_4. This database has a level mapping with $r(R_2) = r(R_3) = 1$, $r(R_4) = 2$, $r(R_1) = 3$; hence it is stratified. The strata are: P_1 = the first 5 clauses in P; P_2 = the sixth clause (definition of R_3) in P; P_3 = the last clause in P. Note that R_4 is placed at the second level because it is defined in terms of $\neg R_3$. In order to compute R_4, one must know all the answers to R_3. Answers to R_3 can be computed using the first stratum, then answers to R_4 can be computed using the second stratum. A similar reasoning places R_1 at the third level.

For stratified databases, negation is not treated separately from the (positive) semantics. The

reason for this is that negation is already built into the definition, as negated atoms are allowed in the definition of a predicate. Hence the declarative, fixpoint, and procedural semantics must involve negation. At the end of the section, there will be a few remarks concerning aspects of the CWA and the CDB.

The following four queries are posed for this database:

3.1 $\leftarrow R_3(x, y), \neg R_2(x, y)$
3.2 $\leftarrow R_4(a_1, a_4, x)$
3.3 $\leftarrow R_3(x, y), R_4(x, y, y)$
3.4 $\leftarrow R_1(a_1)$.

5.3 Declarative semantics for stratified databases

Declarative semantics for stratified databases is defined in terms of Herbrand models as a level-by-level process. An important concept here is the following notion: A model M is *supported* (Apt, Blair, and Walker, 1988) if every atom $A \in M$ is the conclusion of a ground instance of a clause in P whose body is true in M (or empty). For instance, if the language is the same as for the earlier example with C = {a,Male,Female}, but the database contains only the clause

Female(a) $\leftarrow \neg$Male(a),

then the minimal model {Female(a)} is supported, but the minimal model {Male(a)} is not. Intuitively, the supported minimal model appears better than the nonsupported minimal model because there is no way to prove Male(a), so we may assume Male(a) to be false, and that makes Female(a) true.

The definition of the minimal supported model is given by levels as follows: (assuming the strata $P_1, \ldots, P_n$) (Apt, Blair, and Walker, 1988).

> $M_i = \cap$ {M: M is a supported model of $\cup \{P_j \mid 1 \leqslant j \leqslant i\}$ and coincides with M_{i-1} on predicates of level $\leqslant i - 1$}.
> When i = 1, the second part of the definition involving M_{i-1} is omitted.

M_n is taken as the minimal supported model for the stratified database. Note the recursive nature of the definition via levels. The model is constructed on a level-by-level basis for the predicates, that is, first the predicates of level 1 are defined, then the predicates of level 2, and so on.

In the case of Example 3, the minimal supported model is obtained as follows:

> $M_1 = \{R_2(a_1, a_3), R_2(a_3, a_4), R_2(a_4, a_6), R_3(a_1, a_3), R_3(a_3, a_4), R_3(a_4, a_6), R_3(a_1, a_4), R_3(a_1, a_6), R_3(a_3, a_6)\}$ (this is the minimal model of the first 5 clauses in P),
> $M_2 = M_1 \cup \{R_4(a_1, a_3, a_1), R_4(a_1, a_3, a_2), R_4(a_1, a_3, a_3), R_4(a_1, a_3, a_5), R_4(a_1, a_3, a_7), R_4(a_1, a_3, a_8)\}$ (the atoms generated by the rule for R_4 are added),
> $M_3 = M_2 \cup \{R_1(a_2), R_1(a_3), R_1(a_5), R_1(a_7), R_1(a_8)\}$ (the atoms generated by the rule for R_1 are added).

Since in this case n = 3, M_3 is the model that provides the declarative semantics for this stratified theory. Based on M_3, the answers to the queries are as follows:

> for 3.1: $\langle a_1, a_4\rangle, \langle a_1, a_6\rangle, \langle a_3, a_6\rangle$
> for 3.2: no answers: the set of answers is $\varnothing$
> for 3.3: $\langle a_1, a_3\rangle$
> for 3.4: No: $R_1(a_1)$ is not true.

5.4 Fixpoint semantics for stratified databases

Fixpoint semantics can also be defined for stratified databases. Recall first the definition of T $\uparrow$ from section 3. The T_P operator is monotonic for Horn databases, but need not be monotonic for

stratified databases. For example, if in a stratified database $R(a) \in T_P \uparrow i$ by the rule $R(a) \leftarrow \neg P(a)$ and $P(a) \in T_P \uparrow i$, then not $R(a) \in T_P \uparrow (i + 1)$. Hence, in this section $\uparrow$ is used to denote cumulative powers of T as follows:

$$T \uparrow 0(I) = I$$
$$T \uparrow (i + 1)(I) = T(T \uparrow i(I)) \cup T \uparrow i(I)$$
$$T \uparrow \omega(I) = \cup \{T \uparrow i(I) \mid i < \omega\}.$$

The definition of fixpoint semantics here also proceeds through the strata $P_1, \ldots, P_n$ of the program P:

$$M_i = T_{P_i} \uparrow \omega(M_{i-1}),$$

where $M_0 = \varnothing$. The result is the same as for the declarative semantics.

Theorem 9 (Apt, Blair and Walker, 1988). For every stratified program P, the declarative semantics M_n and the fixpoint semantics, M_n, are identical.

Although not explored in this paper, it should be noted that there is another way to define M_n, as the perfect model of a stratified database, by using a semantics based on a preference relation (Przymusinski, 1988a).

5.5 Procedural semantics for stratified databases

A procedural semantics for stratified databases called SLS-resolution was defined in (Przymusinski, 1988b). SLS-resolution is a generalization of SLDNF-resolution, where an infinite branch of a solution tree is considered to fail. Determining when there is an infinite branch in a proof tree is a computationally difficult problem. The reader is referred to Przymusinski (1988b) for details. The equivalence of the declarative, fixpoint, and procedural semantics is shown by the following theorem.

Theorem 10 (Przymusinski, 1988b). For a stratified database P, M_n (declarative) = M_n (fixpoint) = Succ(P). Also, the declarative, fixpoint, and procedural semantics provide identical answers to non-floundering queries (that is, queries that do not require solving a negative atom that contains a variable).

Parts of the solutions for the queries 3.1–3.4 are given in Figures 9–12 using SLS-resolution, which in these cases reduces to SLDNF-resolution. Figure 9 provides a solution for 3.1; Figure 10 demonstrates the lack of solutions for 3.2; Figure 11 finds the single solution for 3.3; and Figure 12 provides an attempt to solve 3.4.

Although negation is not treated separately for stratified databases, one may wonder about the connections with the CWA and the CDB. The proper extension of the CWA to stratified databases is the ICWA (Iterated Closed World Assumption), defined by iteration on the level of the predicates (Gelfond, Przymusinska and Przymusinski, 1989) as follows:

M_i = Extend M_{i-1} by applying the CWA to P_i and leaving all predicates of level $<i$ as in M_{i-1} $(M_0 = \varnothing)$

Theorem 11 (Gelfond, Przymusinska and Przymusinski, 1989). The M_n obtained by the ICWA is identical to the M_n obtained by the declarative and fixpoint semantics.

Finally, as far as the CDB is concerned, the following result shows that the semantics of M_n is stronger than the CDB.

Theorem 12 (Przymusinski, 1988b). CDB $\models F \Rightarrow M_n \models F$ for every sentence F.

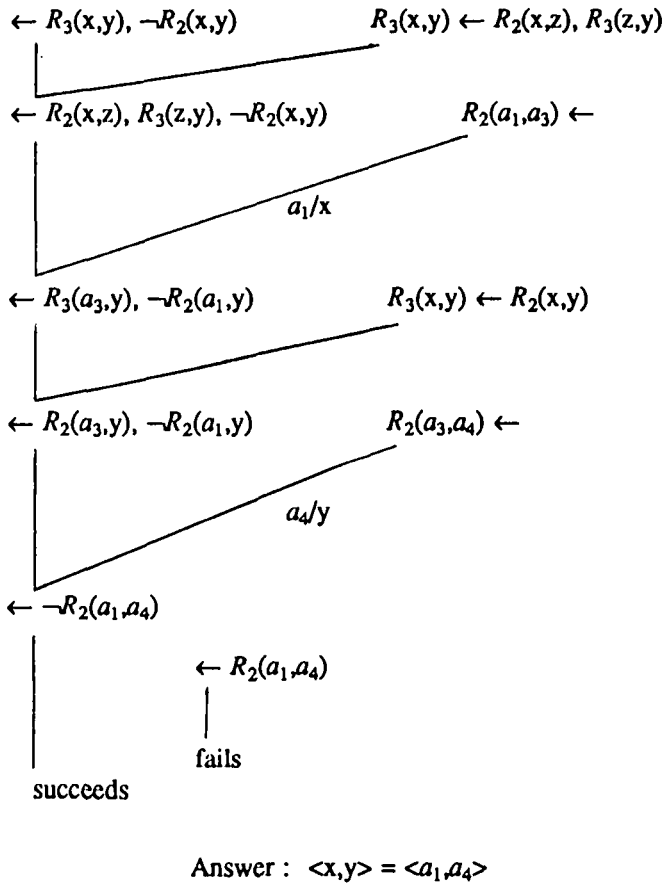


Figure 9

The database consisting of only R(a) ← R(a) (C = {a, R}) shows that the arrow goes in one direction only. In this case n = 1, $M_1 = \varnothing$, so $M_n \vdash \neg$R(a). However, the CDB =

R(a) ← R(a)
R(x) → (x = a) & R(a),

so

CDB $\nvdash \neg$R(a).

Stratified databases have been implemented as parts of the NAIL! system developed at Stanford University and the LDL system at MCC in Austin, Texas. Chapter 16 of Ullman (1988) contains some details and additional references. The LDL language and additional results about stratified databases are explained in detail in Naqvi and Tsur (1989).

6 Conclusion and summary

This article (including the Appendix) showed that there are many different types of database theories. Different theories may allow a different set of formulas and may have different semantics. Three major semantics were discussed. Declarative semantics constructs a minimal model or a set of minimal models. The true statements are the ones that are true in the minimal model or models. Fixpoint semantics also constructs a model, or a set of models, by taking the least fixpoint of an operator on models. Procedural semantics provides a resolution based proof procedure. The typical major result is the equivalence of these three semantics.

Negation represents a problem for deductive databases because of the difficulty of storing negative facts and conclusions. Three major methods for handling negation were considered. The

Closed World Assumption adds all unprovable facts to the database as negative facts. The Completed Database changes "if" definitions of predicates to "if and only if" definitions. Negation as Finite Failure expands the resolution based proof procedure for positive formulas by taking a negative atom to be solved if the corresponding positive atom fails finitely along every proof path, and not solved if the corresponding positive atom has a proof. In the case of normal databases negation is inherent already in the semantics and is not treated separately.

The following major types of theories were studied: relational, Horn, protected, disjunctive, typed, extended Horn, and normal databases. The body of the paper concentrated on the Horn (of which

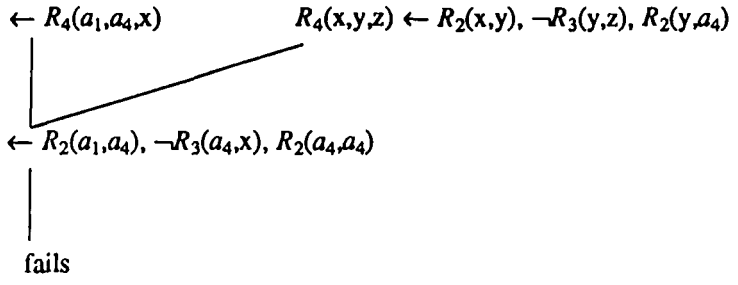


Figure 10

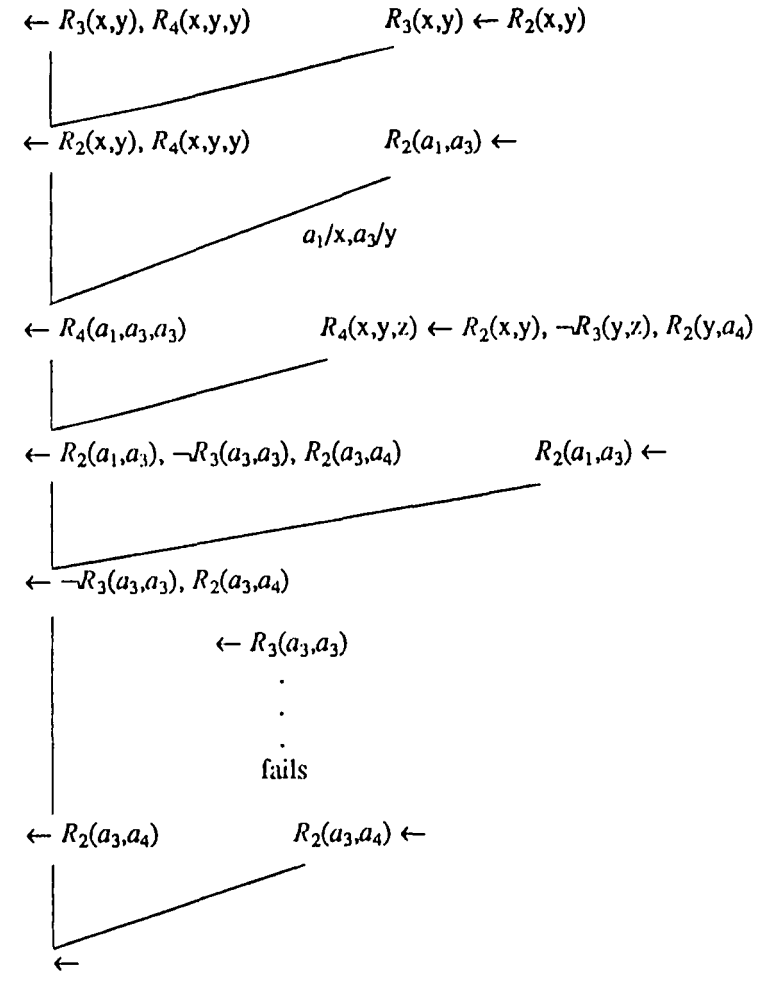


Figure 11

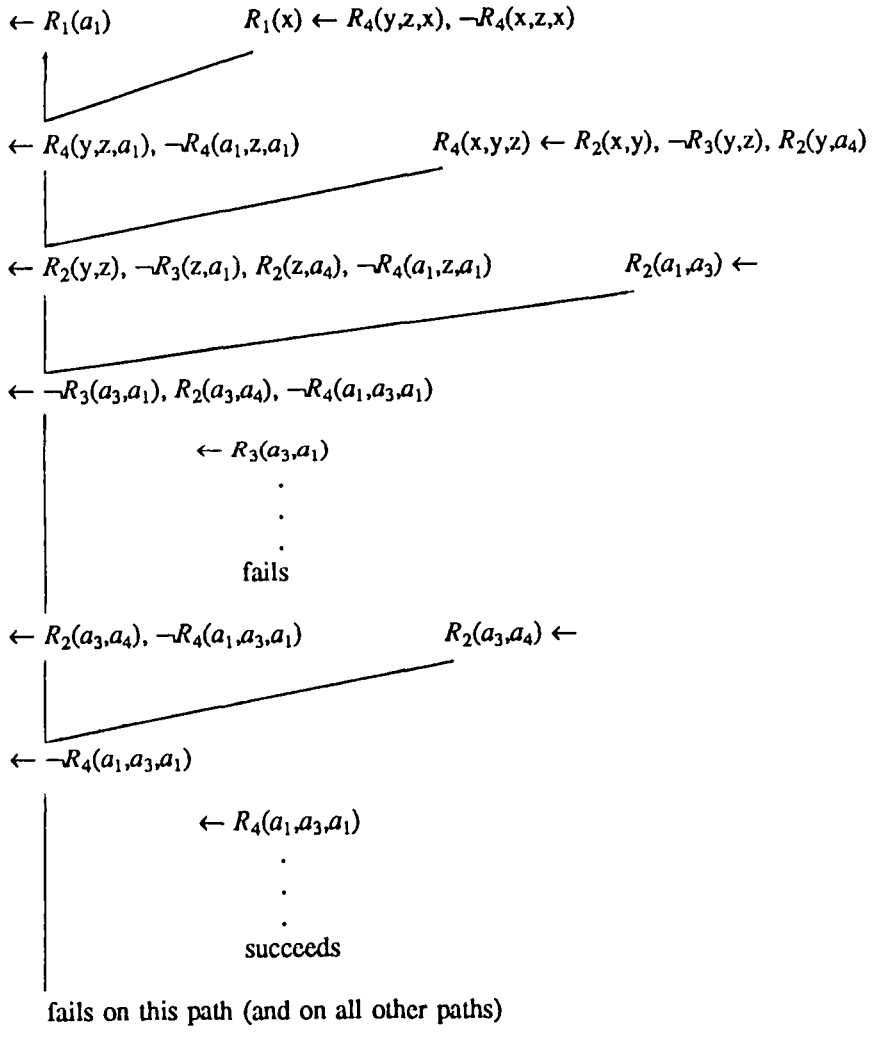


Figure 12

the relational is an important subcase) and stratified cases. In the Appendix protected and disjunctive databases were treated in more detail than the last three. It is important to understand that more complex types of theories may be able to represent more complex situations at the price of requiring more complicated proof procedures. The user should be able to decide on the complexity required and the limitations of a deductive database system should be made clear.

Appendix

A1 Protected databases

For Horn databases the CWA is one approach for handling negation, as discussed in section 4. A problem with the CWA is that all atoms must be either true or false; there is no way of handling exceptional (unknown) data. The present section deals with the process of representing a situation where neither the truth nor the falsity of some atoms should be assumed. Such a database is said to be *protected.* Protected databases, introduced by Minker and Perlis, are discussed in several papers; Minker and Perlis (1985) is used as the main reference. This section provides an example and contrasts the three main methods of dealing with negation for protected databases.

A1.1 Notation and example for protected databases

The protection of an atom R_i(a) is represented by writing ER_i(a). This is understood to mean that

R_i(a) should not be assumed to be false. For the Horn database of family relationships sketched in section 2, the following protected atoms might be appropriate:

Eparent(steve,susan) ←,
Eparent(ann,bob) ←.

In the protected framework, this means that the atoms (not asserted in the database) parent(steve,susan) and parent(ann,bob) should not be assumed false.

The notion of protection is further illustrated by the following example, which is obtained from Example 2 by protecting the atoms $R_1(a_1, a_4)$ and $R_1(a_2, a_3)$.

Example 4 A protected database theory.

C contains the constants $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$, the binary predicate symbols R_1, R_2, and the ternary predicate symbol R_3.

P contains

$R_1(a_1, a_3) \leftarrow$
$R_1(a_3, a_4) \leftarrow$
$R_1(a_4, a_6) \leftarrow$
$R_2(x, y) \leftarrow R_1(x, y)$
$R_2(x, y) \leftarrow R_1(x, z), R_2(z, y)$
$R_3(a_1, a_2, a_3) \leftarrow$
$R_3(x, y, z) \leftarrow R_1(x, y), R_2(y, z)$
$ER_1(a_1, a_4) \leftarrow$
$ER_1(a_2, a_3) \leftarrow$
$ER_1(x, y) \leftarrow R_1(x, y)$
$ER_2(x, y) \leftarrow R_2(x, y)$
$ER_3(x, y, z) \leftarrow R_3(x, y, z)$
$ER_2(x, y) \leftarrow ER_1(x, y)$
$ER_2(x, y) \leftarrow ER_1(x, z), ER_2(x, y)$
$ER_3(x, y, z) \leftarrow ER_1(x, y), ER_2(y, z).$

Note that in addition to the two protection axioms for $R_1(a_1, a_4)$ and $R_1(a_2, a_3)$, six more axioms have been added. The first three assert that any tuple in R_i is automatically in ER_i, that is, should not be assumed false. The last three are the ER_i versions of the axioms involving R_i.

The ER_i predicates are extra predicates used to express protection from negative assumptions. Consider now the three semantics discussed in section 3: declarative, fixpoint, and procedural. When no negation is involved in a query, and also no ER_i, then the axioms involving ER_i are irrelevant. Hence for this case, a protected database reduces to a Horn database. In particular, the answers are the same to 2.1–2.3 in Examples 2 and 4.

A1.2 Negation for protected databases

The ER_i predicates come into play in connection with negation. The CWA is extended to protected databases by applying it to ER_i instead of R_i. This is reasonable because ER_i contains R_i as well as tuples that are protected. Hence any tuple not in ER_i may be assumed as false. The protected CWA is contrasted with the CWA and illustrated on the following two queries.

4.1 $\leftarrow \neg R_1(a_1, x)$.

The CWA answers, using Example 2, are $a_1, a_2, a_4, a_5, a_6, a_7, a_8$. But the protected CWA answers are $a_1, a_2, a_5, a_6, a_7, a_8$ because $\neg R_1(a_1, a_4)$ may no longer be assumed.

4.2 $\leftarrow \neg R_2(a_2, x)$.

The CWA answers, using Example 2, are all the constants: a_1–a_8. The protected CWA answers are a_1, a_2, a_5, a_7, a_8, because P implies $ER_2(a_2, a_3)$, $ER_2(a_2, a_4)$, and $ER_2(a_2, a_6)$.

The Completed Database (CDB) is extended to protected databases in the following way: All the ER_i axioms are deleted; however, for the completion axioms the ground ER_i facts are treated as if they were R_i facts. In the case of Example 4 the CDB yields

$$R_1(a_1, a_3) \leftarrow$$
$$R_1(a_3, a_4) \leftarrow$$
$$R_1(a_4, a_6) \leftarrow$$
$$R_2(x, y) \leftarrow R_1(x, y)$$
$$R_2(x, y) \leftarrow R_1(x, z), R_2(z, y)$$
$$R_3(a_1, a_2, a_3) \leftarrow$$
$$R_3(x, y, z) \leftarrow R_1(x, y), R_2(y, z)$$
$$R_1(x_1, x_2) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_3) \vee (x_1 = a_3 \,\&\, x_2 = a_4) \vee (x_1 = a_4 \,\&\, x_2 = a_2) \vee (x_1 = a_1 \,\&\, x_2 = a_4) \vee (x_1 = a_2 \,\&\, x_2 = a_3)$$
$$R_2(x_1, x_2) \rightarrow R_1(x_1, x_2) \vee \exists y(R_1(x_1, y) \,\&\, R_2(y, x_2))$$
$$R_3(x_1, x_2, x_3) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_2 \,\&\, x_3 = a_3) \vee (R_1(x_1, x_2) \,\&\, R_2(x_2, x_3)).$$

This CDB may be contrasted with the CDB presented in Section 4.2 for the same database without the two protected atoms. The only difference is in the eighth clause where the protected values are included as possible values in R_1.

In typical cases for Horn databases, the protected CWA and the protected CDB are equivalent. However, the example given in section 4.3, showing that the CWA and the CDB need not always be equivalent, also applies in the protected case.

Finally, NFF can also be extended to the protected case by slightly modifying SLDNF resolution: For any positive ground atom A, $\neg$A is proved if it is neither possible to prove A nor EA. The queries 4.1 and 4.2 are not candidates for SLDNF resolution because the atoms are not ground. However, consider the queries.

4.3 $\leftarrow \neg R_1(a_1, a_4)$,
4.4 $\leftarrow \neg R_2(a_2, a_3)$.

The answer is No in both cases, as shown in Figures 13 and 14. The result analogous to Theorem 8 is the following.

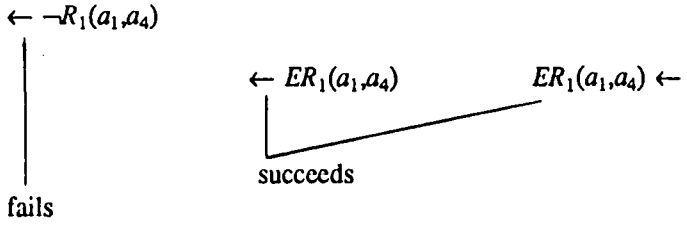


Figure 13

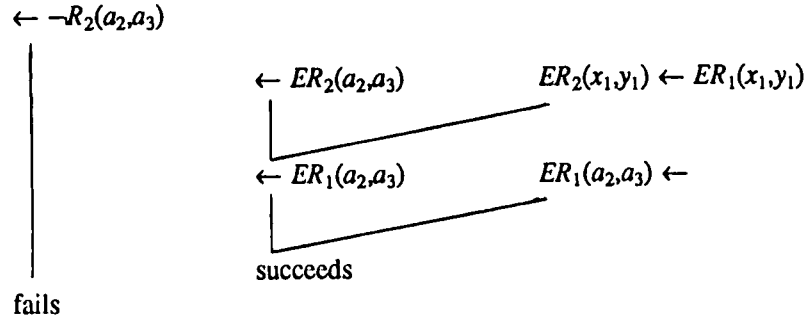


Figure 14

Theorem 13 (Minker and Perlis, 1985). Protected NFF ⇒ Protected CDB ⇒ Protected CWA.

A1.3 Implementation of protected databases

Minker and Perlis (1985) contains several techniques for the implementation of protected databases. A protected relational algebra algorithm is given for answering queries in Horn databases. Also, it is shown there that a simple Prolog implementation can define $\neg_E$ to be a new negation relative to the protected CWA by using the representation

(1) $\neg_E$ P ← P & / & fail.
(2) $\neg_E$ P ← EP & / & fail.
(3) $\neg_E$ P ←.

The symbol "/" is the *cut* operator which automatically succeeds when encountered as a goal, and has the side effect that the system becomes committed to all choices made since the parent goal was invoked; thus the cut limits backtracking.

A2 Disjunctive databases

The previous sections dealt with the important issues of semantics and negation for database theories where the head of each clause contains a single atom. However, true disjunctive information is not expressible in such a way. This section studies the semantics, including negation, for disjunctive databases.

A2.1 Notation and example for disjunctive databases

Section 2.2 indicated that standard examples of relational databases contain relations such as Supplier, Part, Employee, etc. Typical disjunctive data might be

Supplier(1536,ABC Co.,Washington), Supplier(1536,ABC Co.,New York) ←

indicating that the supplier with supplier number 1536, whose name is ABC Co. is either in Washington or in New York (or in both places). For the Horn database representing family relationships sketched in section 2, an example of a disjunctive rule is

father(x,y), mother(x,y) ← parent(x,y)

indicating that a parent is either a father or a mother.

For the rest of this section, the following abstract example is used to illustrate the issues involving disjunctive databases.

Example 5 A disjunctive database.
C contains the constants $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$, the unary predicate symbols R_1, R_2, and the binary predicate symbols R_3, R_4.

P contains

$R_3(a_1, a_3), R_3(a_3, a_4) \leftarrow$
$R_3(a_4, a_6) \leftarrow$
$R_4(x, y) \leftarrow R_3(x, y)$
$R_4(x, y) \leftarrow R_3(x, z), R_4(z, y)$
$R_1(x), R_2(y) \leftarrow R_4(x, y)$
$R_1(y) \leftarrow R_3(x, y).$

The following three queries are posed for this database:

5.1 $\leftarrow R_4(x, y)$
5.2 $\leftarrow R_1(a_6)$
5.3 $\leftarrow R_3(x, y), R_4(x, a_6).$

The notion of an answer to a query needs to be redefined, because the previous definition in section 2 dealt only with definite answers. For the query $Q(x_1, \ldots, x_m)$ an (*indefinite*) *answer* has the form

$$\langle a_{11}, \ldots, a_{1m}\rangle + \langle a_{21}, \ldots, a_{2m}\rangle + \cdots + \langle a_{k1}, \ldots, a_{km}\rangle \text{ such that}$$
$$P \models Q(a_{11}, \ldots, a_{1m}) \vee Q(a_{21}, \ldots, a_{2m}) \vee \ldots \vee Q(a_{k1}, \ldots, a_{km}).$$

Thus an answer is a sum of tuples such that in every model of P at least one of $Q(a_{i1}, \ldots, a_{im})$ is true. A *minimal indefinite answer* is one none of whose subsums is also an answer. Usually, one is interested only in minimal indefinite answers, so that only those will be listed. For the above queries, the answers are as follows:

for 5.1, $\langle a_4, a_6\rangle$, $\langle a_1, a_3\rangle + \langle a_3, a_4\rangle$, $\langle a_1, a_3\rangle + \langle a_3, a_6\rangle$,
for 5.2, yes,
for 5.3, $\langle a_4, a_6\rangle$.

A2.2 Declarative semantics for disjunctive databases

As in the case of Horn databases, declarative semantics involves Herbrand models. However, a disjunctive database, such as the one above, does not possess a unique smallest Herbrand model. Such a database has *minimal* Herbrand models, M_i, that is, subsets of HB_C which are models of P, but such that no subset of M_i is a model of P. This approach was introduced in (Minker, 1982).

In the case of Example 5 there are six minimal models:

$$M_P^1 = \{R_1(a_1), R_1(a_3), R_1(a_4), R_1(a_6), R_3(a_1, a_3), R_3(a_4, a_6), R_4(a_1, a_3), R_4(a_4, a_6)\}$$
$$M_P^2 = \{R_1(a_1), R_1(a_3), R_1(a_6), R_2(a_6), R_3(a_1, a_3), R_3(a_4, a_6), R_4(a_1, a_3), R_4(a_4, a_6)\}$$
$$M_P^3 = \{R_1(a_3), R_1(a_4), R_1(a_6), R_2(a_3), R_3(a_1, a_3), R_3(a_4, a_6), R_4(a_1, a_3), R_4(a_4, a_6)\}$$
$$M_P^4 = \{R_1(a_3), R_1(a_6), R_2(a_3), R_2(a_6), R_3(a_1, a_3), R_3(a_4, a_6), R_4(a_1, a_3), R_4(a_4, a_6)\}$$
$$M_P^5 = \{R_1(a_3), R_1(a_4), R_1(a_6), R_3(a_3, a_4), R_3(a_4, a_6), R_4(a_3, a_4), R_4(a_3, a_6), R_4(a_4, a_6)\}$$
$$M_P^6 = \{R_1(a_4), R_1(a_6), R_2(a_4), R_2(a_6), R_3(a_3, a_4), R_3(a_4, a_6), R_4(a_3, a_4), R_4(a_3, a_6), (R_4(a_4, a_6)\}.$$

The result for disjunctive databases that is analogous to Theorem 2 is

Theorem 14 (Minker, 1982). For a disjunctive database P, for every positive clause E, $P \models E$ if and only if E is true in every minimal model of P.

This theorem states that the positive clauses logically implied by P are exactly the ones that are true in all minimal models.

A2.3 Fixpoint semantics for disjunctive databases

A fixpoint semantics for disjunctive databases was recently defined in Minker and Rajasekar (1989). Before the definition of the mapping, it is necessary to define the Extended Herbrand Base, EHB_C, of a language C, as the set of all finite disjunctions of different atoms of the Herbrand Base, HB_C. The closure operator is defined not on Herbrand interpretations, but on states S, where $S \subseteq EHB_C$. A state may be thought of as a set of true disjuncts.

The operator T_P^I is defined as follows:

$T_P^I(S) = \{E \in EHB_C \mid E' \leftarrow A_1, A_2, \ldots, A_n$ is a ground instance of a clause in P, $A_1 \vee E_1, \ldots, A_n \vee E_n$ are in S (any of $E_1, \ldots, E_n$ may be null), $E'' = E' \vee E_1 \vee \ldots \vee E_n$, and E is the smallest factor of E"; the $A_1, \ldots, A_n$ are atoms; $E_1, \ldots, E_n$, E, E' E" are positive auses$\}$.

In this paper $T_P^I(S)$ is simplified by omitting subsumed clauses from it. (A subsumed clause is a disjunction all of whose atoms are in another clause in the set.)

The results of the T_P^I operator for Example 5 are as follows:

$$T_P^I \uparrow 1 = \{R_3(a_1, a_3) \vee R_3(a_3, a_4), R_3(a_4, a_6)\}$$
$$T_P^I \uparrow 2 = T_P^I \uparrow 1 \cup \{R_1(a_6), R_1(a_3) \vee R_3(a_3, a_4), R_1(a_4) \vee R_3(a_1, a_3), R_3(a_1, a_3) \vee R_4(a_3, a_4), R_3(a_3, a_4) \vee R_4(a_1, a_3), R_4(a_4, a_6)\}$$
$$T_P^I \uparrow 3 = T_P^I \uparrow 2 \cup \{R_1(a_1) \vee R_2(a_3) \vee R_3(a_3, a_4), R_1(a_3) \vee R_1(a_4), R_1(a_3) \vee R_2(a_4) \vee R_3(a_1, a_3), R_1(a_3) \vee R_4(a_3, a_4), R_1(a_3) \vee R_4(a_3, a_6), R_1(a_4) \vee R_2(a_6), R_1(a_4) \vee R_4(a_1, a_3), R_3(a_1, a_3) \vee R_4(a_3, a_6), R_4(a_1, a_3) \vee R_4(a_3, a_4), R_4(a_1, a_3) \vee R_4(a_3, a_6)\}$$
$$T_P^I \uparrow 4 = [T_P^I \uparrow 3 - \{R_1(a_3) \vee R_2(a_4) \vee R_3(a_1, a_3)\}] \cup \{R_1(a_1) \vee R_1(a_4) \vee R_2(a_3), R_1(a_1) \vee R_2(a_3) \vee R_4(a_3, a_4), R_1(a_1) \vee R_2(a_3) \vee R_4(a_3, a_6), R_1(a_3) \vee R_2(a_4), R_1(a_3) \vee R_2(a_6)\}$$
$$T_P^I \uparrow \omega = T_P^I \uparrow 4.$$

One can check that $T_P^I \uparrow \omega$ is logically equivalent to the set of all disjunctions that are true in each minimal model. The next theorem is the general result that provides the equivalence of declarative and fixpoint semantics.

Theorem 15 (Minker and Rajasekar, 1989). For a disjunctive database P and a positive clause E, $P \models E$ if and only if E is true in every minimal model of P if and only if $T_P^I \uparrow \omega \models E$.

A2.4 Procedural semantics for disjunctive databases

SLD-resolution provides a procedural interpretation for Horn databases but is not sufficient for disjunctive databases. SLI-resolution, first defined in Minker and Zanon (1982) as LUST-resolution, is the counterpart to SLD-resolution for disjunctive databases. The formal definition of SLI-resolution is somewhat complex and is not given here; the reader is referred to Minker and Zanon (1982) and Minker and Rajasekar (1989). The success set of P, *Succ*(*P*), can now be defined as the set of all $E \in EHB_C$, $E = E_1 \vee \ldots \vee E_k$, such that $P \cup \{\leftarrow E_1, \ldots, E_k\}$ has an SLI-refutation. The equivalence of declarative, fixpoint, and procedural semantics is shown by the following theorem.

Theorem 16 (Minker and Rajasekar, 1989). For a disjunctive database P and positive clause E, $P \models E$ if and only if E is true in every minimal model of P if and only if $T_P^I \uparrow \omega \models E$ if and only if $E \in$ Succ(P). Also, the declarative, fixpoint, and procedural semantics provide identical answers to positive queries.

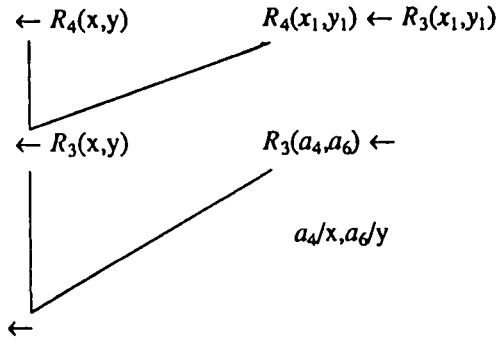
A modified version of SLI-resolution is illustrated next by deriving the answers to queries 5.1–5.3. The answers to 5.1 are shown in Figures 15–17. The SLI-resolution for 5.2 is given in Figure 18. For 5.3, Figure 19 provides the answer.

A2.5 Negation for disjunctive databases

The methods given in section 4 for dealing with negation in Horn databases do not work properly for disjunctive databases. For example, since neither $R_1(a_1, a_3)$ nor $R_1(a_3, a_4)$ are logically implied by P in Example 5, the CWA allows for the addition of $\neg R_1(a_1, a_3)$ and $\neg R_1(a_3, a_4)$, thereby making the database inconsistent. The rest of this section shows how the CWA, CDB, and NFF can be generalized to apply to disjunctive databases.

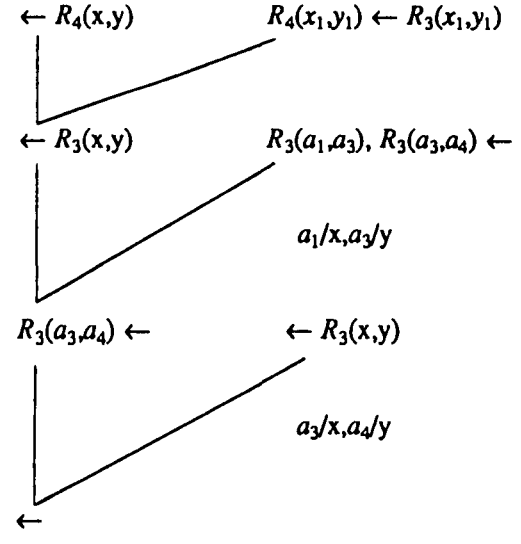
The *Generalized Closed World Assumption* (*GCWA*) was introduced in Minker (1982). This metarule is consistent with disjunctive databases. The syntactic definition is as follows:

> If A is a ground atom, then $\neg$A is inferred from P by the GCWA if not($A \in T$) where $T = \{B \mid B$ is a ground atom, $P \models B \vee K$, K is a positive (or null) clause and $P \not\models K\}$.



Answer : <x,y> = <a_4,a_6>

Figure 15



Answer : <x,y> = <a_1,a_3> + <a_3,a_4>

Figure 16

There is an equivalent semantic definition of the GCWA as follows:

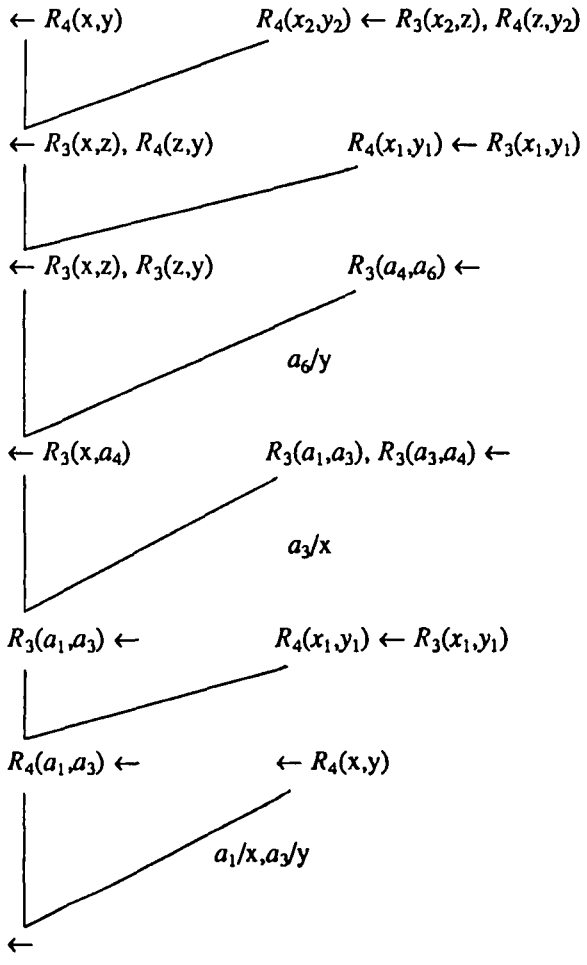
> If A is a ground atom, then ¬A is inferred from P by the GCWA if A is not in any minimal model of P.

The GCWA is now demonstrated by applying it to two new queries for Example 5.

5.4 ← ¬R_1(x)
By the GCWA the answers are a_2, a_5, a_7, and a_8.

5.5 ← R_4(x, y), ¬R_3(x, y)
By the GCWA there are no answers.

The GCWA is a powerful metarule that is consistent for disjunctive databases and reduces to the CWA for Horn databases.



Answer : <x,y> = <a_1,a_3> + <a_3,a_6>

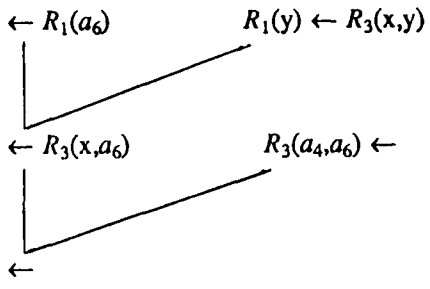
Figure 17

As shown by Lobo, Minker and Rajasekar (1988), to generalize the Completed Database (CDB), each disjunctive clause of P of the form $B_1, \ldots, B_m \leftarrow A_1, \ldots A_n$ is rewritten as the m Horn clauses

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

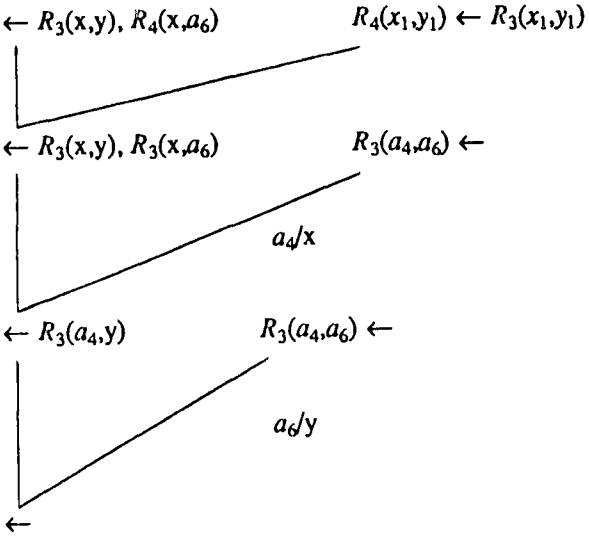
and then the completion formulas are applied to the Horn database so obtained. For Example 5 the CDB yields

$R_3(a_1, a_3), R_3(a_3, a_4) \leftarrow$
$R_3(a_4, a_6) \leftarrow$
$R_4(x, y) \leftarrow R_3(x, y)$
$R_4(x, y) \leftarrow R_3(x, z), R_4(z, y)$
$R_1(x), R_2(y) \leftarrow R_4(x, y)$
$R_1(y) \leftarrow R_3(x, y)$
$R_1(x_1) \rightarrow \exists y_1 R_4(x_1, y_1) \vee \exists y_2 R_3(y_2, x_1)$
$R_2(x_1) \rightarrow \exists y_1 R_4(y_1, x_1)$
$R_3(x_1, x_2) \rightarrow (x_1 = a_1 \,\&\, x_2 = a_3) \vee (x_1 = a_3 \,\&\, x_2 = a_4) \vee (x_1 = a_4 \,\&\, x_2 = a_6)$
$R_4(x_1, x_2) \rightarrow R_3(x_1, x_2) \vee \exists y_1 (R_3(x_1, y_1) \,\&\, R_4(y_1, x_2))$.



Answer : Yes

Figure 18



Answer : <x,y> = <a_4,a_6>

Figure 19

In typical cases of disjunctive databases the GCWA and CDB are equivalent. However, when the clauses of P are not logically independent, there may be differences. The version of the CDB presented here is equivalent to the Weak Generalized Closed World Assumption (see Ross and Topor (1988) and Rajasekar, Lobo and Minker (1989)). Consider the following example, where C has constants a and b, and the unary predicate symbol R. P is

$$R(a) \vee R(b) \leftarrow$$
$$R(a) \leftarrow.$$

In this case the GCWA yields $\neg R(b)$ (note that the second clause logically implies the first), but the additional clause for the CDB

$$R(x) \rightarrow R(a) \vee R(b)$$

assures the existence of a model where R(b) is true.

As in the case of Horn databases, both the GCWA and CDB are computationally complex methods. In analogy to SLDNF resolution, an extension of SLI resolution, called SLINF resolution was defined in Minker and Rajasekar (1989). The query evaluation procedure given there is sound with respect to the GCWA. Note that in solving for $\leftarrow \neg A$, it is no longer sufficient to solve for $\leftarrow A$, but both $\leftarrow A,K$ and $\leftarrow K$ must be solved where K is a sequence of atoms. This section ends by

showing the attempts to obtain the SLINF solutions for the query 5.5, (SLINF is not applicable to 5.4) in the manner of the SLI solutions for query 5.1 in Figures 20–22.

The result analogous to Theorem 8 is now the following.

Theorem 17 (Minker and Rajasekar, 1989). SLINF ⇒ CDB ⇒ GCWA.

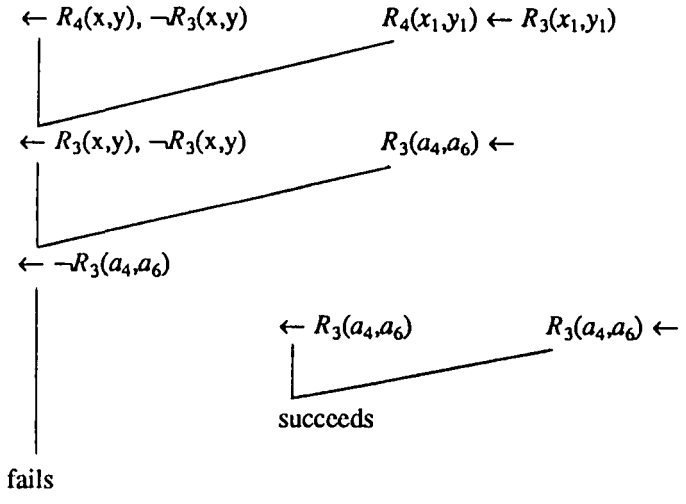


Figure 20

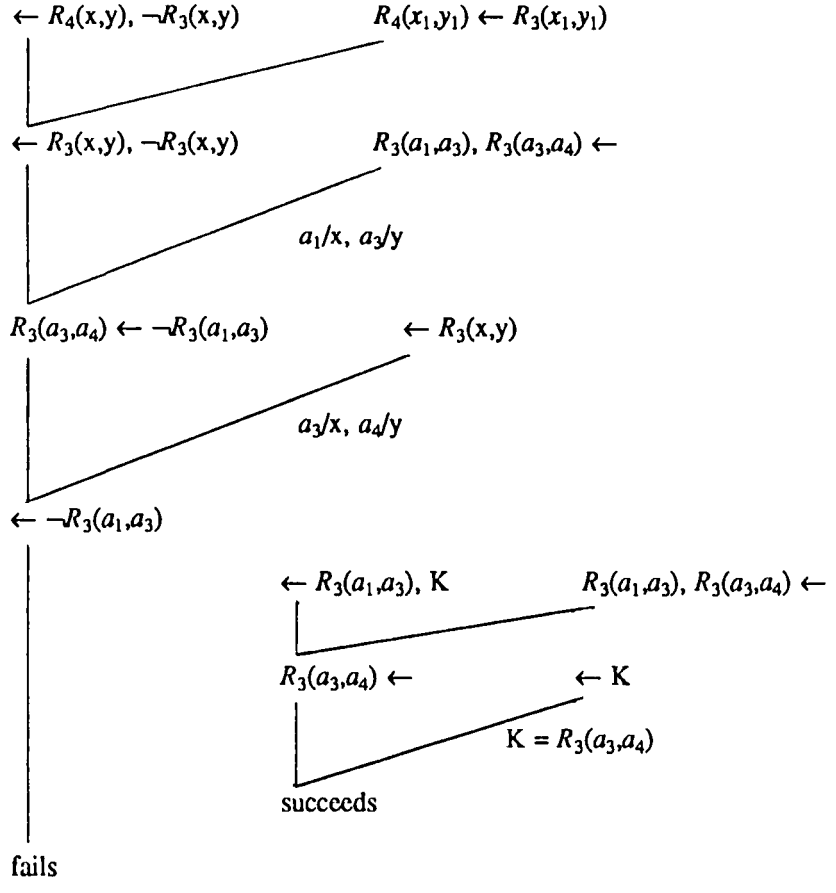


Figure 21

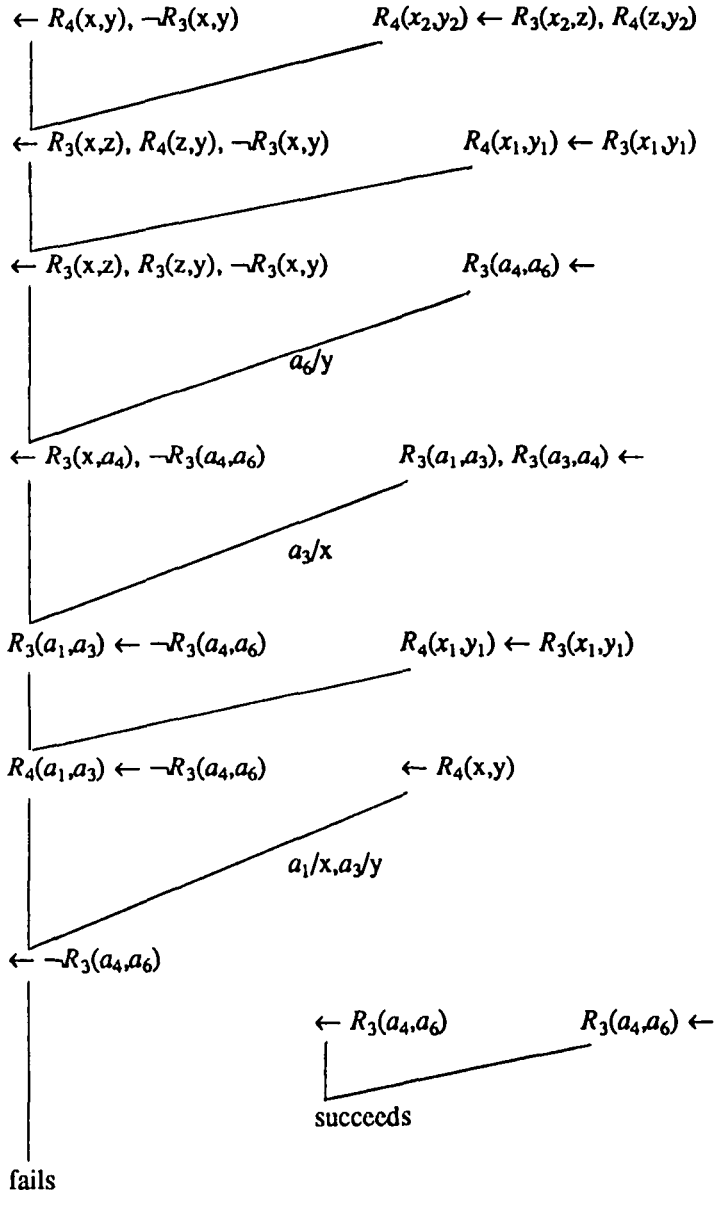


Figure 22

A2.6 Implementation of disjunctive databases

Several algorithms have been proposed in recent years for implementing disjunctive databases. Grant and Minker (1986) developed an algebraic method for testing if a candidate answer is actually an answer to a query, and used this method to obtain an algorithm for finding all the answers in a database that may contain disjunctive facts. Yahya and Henschen (1985) presented a deductive method to determine if a conjunction of ground atoms may be assumed false in a disjunctive database. Bossu and Siegel (1985) gave a different deductive method for answering queries in disjunctive databases. An alternative method for the solution of yes/no queries for disjunctive databases was defined in Henschen and Park (1988).

A3 Additional database theories

The previous sections presented some of the major results about the semantics and treatment of negation for various important database theories. This section contains briefer discussions of some extensions of those theories and their semantics. Typed theories and extended Horn theories

generalize the theories considered earlier. The well-founded approach uses a 3-valued logic to extend semantics to all normal databases.

A3.1 Typed databases

Some database researchers use typed theories in dealing with databases (Reiter, 1984). A typing may be applied to any of the database theories considered in the previous sections. The logical types are used to express the concept of a database domain. For example, in an application where the ternary predicate symbol R is used for the database relation Employee, it may be the case that the first argument represents an employee number, the second argument represents a name, and the third argument a salary. In a typed logic, each constant symbol has an associated type, and each variable symbol is written in a formula with its associated type. Then, in an interpretation, instead of a single domain, a typed logic has a domain for each type. In the case of the Employee relation represented by R, and assuming that τ_1, τ_2, and τ_3 are the types for employee number, name, and salary respectively, the query

$$\leftarrow \mathrm{R}(12345, \mathrm{y}/\tau_2, \mathrm{z}/\tau_3)$$

asks for the name and salary of the employee whose employee number is 12345. Note the restriction of y and z to τ_2 and τ_3 respectively.

While a typed logic may be convenient to use for databases, it is equivalent to an untyped logic with additional unary predicate symbols added to the language, one for each type. In the Employee example, call these predicate symbols R_1, R_2, and R_3. The query given above would be transformed to

$$\leftarrow \mathrm{R}(12345, \mathrm{y}, \mathrm{z}), R_2(\mathrm{y}), R_3(\mathrm{z}).$$

In the case of more general first-order logic formulas, such as for the extended Horn theories that will be considered next, a transformation is still possible. A universal quantifier in typed logic is transformed by using an implication: change "$\forall \mathrm{x}/\tau_1 \ldots$" to "$\forall \mathrm{x}(R_1(\mathrm{x}) \rightarrow \ldots$", and an existential quantifier by using a conjunction: change "$\exists \mathrm{x}/\tau_1 \ldots$" to "$\exists \mathrm{x}(R_1(\mathrm{x}) \,\&\, \ldots$". In general, the results for standard first-order logic continue to hold for typed logic with the appropriate modifications.

A3.2 Extended Horn theories

Extended Horn theories were introduced in the series of papers Lloyd and Topor (1984, 1985, 1986). The main results are included in Lloyd (1987) with the untyped version in Chapter 4 and the typed version in Chapter 5. This presentation uses the untyped version. In an extended Horn database both the notion of a clause and of a query are more general than the definitions given in the previous sections. In particular, an *extended clause* has the form $\mathrm{A} \leftarrow \mathrm{W}$, where the head A is an atom, and the body W is any formula. An *extended query* has the form $\leftarrow \mathrm{W}$ where W is any formula.

Extended clauses are useful in general logic programming because of the increased expressiveness that they allow in the definition of new predicates. For deductive databases extended queries are particularly useful in complex situations. In the case of the Employee relation presented above by the atomic formula R(x, y, z), but without using types, the query "Find the name of the employee with the highest salary" can be written using the extended query

$$\leftarrow \exists \mathrm{x} \exists \mathrm{z} \mathrm{R}(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \forall x_1 \forall y_1 \forall z_1 \, (z_1 \leqslant \mathrm{z} \leftarrow \mathrm{R}(x_1, y_1, z_1))$$

by also using the additional (built-in) predicate symbol $\leqslant$. An *extended Horn database* contains extended clauses.

The notion of stratification presented in the previous section is also useful for extended Horn databases. A level mapping for the predicate symbols provides a stratification if for every extended clause $\mathrm{A} \leftarrow \mathrm{W}$, with W in prenex conjunctive normal form (all quantifiers in front, the rest of the formula consisting of conjunctions of disjuncts, where each disjunct is a disjunction of literals), for

every predicate symbol R occurring positively in W, $r(R) \leqslant r(A)$ and for every predicate symbol R occurring negatively in W, $r(R) < r(A)$. With this definition, the results about the declarative and fixpoint semantics for stratified databases carry over to stratified extended Horn databases. In particular, there exists a minimal supported model for such a database.

The completed database (CDB) of an extended Horn database is defined in the following way. Suppose that the definition of the n-ary predicate symbol R is given by the extended Horn clauses $A_i \leftarrow W_i$, $1 \leqslant i \leqslant k$. Then, the completed database contains in addition to P, for each R the formula

$$\forall x_1 \ldots \forall x_n(\mathrm{R}(x_1, \ldots, x_n) \rightarrow E_1 \vee \ldots \vee E_k)$$

where E_i is $\exists y_1 \ldots \exists y_d((x_1 = t_1) \,\&\, \ldots \,\&\, (x_n = t_n) \,\&\, W_i)$, A_i is $\mathrm{R}(t_1, \ldots, t_n)$, $y_1, \ldots, y_d$ are the variables in A_i and the free variables in W_i, and $x_1, \ldots, x_n$ are variables not appearing anywhere in the definition of R. (Again, in the special case where there are no definitions for R, the CDB yields $\forall x_1 \ldots \forall x_n \neg \mathrm{R}(x_1, \ldots, x_n)$.)

Although, SLDNF-resolution cannot be used directly for the procedural semantics of extended Horn databases, it can be used in an indirect manner. This method transforms an extended Horn clause to a Horn clause by the use of 10 transformations (Lloyd, 1987). The transformation process is applied to both the extended clauses of the database and the query. This process is sound in the sense that if the transformed query is proved by SLDNF-resolution from the transformed theory, then the query formula is a logical consequence of the CDB.

It is useful at this point to introduce a condition on extended Horn databases (also applicable to Horn databases) that is more restrictive than stratification. An extended Horn database is *hierarchic* if there is a level mapping r such that for every extended clause $A \leftarrow W$ and for every predicate symbol R occurring in W, $r(R) < r(A)$. Essentially, a hierarchic database does not contain recursion. The equivalence of the declarative, fixpoint, and procedural semantics, where the latter is obtained by using the method just given, does not always hold for stratified extended Horn databases, but does hold for hierarchic extended Horn databases.

A3.3 The well-founded approach

The semantics of stratified databases is very useful and well-understood. However, there are normal databases that are not stratified and yet have an intuitively best minimal model. Consider the case where $C = \{a, b, c, R\}$ and the axioms (P):

$$\mathrm{R(a)} \leftarrow \neg \mathrm{R(b)}, \neg \mathrm{R(c)}$$
$$\mathrm{R(b)} \leftarrow \neg \mathrm{R(a)}$$
$$\mathrm{R(c)} \leftarrow .$$

This database is not stratified, but the model {R(b), R(c)} appears to be the best minimal model. The well-founded approach was defined in Van Gelder, Ross and Schlipf (1988); it generalizes the semantics of stratified databases to permit a semantics to be defined for every normal database. The presentation here follows the definitions in Przymusinski (1989). A key concept is the use of three-valued logic with the truth values true (1), undefined ($\frac{1}{2}$), and false (0). (It should be noted that this paper uses the terminology "well-founded approach" rather than "well-founded semantics" because semantics is used in the sense of declarative, fixpoint, and procedural, all of which are applicable in the well-founded case.)

A three-valued *interpretation* of a language is a function I: $HB_C \rightarrow \{1, \frac{1}{2}, 0\}$. It is convenient to write $I = \langle T,F \rangle$ where $T = \{A \in HB_C \mid I(A) = 1\}$ and $F = \{A \in HB_C \mid I(A) = 0\}$. An interpretation is two-valued if $T \cup F = HB_C$. An interpretation can be extended to formulas by using the following equations:

$$\mathrm{I}(\neg \mathrm{A}) = 1 - \mathrm{I(A)}$$
$$\mathrm{I(A \,\&\, B)} = \min(\mathrm{I(A)}, \mathrm{I(B)})$$
$$\mathrm{I(A \vee B)} = \max(\mathrm{I(A)}, \mathrm{I(B)})$$
$$\mathrm{I(B \leftarrow A)} = \{1 \text{ if } \mathrm{I(B)} \geqslant \mathrm{I(A)}, 0 \text{ otherwise}\}$$

An interpretation I is a *model* of the theory P if I(S) = 1 for every S ∈ P. Interpretations may be ordered using the pointwise ordering of functions: I ⩽ I' if I(A) ⩽ I(A') for all A ∈ HB_C. A *minimal model* is a model I such that there is no model I' with I' < I. Intuitively, a minimal model involves a minimization of the true atoms and a maximization of the false atoms.

The well-founded approach provides a minimal (3-valued) model for every theory. If the theory is stratified, this model is 2-valued and coincides with the one obtained in the previous section. However, many non-stratified normal theories also possess a well-founded 2-valued model. It is convenient to present the fixpoint semantics first.

Suppose that I is an interpretation (appropriate for the language of P), T ⊆ HB_C, $F \subseteq HB_C$. Define

$\mathbf{T}_I(T) = \{A \in HB_C \mid$ there is a ground instance of a clause in P, $A \leftarrow L_1, \ldots, L_m$, such that for every i, $1 \leqslant i \leqslant m$, either $I(L_i) = 1$ or $L_i \in T\}$.

$\mathbf{F}_I(T) = \{A \in HB_C \mid$ for every ground instance of a clause in P, $A \leftarrow L_1, \ldots, L_m$, there is an i, $1 \leqslant i \leqslant m$ such that either $I(L_i) = 0$ or $L_i \in F\}$.

Intuitively, I represents the facts currently known to be true or false, $\mathbf{T}_I(T)$ contains new facts whose truth can be derived immediately from P using I assuming that all elements of T are true, and $\mathbf{F}_I(T)$ contains new facts whose falsity can be derived immediately from P using I assuming that all elements of F are false. The operations $\mathbf{T}_I$ and $\mathbf{F}_I$ are monotonic and can be iterated to a fixpoint as follows:

Let I = ⟨T,F⟩ be an interpretation. Define:

$T_I \uparrow 0 = \varnothing$, $F_I \downarrow 0 = HB_C$
$T_I \uparrow n + 1 = \mathbf{T}_I(T_I \uparrow n)$, $F_I \downarrow n + 1 = \mathbf{F}_I(F_I \downarrow n)$,
$T_I = \cup \{T_I \uparrow n \mid n < \omega\}$, $F_I = \cap \{F_I \downarrow n \mid n < \omega\}$.

Intuitively, T_I contains the new atomic facts derivable from I by P and F_I contains the new atomic facts whose falsity can be assumed from I using P. The operator T_P is then defined on an interpretation as $T_P(I) = I \cup \langle T_I, F_I \rangle$. The models for P are defined as follows:

$M_0 = \langle \varnothing, \varnothing \rangle$
$M_i = T_P(M_{i-1})$.

The interpretation M_P is taken as the least fixed point of the operator T_P, say $M_m = M_P$. This is a minimal (3-valued) model of P which is called the *well-founded* model of P. In the case of the example given earlier, $M_0 = \langle \varnothing, \varnothing \rangle$, $M_1 = \langle \{R(c)\}, \varnothing \rangle$, and $M_2 = \langle \{R(c),R(b)\}, \{R(a)\} \rangle$ is the well-founded model.

Przymusinski (1989) develops a dynamic stratification for every database theory containing clauses of the form $A \leftarrow L_1, \ldots, L_n$ by using the construction given above. First, the original theory is transformed by instantiating the variables by all combinations of the constants. Thus, the new theory contains only ground atoms in the head of each clause. Levels are given to ground atoms, not predicate symbols as follows: r(A) = i if A ∈ M_i and not A ∈ M_j for any j < i, that is, if the truth or falsity of A is determined in M_i but not in M_j for any j < i. In the case of a 3-valued model ($M_P = M_n$), all A such that not A ∈ M_P are given the level n + 1. The strata for the clauses of P are defined by the level of the atom in the head of the clause, that is,

$A \leftarrow L_1, \ldots, L_n$ is placed in stratum P_i if r(A) = i.

In particular, in the above example, r(R(c)) = 1, r(R(a)) = r(R(b)) = 2.

This dynamic stratification can be used to describe a declarative semantics that constructs the well-founded model as an iterated least model and which coincides with the fixpoint semantics.

M_i = the least model of $\cup \{P_j \mid j < i\}$ which extends M_{i-1} to the atoms in the levels up to i and gives undefined values to all the atoms not in a level ⩽i.

Finally, it is possible to extend SLS-resolution to give a procedural semantics for the well-founded

model. The major difference between the extended SLS-resolution and the SLS-resolution presented earlier for stratified theories is the ability of the former to handle undefined answers. The details may be found in Przymusinski (1989). The three semantics are equivalent for normal databases and provide identical answers to non-floundering queries.

Acknowledgement

We wish to thank the editor, Dr John Fox, the referees, and Jorge Lobo for numerous very useful comments.

References

1.1 General references on deductive databases

Gallaire, H and Minker J, Eds, 1978. *Logic and Databases*, New York: Plenum.

Gallaire, H, Minker, J and Nicolas, J, 1984. "Logic and databases: A deductive approach" *ACM Computing Surveys* **16** 153–185.

Minker, J, 1987. "Deductive databases: An overview of some alternative theories" In Ras, ZW and Zemankova, M, Eds, *Methodologies for Intelligent Systems*, Amsterdam: Elsevier, pp. 148–158.

Minker, J, Ed., 1988. *Foundations of Deductive Databases and Logic Programming*, California: Morgan Kaufmann.

Minker, J, 1988a. "Perspectives in deductive databases" *The Journal of Logic Programming* **5** 33–60.

Ullman, JD, 1988. *Principles of Database and Knowledge-Base Systems*, Volumes I and II, Maryland: Computer Science Press.

1.2 Books on logic, theorem proving, and logic programming

Chang, CL and Lee, RTC, 1973. *Symbolic Logic and Mechanical Theorem Proving*, New York: Academic Press.

Enderton, HB, 1972. *A Mathematical Introduction to Logic*, New York: Academic Press.

Lloyd, JW, 1987. *Foundations of Logic Programming*, Second Extended Edition, New York: Springer-Verlag.

Loveland, D, 1978. *Automated Theorem Proving: A Logical Basis*, Amsterdam: Elsevier North-Holland.

Mendelson, E, 1978. *Introduction to Mathematical Logic*, 2nd Ed., New York: van Nostrand-Reinhold.

2 Relational and deductive Horn databases

Grant, J and Minker, J, 1990. "Integrity constraints in knowledge based systems", In Adeli, H, Ed., *Knowledge Engineering*, Vol. II, Applications, New York: McGraw-Hill, pp. 1–25.

Nicolas, J and Gallaire, H, 1978. "Data base: Theory vs interpretation" In Gallaire, H and Minker, J, Eds, *Logic and Databases*, New York: Plenum, pp. 33–54.

3 Declarative, fixpoint, and procedural semantics

van Emden, MH and Kowalski, R, 1976. "The semantics of predicate logic as a programming language" *Journal of the ACM* **23** 733–742.

4 Negation

Chan, C, 1988. "Constructive negation based on the completed database", In Kowalski, RA and Bowen, KA, Eds, *Logic Programming Proceedings of the Fifth International Conference and Symposium*, Massachusetts: The MIT Press, pp. 111–125.

Clark, KL, 1978. "Negation as failure" In Gallaire, H and Minker, J, Eds, *Logic and Databases*, New York: Plenum, pp. 293–322.

Reiter, R, 1978. "On closed world databases" In Gallaire, H and Minker, J, Eds, *Logic and Databases*, New York: Plenum, pp. 149–178.

Shepherdson, JC, 1988. "Negation in logic programming" In Minker, J, Ed., *Foundations of Deductive Databases and Logic Programming*, California: Morgan Kaufmann, pp. 19–88.

5 Stratified databases

Apt, KR, Blair, HA and Walker, A, 1988. "Towards a theory of declarative knowledge" In Minker, J, Ed., *Foundations of Deductive Databases and Logic Programming*, California: Morgan Kaufmann, pp. 89–148.

Chandra, A and Harel, D, 1985. "Horn clause queries and generalizations" *The Journal of Logic Programming* **2** 1–15.

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

Naqvi, SA, 1986. "A logic for negation in database systems" In Minker, J, Ed., *Proceedings of the Workshop on Foundations of Deductive Databases and Logic Programming* Washington, D.C., pp. 378–387.

Naqvi, SA and Tsur, S, 1989. *A Logical Language for Data and Knowledge Bases*, Maryland: Computer Science Press.

Przymusinski, TC, 1988a. "On the declarative semantics of deductive databases and logic programs", In Minker, J, Ed., *Foundations of Deductive Databases and Logic Programming*, California: Morgan Kaufmann, pp. 193–216.

Przymusinski, TC, 1988b. "On the declarative and procedural semantics of logic programs" *Journal of Automated Reasoning* **4**.

Van Gelder, A, 1988. "Negation as failure using tight derivations for general logic programs" In Minker, J, Ed., *Foundations of Deductive Databases and Logic Programming*, New York: Morgan Kaufmann, pp. 149–176.

A1 Protected databases

Minker, J and Perlis, D, 1985. "Computing protected circumscription" *The Journal of Logic Programming* **4** pp. 235–249.

A2 Disjunctive databases

Bossu, G and Siegel, P, 1985. "Saturation, non-monotonic reasoning and the closed-world assumption" *Artificial Intelligence* **25** 13–63.

Grant, J and Minker, J, 1986. "Answering queries in indefinite databases and the null value problem" In Kanellakis, P, Ed., *Advances in Computing Theory, Vol. 3, The Theory of of Databases*, JAI Press, pp. 247–267.

Henschen, LJ and Park, H-S, 1988. "Compiling the GCWA in indefinite deductive databases", In Minker, J, Ed., *Foundations of Deductive Databases and Logic Programming*, California: Morgan Kaufmann, pp. 295–438.

Lobo, J, Minker, J and Rajasekar, A, 1988. "Weak completion theory for non-Horn programs", In Kowalski, RA and Bowen, KA, Eds, *Logic Programming Proceedings of the Fifth International Conference and Symposium*, Massachusetts: MIT Press, pp. 828–842.

Minker, J, 1982. "On indefinite databases and the closed world assumption", In *Proc. of the 6th Conference on Automated Deduction*, Springer-Verlag Lecture Notes in Computer Science No. 138, New York: Springer-Verlag, pp. 292–308.

Minker, J and Rajasekar, A, 1989. "A fixpoint semantics for non-Horn logic programs" *The Journal of Logic Programming.*

Minker, J and Zanon, G, 1982. "An extension to linear resolution with selection function" *Information Processing Letters* **14** pp. 191–194.

Rajasekar, A, Lobo, J and Minker, J, 1989. "Weak generalized closed world assumption", *Journal of Automated Reasoning* **5** 293–307.

Ross, KA and Topor, RW, 1988. "Inferring negative information from disjunctive databases" *Journal of Automated Reasoning* **4** 397–424.

Yahya, A and Henschen, LJ, 1985. "Deduction in non-Horn databases" *Journal of Automated Reasoning* **1** 141–160.

A3.1 Typed databases

Reiter, R, 1984. "Towards a logical reconstruction of relational database theory", In Brodie, ML, Mylopoulos, J and Schmidt, JW, Eds, *On Conceptual Modelling*, New York: Springer-Verlag, pp. 191–233.

A3.2 Extended Horn databases

Lloyd, JW and Topor, RW, 1984. "Making Prolog more expressive" *Journal of Logic Programming* **1** 225–240.

Lloyd, JW and Topor, RW, 1985. "A basis for deductive database systems" *Journal of Logic Programming* **2** 93–109.

Lloyd, JW and Topor, RW, 1986. "A basis for deductive database systems II" *Journal of Logic Programming* **3** pp. 55–67.

A3.3 The well-founded approach

Przymusinski, TC, 1989. "Every logic program has a natural stratification and an iterated least fixed point model" In *Proc. of ACM PODS* 11–21.

Van Gelder, A, Ross, KA and Schlipf, JS, 1988. "Unfounded sets and well-founded semantics for general logic programs" In *Proc. of ACM PODS* 221–230.