1988

Intensional Query Processing in Deductive Database Systems.

Il Yeol Song
Louisiana State University and Agricultural & Mechanical College

Order Number 8904569

Intensional query processing in deductive database systems

Song, Il Yeol, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1988

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

INTENSIONAL QUERY PROCESSING IN DEDUCTIVE DATABASE SYSTEMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by

Il Yeol Song

B.E., HanYang University, Seoul, Korea, 1975
M.S., Louisiana State University, 1984

August 1988

Acknowledgements

I owe a great deal to my advisor, Professor Peter P. Chen, for his encouragement, help, patience, and guidance throughout my research which has allowed me to successfully complete my degree program. For many years it has been a pleasure and privilege to work with him. His insight and guidance have been of great help during the many stages of my graduate studies, either directly or indirectly.

I would like to thank Professor Donald H. Kraft, who taught me the concepts of databases for the first time at LSU and stimulated me in class as a professor and in the department as a graduate coordinator and chairman. I am especially grateful for his support of my graduate studies.

I would also like to acknowledge my debts and thanks to Professor Sukhamay Kundu who provided valuable insights and long discussions at the various stages of my work. Professor Kundu helped me refine Definition 5.7.2 and clarify other concepts in Chapter 7. It is an honor to acknowledge him as one of the best teacher in my whole educational experience.

I owe thanks to Professor Charles M. Parks who encouraged me and spent a lot of time helping me in various ways. Thanks are also due to Professors Leslie Jones and Dan Rinks who served on my committee and read my work despite their tight schedules. In addition, I would like to extend my appreciation to Professor Arie Tzvieli for his timely encouragement at the earlier stages of my work and for his helpful comments; Professor S. Sitharama Iyengar for his frequent encouragement and advice; and Professor Walter G. Rudd for supporting my graduate studies at my first

year at LSU.

I also owe thanks to Mrs. Andrea K. Martin who supervised my teaching for the last two years and made it a pleasant and valuable experience. I would like to thank all the other faculty members, staff, and fellow graduate students in our department who helped my education become a reality.

I must not forget thanks to my friend, Mr. Shangyong Zhu, for his willingness to give his time for discussion and for his many valuable comments. Dr. Sung-Wu Suh in the department of Mathematics also deserves special thanks for his deep warmth and affection to me and my family during my studies at LSU.

Finally, I wish to thank my wife, Young-choon, and my daughters, Jaewon and Sharon, for their support and patience while I was away working with the computer.

Table of Contents

List of Figures

List of Symbols

$\forall$	Universal quantifier
$\exists$	Existential quantifier
$\vee$	Disjuction
$\wedge$	Conjunction
$\rightarrow$	Implication
$\leftrightarrow$	Logical equivalence
$\leftarrow$	A symbol for a negative clause, used like :- in Prolog
$\neg$	Negation
$\vdash$	Syntactic implication
$\square$	The empty clause
$\blacksquare$	The end of a proof
θ	Most general unifier
$ANS_E(Q)$	The set of extensional answers for a query Q(X)
$ANS_E'(Q)$	The set of extensional answers computed by Axiom (7.1.1)
$ANS_I(Q)$	The set of intensional answers for a query Q(X)
$ANS_I^m(Q)$	The set of minimal intensional answers for a query Q(X)
$ANS_I^{mf}(Q)$	The set of meaningful intensional answers for a query Q(X)
C_{nr}	The set of non-relevant clauses to a query
C_r	The set of relevant clauses to a query
G	A goal clause

$L_{ans_I}(X)$	The set of literals contained in $ans_I(X)$
L_n	The set of relevant clauses to a given query
L_{nr}	The set of non-relevant clauses to a given query
L_p	The set of pure clauses to a query
L_{np}	The set of non-pure clauses to a query
L_Q	The set of literals contained in a query Q(X)
Q(X)	A query
Res(G, C)	Resolution between clauses G and C
R^i	The i^{th} resolvent from the goal G, where R^0 is a query clause
R^n	The last resolvent in a branch of a resolution tree
S	A set of clauses converted from a database theory T
SF	A selection function in SLD resolution
T	A database theory T that consists of an EDB and an IDB
x0	A tuple with Skolem constants
$ans_E(X)$	An extensional answer or Green's Literal
$ans_I(X)$	An intensional answer
$ext(ANS_I^{mf}(Q))$	A set of extensional answers computed by evaluating all meaningful intensional answers against an EDB
$ext(ans_I(X))$	A set of extensional answers computed by evaluating an intensional answer
iff	if and if only

Abstract

This dissertation addresses the problem of deriving a set of non-ground first-order logic formulas (intensional answers), as an answer set to a given query, rather than a set of facts (extensional answers), in deductive database (DDB) systems based on non-recursive Horn clauses.

A strategy in previous work in this area is to use resolution to derive intensional answers. It leaves however, several important problems. Some of them are: no specific resolution strategy is given; no specific methodologies to formalize the meaningful intensional answers are given; no solution is given to handle large facts in extensional databases (EDB); and no strategy is given to avoid deriving meaningless intensional answers.

As a solution, a three-stage formalization process (pre-resolution, resolution, and post-resolution) for the derivation of meaningful intensional answers is proposed which can solve all of the problems mentioned above. A specific resolution strategy called SLD-RC resolution is proposed, which can derive a set of meaningful intensional answers. The notions of relevant literals and relevant clauses are introduced to avoid deriving meaningless intensional answers. The soundness and the completeness of SLD-RC resolution for intensional query processing are proved. An algorithm for the three-stage formalization process is presented and the correctness of the algorithm is proved.

Furthermore, it is shown that there are two relationships between intensional answers and extensional answers. In a syntactic relationship, intensional answers are sufficient conditions to derive extensional answers. In a semantic relationship, intensional answers are sufficient and necessary conditions to derive extensional answers. Based on these relationships, the notions of the global and local completeness of an intensional database (IDB) are defined. It is proved that all incomplete IDBs can be transformed into globally complete IDBs, in which all extensional answers can be generated by evaluating intensional answers against an EDB.

We claim that the intensional query processing provide a new methodology for query processing in DDBs and thus, extending the categories of queries, will greatly increase our insight into the nature of DDBs.

CHAPTER 1

INTRODUCTION

1.1. Introduction

The recent success of expert systems has motivated the research in expert database systems or knowledge base management systems [Kers84, Kers85, Kers86, Mink86]. Expert database systems combine technologies of both expert systems and database management systems (DBMS) [Smit84]. Expert systems can improve their efficiency by introducing the DBMS's technologies, such as the efficient retrieval and manipulation of large shared data. DBMSs can also extend their processing power by introducing artificial intelligence technologies such as deductive capabilities, reasoning capabilities, and knowledge representation techniques. Thus, expert database systems are considered a natural evolution of DBMSs which provide highly efficient management of large, shared knowledge base for knowledge-directed systems [Kers85].

Recently, there has been a lot of research in emerging fields, such as the architectures of knowledge base management systems, deductive reasoning for expert database systems, deductive database systems incorporating the functionality of both logic programming and database systems, and intelligent user interfaces [Gall84, Miss84, Kers85]. Here, we are particularly interested in deductive database systems.

A *Deductive Database* (DDB) is a database in which new facts may be derived from the facts that were explicitly stored, using an inference system [Gall84].

Some of the earlier research on DDB focuses on the theory of DDBs [Reit78b, Clar78, Mink82, Reit84] and the evaluation of non-recursive queries [Reit78a, Mink78, Chan78, Kell78]. Some of the recent research in the DDBs concentrates on the efficient realization of recursive queries [Hens84, Ullm85, Lozi85, Viei86, Yoko86, Banc86a, Han86, Banc86b, Balb87, Sacc87], the improvement of Prolog systems incorporating the super set of Horn clauses [Lloy84, Lloy85, Lloy86, Gabb84, Gall87], efficient ways to communicate between Prolog and database systems [Zani84, Chom83, Luca85, Ceri86], interfacing Prolog with existing database systems [Ioan84, Chan84, Raji86, Bocc86], rule-based query optimizations [Hamm80, King81, Chak84, Chak86a, Chak86b, Shen87, Grae87, Frey87], the issues of integrity constraints [Nico82, Asir85, Deck86], the deductive database systems and models [Gett84, Kell84, Kell86, Spyr87], and other theoretical aspects [Zani86, Naqv86, Warr86, Hens86, Topo86, Przy86, Lifs86, Hens86].

Some of the recent interesting work in DDBs are Cholvy and Demolombe's [Chol86] and Imielinski's [Imie87] work. Cholvy and Demolombe study the notion of having a set of formulas as an answer set, while Imielinski studies the idea of incorporating the intensional predicates as a part of the answers. Cholvy and Demolombe use resolution to generate answer formulas for a query given in a logical formula, while Imielin uses relational algebra to transform the rules and to process a query given in relational algebra expressions.

A fundamental difference of their approaches from conventional database systems is that all or part of their answers to a query consist of a set of first-order logic formulas, rather than a set of facts.

1.2. Informal Presentation *

Queries in deductive database (DDB) systems can be classified by many criteria. Song [Song87] provides an extensive classification of various query types available in deductive database systems. Appendix A briefly summarizes them. The criteria considered are the type of answers, free/bound information given in the queries, the preferred reasoning schemes for a given query, the types of predicates used in a query, and the types of rules in an intensional database.

In this dissertation, we classify the types of queries available in a DDB by the types of answers for a given query. We distinguish two types of answers from a given query in a DDB: a set of facts and a set of non-ground first-order logic formulas. We call the former type *extensional answers* and the latter type *intensional answers*. We also define *extensional queries* as the type of queries which have extensional answers as in the conventional database systems, and *intensional queries* as the type of queries which have intensional answers. One of the trivial query types, according to our classification, is the *yes/no* query type, which does not have any variables in a query. The answer set in this case is simply *yes* or *no*. Thus, in this work, we will not be further concerned about this type of queries.

Some queries are purely extensional, while others could be both extensional and intensional. In the latter case, we can easily imagine that there exist certain relationships between the two types of answers. In this case we argue that, in general, intensional answers are the conditions which extensional answers must satisfy for a

* Terminology used in Section 1.2 and 1.3 is discussed in Section 2.1.

given query under the current database.

For example, suppose a deductive database system has a deductive rule which defines a *grandfather* relationship in terms of *father* as follows:

gf(X, Y) ← father(X, Z), father(Z, Y)

where **father(X, Z)** means *X is the father of Z* and **gf(X, Y)** means *X is the grandfather of Y*. Suppose we have two facts, **father(a, b)** and **father(b, c)**. In a conventional DDB, the answer for the query **gf(X, Y)?**, asking *who is the grandfather of whom?*, will be **gf(a, c)**, saying that *a is a grandfather of c*. However, we want to get **∃Z father(X, Z) ∧ father(Z, Y)**, saying that *there exists a Z such that X is the father of Z and Z is the father of Y*, as an answer in an intensional query processing system. The answer *<a, c>* is an extensional answer, while **∃Z father(X, Z) ∧ father(Z, Y)** is an intensional answer. Hence, we can easily see that the extensional answer *<a, c>* must satisfy the formula **∃Z father(X, Z) ∧ father(Z, Y)** in order to be the answer for the query **gf(X, Y) ?**.

The intensional answers are not dependent on the particular instances of the database, unless a query has bounded information, as in **gf(a, Y)?**. For example, the answer **gf(a, c)** is dependent on the instances, **father(a, b)** and **father(b, c)** in the database. However, the answer **∃Z father(X, Z) ∧ father(Z, Y)** is not dependent on those two tuples, **father(a, b)** and **father(b, c)**. Thus, the intensional queries are useful when we query the general rules captured in the database. They are also useful when extensional answers consist of a large set of facts. In this case we can display the output in a more compact way. Furthermore, we can show what conditions or what

formulas constitute the answer. This will help with the interpretation of extensional answers.

We note that these advantages give additional power to a DDB over conventional database systems. We claim that the capability of intensional query processing extend the category of query types available in a DDB and significantly help the interpretation of queries.

We present several motivational examples of intensional answers and summarize their advantages in the next section.

1.3. Motivational Examples

Intensional answers are useful when we want to know the answer in terms of database rules, because the answers are independent of any particular database state. Intensional answers consist of formulas $\phi_1, \cdots, \phi_n$ such that any data that satisfies one of the formula ϕ_i also satisfies the query. The formula ϕ_i can display the output to a query in a more compact form than the corresponding extensional answers, which may consist of a large set of facts. The intensional answers can also be computationally advantageous as well, when the answers to a query can be generated without accessing an extensional database.

The following example illustrates the notion of intensional answers. We will show an EDB schema, rules, queries, and intensional answers only. Processing intensional queries, in general, does not require the EDB tuples. An intensional query processing strategy is discussed in Chapter 6.

Example 1.3.1:

Suppose we have a database, for students, consisting of 3 relations and 4 rules as follows:

EDB Schema:

(d1): student(Sname, Year)
(d2): course(Cno, Cname, CreditHr)
(d3): has_taken(Sname, Cno)

IDB:

(r1): pre_req(Sname, 4402) ← has_taken(Sname, 3102)
(r2): cannot_take(Sname, Cno) ← student(Sname, 1), Cno = 3102.
(r3): cannot_take(Sname, Cno) ← ¬pre_req(Sname, Cno)
(r4): cannot_take(Sname, Cno) ← student(Sname, 1), Cno > 4000.

The relation *student* has information about students' names (Sname) and their number of years of study (Year). The relation *course* has course numbers (Cno), course names (Cname), and their number of credit hours (CreditHr). The relation *has_taken* describes who (Sname) has taken what courses (Cno). On the other hand, the rule (r1) states that *course number 3102 is a prerequisite for course 4402*. The rule (r2) states that *a freshman cannot take the course 3102*. The rule (r3) states that *if the student does not satisfy the prerequisite of a course, then he/she cannot take that course*. The rule (r4) states that *freshmen cannot take courses above the 4000 level*

Suppose we have a query **cannot_take(X, 4402) ?**, asking that *who cannot take 4402?*. In a conventional database system which computes only extensional answers, the answer might be a long list of student names who either did not take 3102 or who are freshmen. However, If a database system can process intensional queries, the answers can be represented as a set of simple formulas such as

$$\mathbf{ans^1 = student(X, 1)}$$

$$\mathbf{ans^2 = \neg\ has\_taken(X, 3102)}$$

The **ans**^{1} implies that *if any student is a freshman, then hee cannot take 4402*. The second answer **ans**^{2} implies that *if any student has not taken 3102, then hee cannot take 4402*.

We can see from this example that while the intensional answers show in what cases students cannot take 4402, extensional answers do not distinguish between those cases in which students are freshmen and those in which they have not taken 3102. Thus, by telling us what conditions or formulas constitute the answers, intensional answers in some cases are more informative.

Let us look at another example.

Example 1.3.2:

We will use the same rules shown in Example 1.3.1. Suppose we have a database which contains a fact **student(david, 1)**, meaning *David is a freshman*, and a query asking *what courses David cannot take?*. In this query, the extensional answers consist of a list of all the course numbers which are greater than 4000, by the rule (r4), and the course number 3102, by the rule (r2). In intensional answers, however, the answers are compact and simple formulas:

$$\mathbf{ans^1 = eq(Cno, 3102)}$$
$$\mathbf{ans^2 = gt(Cno, 4000)}$$

They can be interpreted as *if the course number is equal to 3102, then David cannot take it*, and *If the course number is greater than 4000, then David cannot take it*, respectively. For the most part, this answer set is simple and clear enough for the

query above, unless users are interested in all the names of courses which David cannot take.

As we can see from these examples, there are many cases where intensional answers may be more compact and informative. Intensional answers can also help us interpret extensional answers, and thus could be a way of providing intelligent user interface.

From these motivational examples, we can summarizes the advantages of having intensional answers as follows:

(1) Intensional answers are represented in terms of the general rules of the database which are independent of any particular database state.

(2) Intensional answers tell us what conditions or formulas constitute extensional answers. Thus, intensional answers can help interpret extensional answers.

(3) When the number of extensional answers is large and they can be represented by first-order logic formulas, the corresponding intensional answers are compact and clear enough, unless users are interested in all the extensional answers. If users are interested in specific factual answers (extensional answers), they can request the system to evaluate the intensional answers to generate extensional answers.

(4) When intensional answers can be derived without accessing the EDB or when users are satisfied with intensional answers, intensional query processing (IQP) has a computational advantage.

(5) IQP is a new methodology for query processing in a DDB. It treats rules as data and it can change the way we view the answers to a given query. Thus, IQP extends the category of queries.

1.4. The Problems

This dissertation describes a deductive database system which derives, as an answer set, a set of non-ground first-order logic formulas, rather than a set of facts. The work herein not only constitutes an extension of the attempt begun in [Chol86], but also provides new relationships between intensional answers and extensional answers. More specifically, we will attempt to solve the following problems along with intensional query processing:

(1) For IQP, we will adopt SLD resolution which is known to be one of the most efficient resolution methods for Horn clause systems. Thus, we will study how we can apply SLD-like resolution for IQP.

(2) When we adopt SLD-like resolution for IQP, what are the methodologies and procedures to remove redundant resolution steps and simplify a derived intensional answer set ?

(3) When we have rules whose heads are comparison literals, the blind application of resolution with these rules may derive meaningless intensional answers. In these cases, how can we avoid generating meaningless intensional answers or avoid resolution among non-interesting predicates ?

(4) How can we handle a large number of facts in an extensional database in IQP ? Cholvy and Demolombe converted all the facts to rules. This strategy is neither

efficient nor practical when we have a large set of facts.

(5) For a given query, what are the relationships between extensional answers and intensional answers ?

(6) What are the relationships between intensional answers and intensional databases ?

(7) Usually, rules in DDBs need to be in range-restricted forms. What forms of rules do we need for intensional query processing ?

These are the specific research questions which we have in this research.

1.5. Assumptions

The assumptions we have made in this work are listed below.

(1) All the rules in an IDB are non-recursive Horn clauses. One of the inherent difficulties in deriving intensional answers occurs when the IDB contains recursive rules. This problem is discussed in Chapter 8.

(2) There are no function symbols in the rules. This is a standard assumption in a DDB community [Gall84]. If a set of clauses does not contain any function symbols, the decision procedure for satisfiability is decidable, but it is semidecidable otherwise.

(3) The constants appearing in a query clause exist in a database. By this assumption, we are removing the possibility of processing trivially meaningless queries asking about objects that do not exist in a database.

(4) Literals are either intensionally defined or extensionally defined. For a given literal p which is both EDB-defined and IDB-defined, Minker and Nicolas

[Mink83] show that one can always obtain such a partition by renaming the extensional literal p^*, and introducing a new rule $p \leftarrow p^*$ in the IDB. It is called *the assumption of the unique intensional literals* and further discussed in Section 5.2.

1.6. Organization of This Dissertation

Chapter 2 describes relevant terminology about first-order logic, resolution, and deductive database systems for the discussion of intensional query processing. Also a brief summary on the classification of query types is included. Finally, previous work in intensional query processing is explained and their problems are summarized.

Chapter 3 formally defines extensional answers, intensional answers, and intensional queries.

Chapter 4 explains the motivation for adopting SLD-like resolution for intensional query processing and shows the problems in adopting a simple-minded SLD resolution, called SLD-IA resolution, for intensional query processing. In addition, an SLD-EA tree, which derives a set of extensional answers, is defined.

Chapter 5 proposes a three-stage formalization process (pre-resolution, resolution, post-resolution) that leads to the four restrictions imposed on intensional answers to make them meaningful. Based on these four restrictions, a set of meaningful intensional answers and a set of minimal intensional answers to a given query are formally defined.

Chapter 6 introduces SLD-RC resolution which can derive a set of meaningful intensional answers based on the formalization process described in Chapter 5.

Important properties of SLD-RC resolution are also proved in this chapter. The finiteness of an SLD-RC tree, the soundness, and the completeness of an SLD-RC tree for intensional query processing are proved. Also presented are an algorithm for intensional query processing and the correctness of the algorithm.

Chapter 7 reviews the definitions of extensional answers and intensional answers and introduces new logical definitions of extensional answers and intensional answers. Then two relationships, called a syntactic relationship and a semantic relationship, between intensional answers and extensional answers are derived. In addition, based on these two relationships, the notions of the global completeness of an IDB and the local completeness of an IDB to a given query are discussed. Then the transformation of an incomplete IDB to a globally complete IDB is proved in this chapter.

Chapter 8 shows the problems of including recursive rules in intensional query processing. Chapter 9 summarizes the achievements of this work, the advantages of intensional answers, and provides some directions for future work.

CHAPTER 2

PRELIMINARIES

2.1. Terminology

We provide background material for the discussion of intensional query processing (IQP) to make this dissertation self-contained. Well-written introductions to first-order logic are described in [Chan73, Love78] and connections between first-order logic and database are described in [Gall78a, Gall84]. We refer to these sources for more detailed descriptions.

We adopt the Prolog convention to denote predicate names, variables, and constants. Those strings of characters starting with a upper case letter denote variables and those starting with a lower case letter denote the predicate names and string constants. For example, in **student(david, X)**, *student* is a predicate name, *david* is a string constant, and *X* is a variable.

The *comparison predicate name* is a predicate name that can be compared using their arguments. Examples are *equal, greater-than*, and *less-than*. When all of the arguments of a comparison predicate have instantiated values, we say the comparison predicate is *evaluable*. For example, **gt(X, 20)** is not evaluable, while **gt(20, 15)** is evaluable.

Terms are defined recursively as follows: i) a constant is a term; ii) a variable is a term; iii) if **f** is a n-ary function symbol and $t_1, \cdots, t_n$ are terms, then $f(t_1, \cdots, t_n)$ is a term; iv) there are no other terms. For example, *3, david, X,* and *f(X,20)* are terms,

while **student(david, 1)** is not a term. However, a term in the context of databases is assumed to be function-free; that is, it is either a constant or a variable.

An *atom* or an *atomic formula* is of the form $p(t_1, \cdots, t_n)$, where **p** is a predicate name of *arity* **n** and each t_i is a constant or a variable. A *literal* is an atom or the negation of an atom. A *positive literal* is an atom. A *negative literal* is the negation of an atom. For example, **father(john, tom)** is an atom, and a positive literal, with arity 2, while $\neg$ **father(john, jack)** is a negative literal. We will represent a literal without arguments when the existence of arguments is not important for the discussion of problems.

Well-formed formulas (WFF) or *formulas* in first-order logic are defined recursively as follows: an atom is a formula; if **F** and **G** are formulas, then $\neg$ **(F)**, $(F \vee G)$, $(F \wedge G)$, $(F \rightarrow G)$, and $(F \leftrightarrow G)$ are formulas; if **F** is a formula and **X** is a free variable in **F**, $(\forall X)F$ and $(\exists X)F$ are formulas; and nothing else is a formula. For example, let **p** and **r** be binary predicates and **q** be a unary predicate. Let **a** and **b** be constants, and **X** and **Y** be variables. Then $\neg p(X, a) \rightarrow q(b)$ is a formula, while $r(X, (p(a, c) \rightarrow q(r)))$ is not a formula since its second argument is not a term.

A variable is *bound* iff the occurrence of **X** in a formula is in the scope of quantifier (either $\forall$ or $\exists$), otherwise **X** is *free*. For example, in a formula $\forall X \exists Y (p(X, Y) \vee q(Y, Z))$, **X** and **Y** are bound, while **Z** is free.

A formula is *closed* iff all variables are bound. For example, $\exists X \exists Y\ q(X, Y)$ is a closed formula, while **q(X, Y)** is not a closed formula (it is often called *open* formula).

A formula is *ground* if it does not contain any variables. Otherwise it is *non-*

ground. For example, *father(john, tom)* is a ground formula, while *father(X, tom)* is not.

We refer to the sources mentioned above for the interpretation of these logical connectiveness and quantifiers.

A *rule* has the form

$$a \leftarrow b_1, b_2, \cdots, b_m, \; m \geq 0. \tag{2.1.1}$$

In (2.1.1), a is an atom and the b_i's are literals. All the variables occurring in the a and b_i are assumed to be universally quantified in front of the formula. The literal a is called the *head* of the rule, or *head literal*, and the $b_i, i = 1, \cdots, m$ are called its *body*, or *body literals*. The "," in the body represents conjunction, as in Prolog.

We use the term *relation*, *predicate*, and *literal* interchangeably to denote the same object in a DDB. We use *relation* in the context of databases, *predicate* in the context of a rule, and *literal* in the context of a clause. Also we use the term *attribute* when we use the term relation and *argument* when we use the term predicate or literal.

A *clause* is a finite disjuction of zero or more literals. A clause form of the rule (2.1.1) can be represented as (2.1.2) below.

$$a \vee \neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_m \tag{2.1.2}$$

Note that all literals in the body of a rule become negative literals in the clause form. A set of clauses is a finite conjunction of clauses. For example, a set of clauses $\{p, q \vee \neg r, \neg q \vee r\}$ is equivalent to $p \wedge (q \vee \neg r) \wedge (\neg q \vee r)$, where p, q, and r are literals. The empty clause has no literals and is always false by its interpretation. It is

represented by □.

A *Horn clause* is a subset of clauses which has at most one positive literal. Horn clauses can be classified into three types and can be interpreted in the database context as follows [Gall78b]:

(1) a single positive literal: for example, **p(a)** is an assertion or a fact stating that **p(a)** is true in the database.

(2) all negative literals: for example, $\neg p(X) \vee \neg q(X)$ is a goal or a query clause in databases; this clause is often represented as $\leftarrow p(X), q(X)$, where the , represents conjunction.

(3) a positive literal and many negative literals: this clause is called a *definite Horn clause*; for example, a clause $p(X) \vee \neg q(X) \vee \neg r(X)$ is equivalent to the deductive rule $p(X) \leftarrow q(X), r(X)$.

A *unit clause* is a rule with an empty body. A *fact* is a ground unit clause.

2.2. Resolution

This section gives a brief overview of resolution theorem proving to make this dissertation self-contained. Comprehensive treatments of resolution theorem proving can be found in [Chan73, Love78, Gene87].

Resolution is an automatic theorem proving method which was first proposed by Robinson [Robi65]. Resolution is considered a major breakthrough in using computers for theorem proving. Resolution theorem proving is a refutation proof procedure. That is, in order to prove that a formula **F** (theorem) follows from a set of axioms **T**, one proves that the formulas $T \wedge \neg F$ are unsatisfiable.

Resolution also requires all formulas to be in the clause form discussed in Section 2.1. Thus, before resolution is performed, all formulas are converted to clause form. Any first-order logic formulas can be converted to the corresponding clause form. This transformation preserves the satisfiability of the original formula[Chan73]. Resolution also requires a variable substitution process called unification, which enables us to extract values for the variables contained in the formula being proved.

Thus, to perform resolution, axioms and the negation of the theorem being proved are converted into a set of clause form. Then, *resolvents* are computed until either the empty clause is derived or no more resolvents can be derived. If the empty clause is derived as a resolvent, then the theorem is proved.

For query processing using resolution in the context of a database, a query formula Q is negated, converted into clause form, and unioned to a set of database axioms. Then, resolvents are computed. If the empty clause is derived as a resolvent, then the query is proved. Answers to the query are extracted via unifications through the resolution process.

In the following, the conversion of formulas to clause form and the resolution principle are discussed. Since converting formulas into clause form requires the formula to be in *conjunctive normal form*, which again requires *prenex normal form*, they are first defined [Chan73].

Definition 2.2.1:

A first-order logic formula F is in *prenex normal form (PNF)* iff it is in the form of $(Q_1X_1) \cdots (Q_mX_m)M$, where Q_i is either $\forall$ or $\exists$ for $i = 1, \cdots, m$, and M is a

formula that does not include $\forall$ or $\exists$. $(Q_1X_1) \cdots (Q_mX_m)$ is called the *prefix* and M is called the *matrix* of F.

Definition 2.2.2:

A formula F is in a *conjunctive normal form (CNF)* iff F has the form of $A_1 \wedge \cdots \wedge A_m$, $m \geq 1$, where A_i is a disjunction of literals, for $i = 1, \cdots, m$.

In order to convert formulas in PNF to clause form, all the existential quantifiers are eliminated by introducing *Skolem functions* or *Skolem constants* [Chan73].

Definition 2.2.3:

Let a formula F be in a PNF in the form of $(Q_1X_1) \cdots (Q_rX_r) \cdots (Q_mX_m)M$. If all $Q_i, i = 1, \cdots, r$, are $\forall$ and $Q_j, j = r + 1, \cdots, m$, are $\exists$, then all variables $X_{r+1} \cdots X_m$ in M can be replaced by a new r-ary function $f(X_1, \cdots, X_r)$ different from other function symbols. These new functions are called *Skolem functions*. Especially, if $r = 0$, then $f(X_1, \cdots, X_r)$ becomes a constant called a *Skolem constant* and denoted as x0.

This transformation is used to eliminate existential quantifiers without affecting the consistency of a formula. For example, suppose we have a formula (2.3.1) below:

$$\exists X \, \forall Y \, \{p(Y) \wedge [q(X) \wedge q(b)] \wedge \neg q(Y)\} \qquad (2.3.1)$$

Formula (2.3.1) is in PNF and in CNF. Eliminating existential quantifiers results in formula (2.3.2) below, with a Skolem constant x0.

$$\forall Y \, \{p(Y) \wedge [q(x0) \wedge q(b)] \wedge \neg q(Y)\} \qquad (2.3.2)$$

Since the formula (2.3.2) is already in CNF, it results in a set of three clauses as

follows:

$$\{p(Y), q(x0) \wedge q(b), \neg q(Y)\}$$

Having discussed the transformation of the formulas into clause form, substitution and unifier are discussed next.

Definition 2.2.4:

A *substitution* θ is a finite set $\{t_1/V_1, \cdots, t_n/V_n\}$, where each V_i is a variable, each t_i is a term different from V_i, the V_i's are distinct from each other, and t_i is substituted for V_i.

Definition 2.2.5:

A substitution θ is called a *unifer* for a set $\{C_1, \cdots, C_k\}$ iff $C_1\theta = C_2\theta = \cdots C_k\theta$. The set $\{C_1, \cdots, C_k\}$ is said to be unifiable if there is a unifier for it.

Definition 2.2.6:

A unifier σ for a set $\{E_1, \cdots, E_k\}$ is a *most general unifier (MGU)* iff, for each unifier θ for the set, there is a substitution λ such that $\theta = \sigma\lambda$.

Intuitively, a unifier is a substitution of terms for variables that makes them identical in a set of clauses. The process of finding such a substitution is called *unification*. A unifier is most general if it makes as few substitutions as possible.

Having discussed necessary definitions, the resolution principle can be defined.

Resolution Principle:

Let $C_1 = p_1 \vee C_1'$ and $C_2 = \neg p_1 \vee C_2'$ be two clauses in a set with no variables in common. Let p_1 and $\neg p_1$ have MGU θ. Then, we can derive a new clause

$\theta(C_1') \vee \theta(C_2')$ and it is called a *resolvent*.

That is, if a literal **p** from a clause C_1 can be unified with the literal $\neg$**p** from a clause C_2, then applying MGU to the remaining literals of C_1 and C_2 and combining them gives a resolvent.

For example, two clauses **p(a, X) ∨ q(Z, c)** and **¬p(Y, b) ∨ r(X, Y)** derive a resolvent **q(Z, c) ∨ r(b, a)** by making use of most general unifier **{a/Y, b/X}** that substitutes **a** for **Y** and **b** for **X**.

This procedure can be seen as follows: first, apply the unifier to clauses C_1 and C_2; then, remove the complementary literals and combine the remaining literals. For example, applying the unifier to the above two clauses gives: **p(a, b) ∨ q(Z, c)** and **¬p(a, b) ∨ r(b, a)**. Removing two complementary literals from these and combining the remaining literals give **q(Z, c) ∨ r(b, a)**.

The resolution procedure in first-order logic is a *semi-decidable* procedure in that it stops when the theorem being proved is derivable under the axioms, but it may or may not stop when the theorem is not derivable. However, the resolution procedure is one of the most powerful and efficient theorem proving procedures because it has important properties of *refutation complete* and *sound* and uses only one inference rule called the resolution principle. By refutation complete we mean that when a refutation strategy is used to prove a theorem, if the theorem being proved is derivable from the axioms, then resolution can prove it (the empty clause is derived). By soundness we mean that if the resolution procedure proves the theorem, then the theorem can be derivable from the axioms.

2.3. Deductive Database (DDB) Systems

A deductive database (DDB) system is a database system in which new facts are derived from the facts that were explicitly stored using an inference rule.

Research on deductive databases has been evolved from the logic database area which applies first-order logic to databases. Logic has mainly contributed to databases as an inference system and as a representation language. A comprehensive survey in this emerging area can be found in [Gall78b, Reit83, Gall84, Brod84, Smit84, Kers85, Brod86, Park86].

A DDB can be either *definite* or *indefinite* depending on the types of rules supported [Gall84]. In definite DDBs, rules supported are limited to *definite clauses*. For example, a definite clause is represented as follows:

$$p(X, Y) \leftarrow q(X, Z), r(Z, W), s(W, Y).$$

That is, the head of the rule (left hand side) is limited to one and only one predicate in the definite Horn clauses.

In indefinite DDBs, not only the head of a rule but also ground clauses (a clause without any variable) can have any number of literals. For example, the following is an example of an indefinite ground clause saying that *John's sibling is either Mary or Tom, but we don't know exactly whom* .

$$sibling(John, Mary) \vee sibling(John, Tom)$$

In this work, we will limit our research to the definite deductive databases. Henceforth, the term deductive database refers to the definite deductive database unless otherwise specified.

A DDB consists of an *extensional database (EDB), an intensional database (IDB), integrity constraints (IC),* and a meta-rule called *negation as failure (NAF).*

An EDB consists of a set of facts or tuples explicitly stored in physical databases. Syntactically, a fact can be represented by a ground unit clause. The literals defined in an EDB are called *extensional literals.*

An IDB consists of a set of deductive rules which can be used to derive new facts using the facts in an EDB. These set of deductive rules are the general laws in databases and can be used in the queries, in the definitions of views, and in integrity constraints. Syntactically, they can be represented by definite Horn clauses having one and only one positive literal and one or more negative literals. In this work, we assume that an IDB consists only of non-recursive rules. The literals defined in an IDB are called *intensional literals. Non-intensional literals* are extensional literals or comparison literals.

Integrity constraints(ICs) are a set of rules which are used to maintain the consistency of databases. Syntactically, they can be represented by any set of closed well formed formulas.

The NAF is a meta-rule for the negative information. It allows us to assume that any facts which do not exist in the database are false. With this rule we do not need to store the negative information in databases.

The main difference between conventional databases and DDBs is that the general semantic laws are represented as ICs in conventional databases, while they can be either deductive rules or ICs in DDBs. Thus, in a DDB, not only are queries evaluated against the EDB and IDB, but also ICs are checked against the EDB and

IDB. Integrity checking in DDB systems is therefore more difficult than that in conventional database systems. However, since we are interested in the deductive aspects of DDBs, we will not discuss the issues of ICs in this work.

2.4. Classification of Query Types in DDB

Queries in DDBs can be classified by many criteria. [Song87] provides an extensive classification of the various query types available in DDBs. That classification is based on the following criteria:

(1) the types of answers (fact or formula),

(2) the free/bound information given in a query,

(3) the preferred reasoning schemes to process a query,

(4) the types of predicates used in a query, and

(5) the types of rules in the IDB.

We neither claim that these are all the possible criteria for classification, nor that they are mutually exclusive.

The purpose of this classification is to increase our insight into the various query types in DDBs by understanding the relationships among them. However, the work considering all of the factors related to the classification would be enormous. For example, in their respective series of papers, Ullman [Ullm85, Ullm86, Banc86], Lozinskii [Lozi84, Lozi85, Lozi86], and Zaniolo [Zani86, Sacc86] have been carrying out research for several years in the area of utilizing bounded arguments in a query. In this dissertation, we are only concerned with a query type whose answers are first-

order logic formulas.

We believe that the best query processing strategy can be determined by considering various criteria like those listed above, together with the conventional query optimization techniques. Thus, we believe that a DDB must have several query processing strategies and it must be able to determine the best processing strategy depending on the type of a given query. Studying this classification and relationships among the query types will, therefore, help us decide the best query processing strategy for a given query. Appendix A further elaborates on each of these types.

2.5. Previous Work

This section gives a brief review of previous work in the area of intensional query processing. The idea of having a set of formula as an answer set has been studied by Cholvy and Demolombe [Chol86] and Imielinski [Imie87]. Cholvy and Demolombe study the notion of having a set of formulas as an answer set, while Imielinski studies the idea of incorporating intensional predicates as a part of answers. Cholvy and Demolombe use resolution to generate answer formulas for a given query in logical formulas, while Imielinski uses relational algebra to transform rules and to process a given query in relational algebra expressions.

2.5.1. Cholvy and Demolombe's Work

This section reviews Cholvy and Demolombe's approach and makes a summary of their problems.

The database theory T is defined as a set of facts and rules. A query is represented by Q(X), where X is a tuple of free variables. The intensional answers to a query Q(X), denoted as $ans_I(X)$, are a set of formulas defined by (2.5.1) below:

$$T \vdash \forall X(ans_I(X) \rightarrow Q(X)) \tag{2.5.1}$$

In (2.5.1) a theorem $\forall X(ans_I(X) \rightarrow Q(X))$ is derived from the database theory T. The literal $ans_I(X)$ was defined such that, under the theory T, any element X, where $X \in ans_I(X)$, must satisfy the Q(X).

Thus, an intensional answer set to the query Q(X) is defined by

$$ANS(Q) = \{ans_I(X): T \vdash \forall X(ans_I(X) \rightarrow Q(X))\} \tag{2.5.2}$$

In fact, an answer formula $ans_I(X)$ can be *any* formula in a logical sense, regardless of its truth value. Thus, it is necessary to impose some restrictions on $ans_I(X)$ to make it a meaningful answer. Some basic ideas considered by Cholvy and Demolombe were limiting $ans_I(X)$ to the database predicates, removing contradictory formulas and removing redundant answers. Thus, the predicates in an answer set is restricted to predicates in T. Let $DP = \{P_1, \cdots, P_n\}$ be a set of predicate symbols either in IDB or EDB. Also Let L(DP) be a first order language whose predicate symbols are in $P_1, \cdots, P_n$.

Thus, an intensional answer set ANS(Q, DP) to a query Q(X), is defined by:

$$\begin{aligned} ANS(Q, DP) = \{&ans_I(X): ans_I(X) \in L(DP) \text{ and} \\ &T \vdash \forall X(ans_I(X) \rightarrow Q(X)) \text{ and} \\ &(ans_I(X) \text{ is not the negation of tautology}) \text{ and} \\ &(\text{each } ans_I(X) \text{ is not redundant})\} \end{aligned} \tag{2.5.3}$$

The last restriction in (2.5.3) removes redundant answers. However, the methodologies and the procedures to simplify the answers were not discussed.

The basic idea of an evaluation strategy is to use the resolution from both the database theory T and the negated theorem given in (2.5.1). For the evaluation of a query, they convert all facts into new rules before resolution. They use the term rulebase to represent these new rules and rules in an IDB. To apply the resolution principle, the theorem is first negated and transformed into clause form as follows:

$$\neg\forall X(ans_1(X) \rightarrow Q(X))$$
$$\equiv \exists X(ans_1(X) \wedge \neg Q(X))$$

Then the clause form of negated theorem is:

$$\{ans_1(x0), \neg Q(x0)\}$$

x0 is a tuple of Skolem constants, which were introduced to remove the existential quantifier. Suppose S is a set of clauses transformed from the theory T. Then, a set of clauses for resolution which derives the empty clause is given as (2.5.4).

$$S \cup \{ans_1(x0), \neg Q(x0)\} \qquad (2.5.4)$$

However, initially we do not know what $ans_1(x0)$ is. Thus, an initial set of clauses that we can start resolution is given by:

$$S \cup \{\neg Q(x0)\} \qquad (2.5.5)$$

Thus, resolving $S \cup \{\neg Q(x0)\}$ without a clause $ans_1(x0)$ will result in a resolvent, say $R(x0)$. Resolving $R(x0)$ together with $ans_1(x0)$ therefore will result in the empty clause.

Since resolution between **R(x0)** and **¬R(x0)** can derive the empty clause, a simple way to compute $\mathbf{ans_1(x0)}$ is to negate the **R(x0)**. Hence, we can have:

$$\mathbf{ans_1(x0) = \neg R(x0)} \qquad (2.5.6)$$

Thus, an intensional answer is a formula corresponding to **¬R(x0)**. Cholvy and Demolombe derives an intensional answer for every resolvent. After negation, a Skolem constant in the **R(x0)** is replaced by a free variable and a variable in the **R(x0)** becomes an existentially quantified variable in an intensional answer.

Summary of Problems

From our review, we note the following important problems to be solved in this area.

(a) No specific resolution strategy was used. Without a specific strategy for resolution, a lot of redundant resolvents can be derived.

(b) No specific methodologies and procedures were given to remove the redundant resolution steps or meaningless answers in an answer set.

(c) No solution was given to handle a large number of facts in EDB. They converted all the facts to the rules. However, this strategy is neither efficient nor practical when we have a large set of facts in EDB.

(d) No solution was given to handle rules whose heads are comparison literals. Since comparison literals do not have their own database-related semantics, the blind application of resolution with this type of rules causes resolution among non-interesting predicates or different domains, and thus derives meaningless intensional answers.

(e) No relationships between extensional answers and intensional answers were studied. If we consider the coupling of intensional query processing with conventional database systems, we must understand the relationships between them.

(f) No relationships between the types of rules and intensional queries were studied. Some types of rules prevent the derivation of intensional answers even though intensional answers exist. Rules should be properly transformed in this case.

(g) Cholvy and Demolombe convert all intermediate resolvents to intensional answers. However, it can be shown that the last resolvent is logically implied by both an intermediate resolvent and an intensional database. Hence, we argue that we do not need to convert all the intermediate resolvents to intensional answers.

Our Approaches to the Problems

For problem (a) above, we provide SLD-RC resolution, which is based on the notions of relevant literals and relevant clauses. The SLD-RC resolution is discussed in Chapter 6. For problem (b), we study how we can reduce the initial set of clauses and what operations we need to perform to each resolvent, etc. These procedures are called the formalization process of intensional answers and are discussed in Chapter 5. For problem (c), we propose a modified compiled approach to solve this problem, which is discussed in Section 5.6. For problem (d), the notion of relevant literals and relevant clauses are introduced in Section 5.3. These two notions are built into the

SLD-RC resolution. Problem (e) is discussed in Chapter 7. We identify a class of rules called term-restricted rules and extended term-restricted rules, and they are discussed in Section 5.2. Problem (g) is discussed in Section 5.5 and taking only the last resolvent from the sequence of resolution is justified.

2.5.2. Imielinski's Work

When answers can be expressed in terms of existing rules, intensional answers can be generated as a part of the answers. Imielinski [Imie87] explores this idea, as shown by the following example taken from [Imie87].

Example 2.5.1:

Suppose a database has relations **teach(X, Y)** and **group(X, Y)**. **Teach(X, Y)** means that *a professor X can teach the course Y*, and **group(X, Y)** means that *professors X and Y are in the same research group*. In this case, we can assume that the professors in the same group can teach the same courses. This can be represented by the following rule:

teach(Z, Y) ← teach(X, Y), group(X, Z).

In this situation suppose we have a query Q(X) = *Who can teach 4402?*, where Q(X) represents a query predicate. The answer to this query could involve the rule above with some professors' names as follows:

Q(Y) ← Q(X), group(X, Y).

This means that *if X is an answer and Y is in the same group as that of X, then Y is also an answer*. That is *if X can teach the course 4402 and Y is in the same group*

as that of X, then Y can also teach 4402. Imielinski argues that a user may be satisfied with this answer if he/she knows who belongs to the same group as that of *X* or *X* is the *right* person about whom one wants to know. On the other hand, a user can further ask to evaluate the rule in the answer to see who else is in the same group with *X* for the full extensional answers.

In order to generate answers like this case, not only must the database contain such rules, which can be embedded in answers, but a query processor must also be able to recognize such rules.

CHAPTER 3

EXTENSIONAL ANSWERS, INTENSIONAL ANSWERS, AND INTENSIONAL QUERIES

In this chapter, we first study the form of queries that appear in the deductive database literature We then formally define extensional answers, intensional answers, and intensional queries.

3.1. Form of Queries

A query can be formulated as a closed formula or an open formula. When a query is formulated as a closed formula, the answer will be *yes/no* depending on whether the query formula can be provable under the current database axioms. On the other hand, when the query is formulated as an open formula, the (extensional) answer will be the instantiated values of the free variables included in the query formula. In this case, the instantiated values satisfy the condition given in the query formula under the current database.

Example 3.1.1:

Suppose our database theory consists of two rules and a fact as follows:

at(mary, X) ← at(john, X)
at(john, X) ← at(bob, X)
at(bob, school)

The first rule says that *Mary goes wherever John does*, and the second rule says that *John goes wherever Bob does*. The fact says that *Bob is at school*. The clause form of

the rules and the fact are:

$\neg$at(john, X) $\vee$ at(mary, X)
$\neg$at(bob, Z) $\vee$ at(john, Z)
at(bob, school)

Note that variable **X** in the second rule has been renamed to **Z** to make variable names unique in the clause set.

In this example, a possible form of a closed query is

$\exists$X at(mary, X)? **: Is there a place X where Mary is at?**

or

at(mary, school)? **: Is Mary at school?**

The answer for the first query could be *Yes, Mary is at school,* and the second query *yes*.

On the other hand, a possible open query is:

at(mary, X)? **: Where is Mary?**

The answer for the above query could be <*X = school*> in this case.

Let us look at the general forms of the queries represented in logical formulas. A goal or query is a conjunction of literals of the form:

$$\leftarrow Q_1, \cdots, Q_n \qquad (3.1.1)$$

That is, a goal clause is a rule which has a body, but is without a head. In a closed query, any free variables appearing in the query clause are assumed to be existentially quantified and written as follows:

$$\leftarrow (\exists X_1 \cdots \exists X_m)\, Q_1, \cdots, Q_n \qquad (3.1.2)$$

This can be read as "are there $X_1, \cdots, X_m$ such that $Q_1, \cdots, Q_n$ is true?" We abbreviate the query clause above as $\exists X\ Q(X)$ in this case, where X is a tuple of free variables appearing in the query clause. Thus the evaluation of a $\exists X\ Q(X)$ query can be viewed as a process of selecting X from a database such that X makes Q(X) true. This formula can also be written equivalently in the clause form as follows:

$$\forall X_1 \cdots \forall X_m (\neg Q_1 \vee \cdots \vee \neg Q_n) \tag{3.1.3}$$

This form of queries is seen in [Clar78, Nico78, Lloy84, Lloy85, Naqv86].

On the other hand, an open query can simply be represented as follows:

$$\leftarrow Q(X) \tag{3.1.4}$$

where X is a tuple of free variables appearing in the query formula. This can be read as *what is X which makes Q(X) true?* This form of queries is seen in [Kowa79, Smit84, Chol86].

The distinction between the two forms of queries, however, is not difficult to understand. In fact, a query with existential quantifiers is an instance of the query form with free variables. The rule of *existential instantiation* [Ende72] allows us to infer $\exists X\ Q(X)$ from Q(X). Thus, in our work, we will generally regard the variables appearing in a query formula to be free, unless otherwise specified.

3.2. Extensional Answers and Green's Method

If a query is formulated as an open formula, the answering process could be regarded as a process of finding bindings for the free variables in the query. Let T be *a database theory* that consists of an EDB and an IDB. In this case, the query formula

obtained after substituting the bindings into the query is a logical consequence of the database theory T.

Let Q(X) be a query formula with a tuple of free variables X. Then by the argument made above, we want to find Xs that satisfy the following relationship.

$$\{X:\ T \vdash Q(X)\} \tag{3.2.1}$$

From (3.2.1), if Q(X) with its bindings of X is indeed a logical consequence of T, then resolution must be able to derive the empty clause from the conjunction of T and $\neg Q(X)$ as follows:

$$(T \wedge \neg Q(X)) \vdash \square$$

Here $\square$ means the empty clause. Note that the empty clause is unsatisfiable. Let S be the clause form of T. Then we have:

$$S \cup \{\neg Q(X)\} \vdash \square \tag{3.2.2}$$

Using resolution, Green [Gree69] proposed a way of extracting extensional answers for a query from a set of axioms. His method is to add a special literal called an *answer literal* to the theorem being proved. This literal contains the same variables as in the theorem(i.e., a query in the context of databases). The purpose of this literal is to collect all substitutions made by the resolution process. Let $ans_E(X)$ be the special answer literal defined by Green, where X is a tuple of free variables appearing in the query. Then a negated query augmented with the answer literal is $\neg Q(X) \vee ans_E(X)$. After a resolution proof is performed on the clauses of both $\neg Q(X) \vee ans_E(X)$ and the database axioms, the instantiated values of the variables in $ans_E(X)$ are the answers for the variables of the query.

In the following, we will define extensional answers based on Green's idea. A negated query augmented with the answer literal is:

$$\neg Q(X) \vee ans_E(X)$$

Note that this clause is equivalent to the formula below.

$$\forall X\ (Q(X) \rightarrow ans_E(X)) \qquad (3.2.3)$$

Thus, if we define an answer to a query Q(X) as a Green's answer literal $ans_E(X)$ and use resolution to derive the answer, then we can use the formula (3.2.3) to define extensional answers as follows:

Definition 3.2.1:

Let T be a database theory and Q(X) be a query, where X is a tuple of free variables appearing in the query. An *extensional answer* is a tuple X which makes Q(X) true in the database theory T. This relationship can be represented by the following axiom.

$$T \wedge \forall X\ (Q(X) \rightarrow ans_E(X)) \qquad (3.2.4)$$

We note that this axiom, which comes from the notion of Green's answer literal $ans_E(X)$, is widely accepted as the definition of answers [Gree69, Luck71, Nils80, Gene87, Maie88].

A set $ANS_E(Q)$ of extensional answers to a query Q(X) can be defined as (3.2.5).

$$ANS_E(Q) = \{X \mid T \wedge [\forall X\ (Q(X) \rightarrow ans_E(X))]\} \qquad (3.2.5)$$

To use resolution to derive an $ans_E(X)$, the clause form of (3.2.4) can be directly

used. This clause form may be used because, for the definition of an extensional answer, we chose Green's literal which is augmented to the negated query **Q(X)**. Note that the clause form of a database theory **T** is **S** and the clause form of $\forall X\,(Q(X) \rightarrow ans_E(X))$ is $\neg Q(X) \vee ans_E(X)$. Thus a set of axioms for the resolution derivation becomes:

$$S \cup \{\neg Q(X) \vee ans_E(X)\} \tag{3.2.6}$$

In this case, we are not deriving the empty clause. Instead we try to derive a clause consisting only of an answer literal $ans_E(X)$ as can be seen from the following proposition.

Proposition 3.2.1:

Let **T** be a database theory and **S** be the set of clauses corresponding to **T**. Let **Q(X)** and $ans_E(X)$ be defined as in (3.2.4). Then we have the following relationship:

$$(S \cup \{\neg Q(X) \vee ans_E(X)\}) \vdash ans_E(X) \tag{3.2.7}$$

Proof:

From (3.2.6), we have:

$$[S \cup \{\neg Q(X) \vee ans_E(X)\}] \vdash [(S \cup \{\neg Q(X)\}) \vee (S \cup \{ans_E(X)\})]$$

Since $S \cup \{\neg Q(X)\} \vdash \square$ (from 3.2.2), we have:

$$[(S \cup \{\neg Q(X)\}) \vee (S \cup \{ans_E(X)\})] \vdash [S \cup \{ans_E(X)\}] \vdash ans_E(X)$$

■

Example 3.2.2:

We use Example 3.1.1 to illustrate the idea presented above. We repeat the rules and fact in Example 3.1.1 for convenience.

at(mary, X) ← at(john, X)
at(john, X) ← at(bob, X)
at(bob, school)

Thus the database theory T for this example is:

[at(mary, X) ← at(john, X)] ∧ [at(john, X) ← at(bob, X)] ∧ [at(bob, school)]

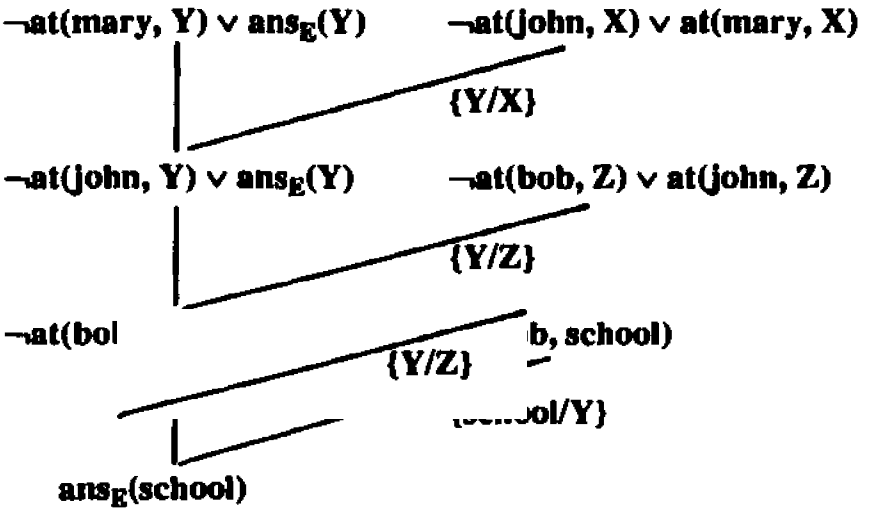
And the set S of clauses corresponding to T is:

¬at(john, X) ∨ at(mary, X)
¬at(bob, X) ∨ at(john, X)
at(bob, school)

Let the query Q(X) be **at(mary, X)?** asking *where is Mary?*. According to the statement (3.2.7), the answer literal $\mathbf{ans_g(X)}$ is disjuncted to the negation of the query **¬(mary, X)**, producing $\mathbf{\neg at(mary, X) \vee ans_g(X)}$. Thus the set of clauses for the resolution proof is as follows:

(s1) $\mathbf{\neg}$ **at(john, X) ∨ at(mary, X)**
(s2) $\mathbf{\neg}$ **at(bob, Z) ∨ at(john, Z)**
(s3) **at(bob, school)**
(Q′) $\mathbf{\neg}$ **at(mary, Y) ∨** $\mathbf{ans_g(Y)}$

Note that the variables were renamed so that they are unique in a clause set. The resolution proof will look as follows:



As it is shown from Proposition 3.2.1, the last resolvent from the resolution proof in this example consists of an answer literal, $\mathbf{ans_E(school)}$, where $\mathbf{X = school}$ is an extensional answer.

We note that extensional answers can be computed either by Green's answer literal (Definition 3.2.1) or by any other method.

3.3. Intensional Answers

In the previous section, we have illustrated Green's method to extract extensional answers using resolution. In this section, we define intensional answers and also illustrate the relationship between extensional answer $\mathbf{ans_E(X)}$ and the intensional answer $\mathbf{ans_I(X)}$. This relationship is further discussed in Chapter 7. We adopt the definition of an intensional answer given in Cholvy and Demolombe [Chol86].

Definition 3.3.1:

Let T be a database theory and Q(X) be a query, where X is a tuple of free variables appearing in a query Q(X). An intensional answer is a non-ground first-order logic formula such that whenever X satisfies the $\mathbf{ans_I(X)}$, it satisfies the Q(X) under a set of current database axioms in T. The relationship among T, $\mathbf{ans_I(X)}$, and Q(X) can be represented as follows:

$$\mathbf{T \vdash \forall X(ans_I(X) \rightarrow Q(X))} \tag{3.3.1}$$

In this definition, the formula $\mathbf{\forall X(ans_I(X) \rightarrow Q(X))}$ is the theorem being proved under the database theory T. A set $\mathbf{ANS_I(X)}$ of intensional answers to a query Q(X) is defined as follows:

$$\mathbf{ANS_I(Q) = \{ans_I(X): T \vdash \forall X(ans_I(X) \rightarrow Q(X))\}} \tag{3.3.2}$$

In a logical sense, an intensional answer $\mathbf{ans_I(X)}$ in its definition can be any formula regardless of its truth value. Thus we need to impose some restrictions on $\mathbf{ans_I(X)}$ to make it an acceptable answer. We will call such conditions imposed on $\mathbf{ans_I(X)}$ "restrictions on $\mathbf{ans_I(X)}$." These restrictions and the formalization of these restrictions are discussed in Chapter 5. A specific resolution method to generate intensional answers incorporating these restrictions is discussed in Chapter 6.

In Section 2.5.1, the procedures of deriving intensional answers have been reviewed. The initial set of clauses for resolution to derive intensional answers has been given as (2.5.5).

$$\mathbf{S \cup \neg Q(x0)} \tag{2.5.5}$$

S is a set of clauses corresponding to a database theory T and x0 is a Skolem constant.

Finally, an intensional answer is taken as the negation of a resolvent as in (2.5.6).

$$\mathbf{ans_I(x0) = \neg R(x0)} \qquad (2.5.6)$$

After negation, Skolem constants in **R(x0)** are replaced by free variables, and variables in **R(x0)** are existentially quantified in intensional answers [Chol86]. Here, we illustrate the idea of removing the Skolem constants in **R(x0)**.

Mainly **R(x0)** can take either of the following forms.

(a) *When* **R(x0)** *consists only of* **x0:**

In this case, a Skolem constant in **R(x0)** is instantiated to a free variable. Suppose $\mathbf{R(x0) = \neg p(x0)}$, where **p** is any predicate name. Note that **x0** is a Skolem constant. A Skolem constant is an arbitrary unknown constant that is distinct from any other constants in T. We do not know what the exact value of **x0** is, but we do know we have **x0** in T. Hence, we can instantiate **x0** to a free variable **X** in a formula. On the other hand, we can consider instantiating **x0** to an existentially quantified variable. However, since we can always infer $\exists$**X p(X)** from **p(X)** by the rule of existential instantiation [Ende72], we prefer the more general form **p(X)** to $\exists$**X p(X)**. Then by (2.5.6) we have:

$$\begin{aligned}\mathbf{ans_I(x0)} &= \mathbf{\neg(\neg p(x0))} \\ &= \mathbf{p(X)}\end{aligned}$$

One might argue that, in order to derive the empty clause from $\neg$**p(x0)**, we might take **p(X)** as an $\mathbf{ans_I(x0)}$. However, since **p(X)** is a clause and every variable in a clause is universally quantified, the formula corresponding to **p(X)** in this case is $\forall$**X p(X)**. However, this choice can be shown to be wrong, because the

meaning of a Skolem constant, which is *an arbitrary unknown constant*, was ignored during the conversion. That is, instead of an unknown constant implied by a Skolem constant, all the constants in T can satisfy p(X) because of the universal quantifier in ∀X p(X). Example 3.3.1 below shows this idea.

(b) *When* R(x0) *consists of* x0 *and other variable:*

In this case, the variable in resolvent is instantiated to an existentially quantified variable. Suppose R(x0) = ¬p(x0, Y). Then by (2.5.6), we have:

$$\begin{aligned} ans_1(x0) &= \neg R(x0) \\ &= \neg(\neg p(x0, Y)) \end{aligned}$$

Since every variable in a clause is universally quantified and, in addition, negation operation changes the quantifiers of variables in a formula, we replace the clause by a quantified formula. Hence, we have:

$$\begin{aligned} &= \neg(\forall Y \neg p(x0, Y)) \\ &= \exists Y\, p(X, Y) \end{aligned}$$

Example 3.3.1:

Suppose we have a rule as follows:

$$gf(X, Y) \leftarrow f(X, Z), f(Z, Y)$$

Here, gf(X, Y) means *X is the grandfather of Y* and f(X, Z) means *X is the father of Z*. Thus the rule means *X is the grandfather of Y if X is the father of Z and Z is the father of Y*. If we have a query gf(X, Y)? *asking who is the grandfather of whom ?*, then the set of clauses for resolution in this case by (2.5.5) is :

$$\{gf(X, Y) \vee \neg f(X, Z) \vee \neg f(Z, Y), \neg gf(x0, y0)\}$$

The resolution between the two clauses will derive the resolvent as follows:

$$\neg f(x0, Z) \vee \neg f(Z, y0)$$

Here, even though there are many options to resolve the above resolvent to the empty clause, we will follow the simplest choice as shown in (2.5.6) by taking the negation of **R(x0)**. Hence, **R(x0)** becomes $\neg f(x0, Z) \vee \neg f(Z, y0)$. Then we have:

$$\begin{aligned} ans_1(x0) &= \neg R(x0) \\ &= \neg(\neg f(x0, Z) \vee \neg f(Z, y0)) \end{aligned}$$

Recall that every variable in a clause is universally quantified. Since the negation operation changes the quantifiers of variables, we replace the clause by a quantified formula. Hence, we have:

$$\begin{aligned} ans_1(x0) &= \neg(\forall Z\ (\neg f(x0, Z) \vee \neg f(Z, y0))) \\ &= \exists Z\ f(x0, Z) \wedge f(Z, y0) \end{aligned}$$

Here, if we instantiate the Skolem constants **x0** and **y0** by universally quantified variable, then we have:

$$ans_1(X) = \exists Z\ \forall X\ \forall Y\ f(X, Z) \wedge f(Z, Y)$$

However, this formula is contradictory since it means *a person Z is everybody's son and everybody's father.*

Hence, when **R(x0)** has a Skolem constant, we instantiate it to a free variable. This result also agrees to our intuitive notion of a Skolem constant which implies an arbitrary constant in **T** that is distinct from others. The correct answer formula is:

$$ans_1(X) = \exists Z\ f(X, Z) \wedge f(Z, Y)$$

In the following, we give an example of intensional answers using the same

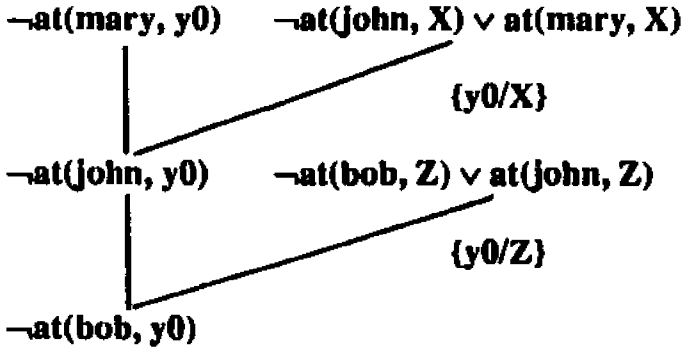
database and query as in Example 3.2.1.

Example 3.3.2:

The query was **at(Mary, Y)?** asking *where is Mary*. By (2.5.5), an initial set of clauses for resolution to derive intensional answers for the query above is:

(s1)	**¬ at(john, X) ∨ at(mary, X)**
(s2)	**¬ at(bob, Z) ∨ at(john, Z)**
(s3)	**at(bob, school)**
(Q(y0))	**¬ at(mary, y0)**

Note that the variables were renamed so that they are unique in a clause set. The resolution proof for the goal **¬at(mary, y0)** will look as follows:



Since we have two resolvents, we have two intensional answers as follows:

$$ans_I^1(Y) = at(john, Y).$$
$$ans_I^2(Y) = at(bob, Y).$$

The answers can be read, according to the definition of an intensional answer given in (3.3.1), as *if John is at Y then Mary is at Y* and *if Bob is at Y then Mary is at Y*. As has been shown in Examples 3.2.2 and 3.3.2, the extensional answer is $ans_E(school)$, thus **<Y = school>**; and the intensional answers are $ans_I^1(Y) = at(john, Y)$ and $ans_I^2(Y) = at(bob, Y)$. The extensional answer **<Y = school>** satisfies the intensional

answer **at(bob, Y)**, since **at(bob, school)** exists in the database. Note that the extensional answer **<Y = school>** also satisfies the intensional answer **at(john, Y)**, since, by the rule "**at(john, X) ← at(bob, X)**", **at(john, school)** is implied by **at(bob, school)**. Thus we can see the extensional answer **<Y = school>** satisfies the intensional answers, both **at(john, Y)** and **at(bob, Y)**, under the current database theory **T**.

3.4. Intensional Queries

We have discussed two types of answers for a given query. Extensional answers can be represented by a set of facts as in conventional database systems, while intensional answers can be represented by a set of non-ground first-order logic formulas. Based on these two types of answers, we define extensional queries and intensional queries.

Definition 3.4.1:

A query **Q(X)** is *extensional* if all the answers to a query **Q(X)** can be represented by extensional answers defined by definition 3.2.1.

Definition 3.4.2:

A query **Q(X)** is *intensional* if answers to a query **Q(X)** can be represented by intensional answers defined by definition 3.3.1.

Example 3.4.1:

In Example 3.2.1, the query **at(mary, X)** is not only an extensional query but also an intensional query since it has both an extensional answer and intensional answers as follows:

$ans_E(Y) = ans_E(school)$

$ans_I^1(Y) = at(john, Y)$

$ans_I^2(Y) = at(bob, Y)$

Some queries are purely extensional, while others could be both extensional and intensional. For example, if a query Q(X) consists only of extensional literals and/or comparison literals, we have no way to compute intensional answers. In the following several examples are given.

Example 3.4.2:

The examples given in Section 1.3 are intensional queries. They are reproduced for convenience.

EDB Schema:
- **(d1): student(Sname, Year)**
- **(d2): course(Cno, Cname, CreditHr)**
- **(d3): has_taken(Sname, Cno)**

IDB:
- **(r1): pre_req(Sname, 4402) ← has_taken(Sname, 3102)**
- **(r2): cannot_take(Sname, Cno) ← student(Sname, 1), Cno = 3102.**
- **(r3): cannot_take(Sname, Cno) ← ¬pre_req(Sname, Cno)**
- **(r4): cannot_take(Sname, Cno) ← student(Sname, 1), Cno > 4000.**

As presented in Section 1.3, for this database the first query is $Q_1(X) = cannot\_take(X, 4402)$, asking *who cannot take 4402 ?* The set of intensional answers for $Q_1(X)$ has been:

$ans_I^1(X) = student(X, 1)$

$ans_I^2(X) = \neg\ has\_taken(X, 3102)$

The interpretations of these intensional answers are:

$ans_I^1(X)$: *if any student is a freshman, then he cannot take 4402.*

$ans_I^2(X)$: *if any student has not taken 3102, then he cannot take 4402.*

The second query is Q_2(Cno) = **cannot_take(david, Cno)**, asking *what courses david cannot take ?* The set of intensioanl answers for Q_2(X) has been:

$$ans_I^1(Cno) = eq(Cno, 3102)$$
$$ans_I^2(Cno) = gt(Cno, 4000)$$

The interpretations are:

ans_I^1(Cno) = *if the course number is equal to 3102, then David cannot take it.*

ans_I^2(Cno) = *If the course number is greater than 4000, then David cannot take it.*

These are two examples of intensional queries. Note that Q_1(X) and Q_2(Cno) are also extensional queries, since their intensional answers can be evaluated against the EDB and generate a set of factual data as an answer set. On the other hand, the queries Q_3(X) and Q_4(X) below are purely extensional queries.

$$Q_3(X) = student(X, Year), has\_taken(X, 4402)$$
$$Q_4(X) = student(john, Year), has\_taken(john, X)$$

The query Q_3(X) asks for *students who have taken 4402*, and Q_4(X) asks for *all courses that a student John has taken so far.*

The characteristics of Q_1(X) and Q_2(X) are that their intensional answers are the conditions to their queries. That is, by intensional answers, we can derive a set of conditions to the given query, which are not available in conventional database systems. The characteristics of Q_3(X) and Q_4(X) are that they consist only of extensional literals.

Example 3.4.3:

Another example for intensional queries are those using a family database. Suppose

a database contains the following rules defining family relationships based on *father*, *mother*, and *sex*.

child(X, Y) ← father(Y, X)
child(X, Y) ← mother(Y, X).
son(X, Y) ← child(X, Y), sex(X, male)
daughter(X, Y) ← child(X, Y), sex(X, female)
siblings(X, Y) ← father(Z, X), father(Z, Y)
siblings(X, Y) ← mother(Z, X), mother(Z, Y)
brother(X, Y) ← siblings(X, Y), not(X = Y), sex(X, male)
sister(X, Y) ← siblings(X, Y), not(X = Y), sex(X, female)
grandfather(X, Y) ← father(X, Z), father(Z, Y)

All these rules describe the definitions of family relationships. Any queries concerning these relationships can be intensionally defined. That is, one definition can be written in terms of other definitions.

CHAPTER 4

SLD RESOLUTION TREES

In this chapter we first discuss the notation and terminology for SLD resolution trees. Then we discuss the motivation for adopting SLD resolution for our problem.

Next we define an SLD resolution tree. Then we modify it for an SLD-EA resolution tree for extensional answers, which use Green's literal, and for an SLD-IA resolution tree for intensional answers, respectively.

We also discuss the problems of applying SLD-IA tree for the derivation of intensional answers. In Chapter 6, we refine an SLD-IA tree to an SLD-RC tree that can avoid deriving meaningless intensional answers.

4.1. Preliminaries

We first define some terminology for the discussion of SLD resolutions in our approach.

A *negative clause* is a rule with a body only. For example,

$$\leftarrow p_1, p_2, \cdots, p_n \tag{4.1.1}$$

is a negative clause. In *clause form* it is equivalent to:

$$\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \tag{4.1.2}$$

We use a negative clause as in (4.1.1) to represent a goal or nodes in resolution trees, which will be discussed in this chapter and later chapters.

An *introduced clause* is a clause which is resolved with either a goal clause or its

resolvent. The literals in the body of an introduced clause are called *introduced literals* in that they are added to a current goal clause, creating a new resolvent. A *selected literal* in a goal clause is a literal which is resolved away. A *selection function* SF is a function which chooses a selected literal among the literals in a goal clause.

Let the goal G be represented by a negative clause $\leftarrow p_1, \cdots, p_m, \cdots, p_k\ (k \geq 1)$ and C be an introduced clause $p \leftarrow q_1, \cdots, q_n\ (n \geq 1)$, with no variables in common between G and C. Let SF be a selection function. Suppose p_m is a selected literal chosen by SF from G, and let p_m and p be unifiable with the most general unifier θ. Then *a resolution derivation* between two clauses G and C infers the *resolvent* Res(G, C) as follows:

$$\mathbf{Res(G, C)} = \leftarrow (p_1, \cdots, p_{m-1}, q_1, \cdots, q_n, p_{m+1}, \cdots, p_k)\,\theta \qquad (4.1.3)$$

Let G be R^0, S be a set of database clauses consisting of {EDB} $\cup$ {IDB}, and $C^i \in S$, where $(i \geq 0)$. Then *a sequence of resolution derivation* derives a sequence of resolvents R^i as follows:

$$\begin{aligned}
&\mathbf{Res}(G, C^0) = R^1\\
&\mathbf{Res}(R^1, C^1) = R^2\\
&\mathbf{Res}(R^2, C^2) = R^3\\
&\quad\vdots\\
&\mathbf{Res}(R^n, C^n) = R^{n+1},
\end{aligned}$$

where *G and* C^0 and R^i *and* C^i are assumed to be unifiable by their most general unifiers, respectively.

4.2. SLD Resolution

SLD resolution stands for *Linear Resolution with Selection Function for Definite Clauses* [Kowa71, Lloy84]. In the SLD resolution, one of the parent clause of a resolvent is always taken from an initial set of clauses. As indicated in Section 4.1, this clause has been named as an introduced clause. Another clause of a resolvent is either a goal clause or a resolvent from the previous resolvent. For these reason, it is also viewed as a *linear input resolution*. (It is *input* in that one clause is always taken from an initial set of clauses and *linear* in that another clause is always either a goal clause or a resolvent.) A selected literal and an introduced clause for resolution depends on a selection function SF and a search strategy, respectively. For example, the Prolog interpreter is based on SLD resolution. Its selection function selects the leftmost literal from a goal clause, and its search function selects a rule in a top-down fashion.

We adopted SLD-like resolutions for intensional query processing for two reasons. First, SLD resolution has been proven to be *complete* and *sound* for any selection function, which means any goal implied by the database can be provable by an SLD resolution and the proof by an SLD resolution is correct regardless of the selection function used [Lloy84]. Second, being goal-oriented, the procedures of SLD resolutions can be easily visualized as a tree form. Thus, a branch of an SLD resolution tree which has a successful leaf defines an answer to a given query. Also an efficient search strategy, such as depth-first search, can be easily implemented, based on this tree form.

4.3. SLD Resolution Tree

We first define an SLD tree. An SLD resolution tree represents particular resolution derivations for a given query clause and database clauses. The root node of an SLD resolution tree is a negated query clause. A set of arcs of an SLD resolution tree is a set of edges drawn between two successive resolvents. A leaf of an SLD resolution tree is dependent on the type of a resolution tree, which is discussed in this chapter. *A branch* of an SLD resolution tree is *a sequence of particular resolution derivations* that lead to a leaf of an SLD resolution tree.

Let Q(X) be a query clause, where X is a tuple of free variables, and let S be a set of database clauses consisting of {EDB} ∪ {IDB}. Then the set of clauses for resolution to process a query Q(X) is:

$$S \cup \{\neg Q(X)\} \tag{4.3.1}$$

Let us define G as a goal clause as follows:

$$G = \{\neg Q(X)\} \tag{4.3.2}$$

The goal clause G can be represented by a negative clause as follows:

$$\leftarrow Q(X) \tag{4.3.3}$$

Then the set of clauses for resolution using a goal clause G can be defined by (4.3.4).

$$S \cup \{G\} \tag{4.3.4}$$

Since Q(X) with its bindings of X is a logical consequence of S, we have the following relationship:

$$S \cup \{G\} \vdash \square, \tag{4.3.5}$$

where □ is the empty clause. Based on the relationship (4.3.5), an SLD resolution is defined as an SLD-tree as follows:

Definition 4.3.1:

Let S be a set of rules, Q(X) be a query formula, and G be a goal clause defined by $\leftarrow Q(X)$. Then the SLD-tree for $S \cup \{G\}$, with a selection function SF, is defined as follows:

(a) Each node of the tree is either a goal clause or its resolvent(possibly empty).

(b) The root node is G.

(c) Let $\leftarrow p_1, \cdots, p_m, \cdots, p_k$ $(k \geq 1)$ be a node in the tree, and suppose that p_m is a selected literal for resolution. Then the node has a descendent for each input clause $p \leftarrow q_1, \cdots, q_n$ $(n \geq 1)$ such that p_m and p are unifiable. The descendent is

$$\leftarrow (p_1, \cdots, p_{m-1}, q_1, \cdots, q_m, p_{m+1}, \cdots, p_k)\,\theta$$

where θ is the most general unifier of p_m and p.

(d) Nodes which represent the empty clause have no descendents.

In this definition, a leaf node with the empty clause is a *success* node, while a leaf node with a non-empty clause which cannot be further resolved is a *failure* node.

One of the advantages of the above definition is that we do not need to convert a set of rules and queries to clause form explicitly in order to perform the resolution. The reason is that we can easily identify the positive literals and the negative literals from a rule. That is, the literal in the head of a rule is a positive literal, while the literals in the body of rules, after we convert the rules into clause form, are negative

literals. The literals in a query clause are all negative literals. Thus if a literal in a query clause can unify with the head literal of a rule, then via unification process the body of the rule is substituted for the literal in the query. Thus informally, from the operational point of view, an SLD resolution process can be viewed as a sequence of derivations of negative clauses, starting from the goal clause, by clause substitution with unification.

4.4. SLD Resolution with Depth-First Search

Prolog employs the SLD resolution which selects the leftmost literal from a goal clause and searches a database with depth-first search strategy. It is recursive rules with depth-first search mechanism that could make Prolog's search *incomplete*. That is, if the depth-first search terminates then it is complete; however, it may not terminate even though other search strategies (e.g. breadth-first search) may terminate [Lloy84]. However, depth-first search is more efficient and easy to implement than other search strategies such as breadth-first search. Also when only one solution is desired, depth-first search is clearly much more efficient. Thus, depth-first search is also employed for the discussion of other SLD resolution trees. Since it was assumed that there will be no recursive rules in IDBs, the depth-first search here will be complete.

In the following, an SLD resolution with depth-first search is illustrated.

Example 4.4.1:

Suppose we have the following rules and facts.

EDB and IDB:

like(mary, X) ← respect(john, X)
respect(john, X) ← smart(X), famous(X)
respect(john, X) ← writer(X)
writer(smith)
smart(bob)
famous(jack)

For a query *like(mary, X)?*, the SLD resolution tree which selects the leftmost intensional literal will look as follows by Definition 4.4.1.

Goal: ← like(mary, X)

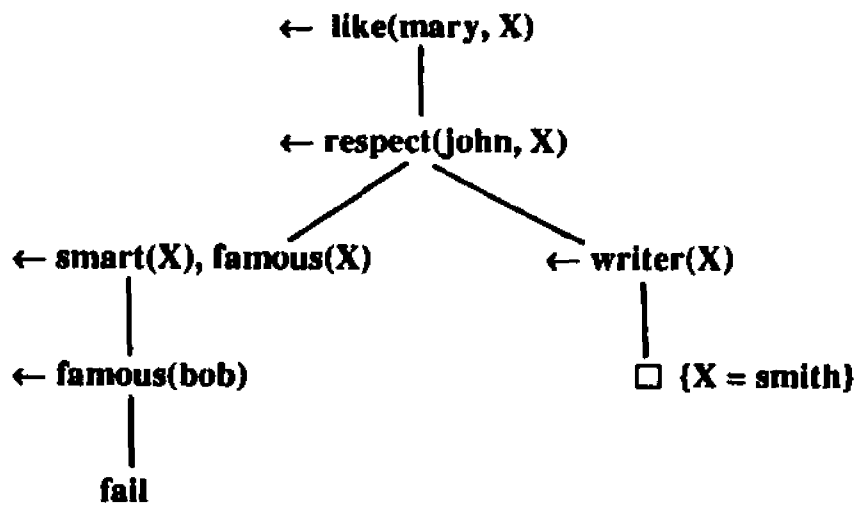


Figure 4.4.1 An SLD tree for a LIKE database

In this example there are two branches in the SLD tree. However, there is only one success branch that ends up with the empty clause. The other branch is a failure branch since it neither has the empty clause nor can be resolved further. Extensional answers are obtained through unification during the search for the refutation. Here the variable X was bounded to an extensional answer smith through unification.

4.5. SLD Resolution Tree for Extensional Answers (SLD-EA Tree)

In this section, we define an SLD-EA tree for deriving extensional answers. The SLD-EA tree is an SLD tree whose goal clause is augmented by the Green's answer literal. Thus the branch of an SLD-EA tree consists of the answer literal, instead of the empty clause as in conventional SLD trees.

As we have discussed in Section 3.2, extensional answers can be obtained by using Green's literals (of course, they can be computed by any other method, too). Then, according to the axiom (3.2.6), a set of clauses for resolution to compute extensional answers are:

$$S \cup \{\neg Q(X) \vee ans_E(X)\} \tag{4.5.1}$$

A goal clause G, in this case, is defined as follows:

$$G = \{\neg Q(X) \vee ans_E(X)\} \tag{4.5.2}$$

The negative clause form of G is:

$$\leftarrow Q(X), \neg ans_E(X) \tag{4.5.3}$$

By Proposition 3.2.1, a success branch of resolution tree from the clause set (4.5.1) must result in a resolvent which consists only of answer literal $ans_E(X)$ with its instantiated value. That is, we have the following relationship:

$$S \cup \{\neg Q(X) \vee ans_E(X)\} \vdash ans_E(X) \tag{4.5.4}$$

Based on the relationship (4.5.4), an SLD-EA tree is defined as follows:

Definition 4.5.1:

Let S be a set of rules, Q(X) be a query formula, and G be a goal clause defined by $\leftarrow Q(X), \neg ans_E(X)$. Then the SLD-EA tree for $S \cup \{G\}$, with a selection function SF,

is defined as follows:

(a) Each node of the tree is either a goal clause or its resolvent.

(b) The root node is G.

(c) Let $\leftarrow p_1, \cdots, p_m, \cdots, p_k$ $(k \geq 1)$ be a node in the tree, and suppose that p_m is a selected literal for resolution. Then the node has a descendent for each input clause $p \leftarrow q_1, \cdots, q_n$ such that p_m and p are unifiable. The descendent is

$$\leftarrow (p_1, \cdots, p_{m-1}, q_1, \cdots, q_n, p_{m+1}, \cdots, p_k)\,\theta$$

where θ is the most general unifier of p_m and p.

(d) Nodes which consist only of the answer literal have no descendents.

In an SLD-EA tree, a branch with a leaf node which consists only of an answer literal is a *success* branch, while a branch with a leaf node which consists of the answer literal and other literals, that cannot be further resolved, is a *failure* branch. We illustrate an SLD-EA tree below.

Example 4.5.1:

The same database and query as in Example 4.4.1 will be used. For a query *like(mary, X)?*, an SLD-EA tree which selects the leftmost intensional literal is shown below:

Goal: $\leftarrow$ like(mary, X), $\neg ans_E(X)$

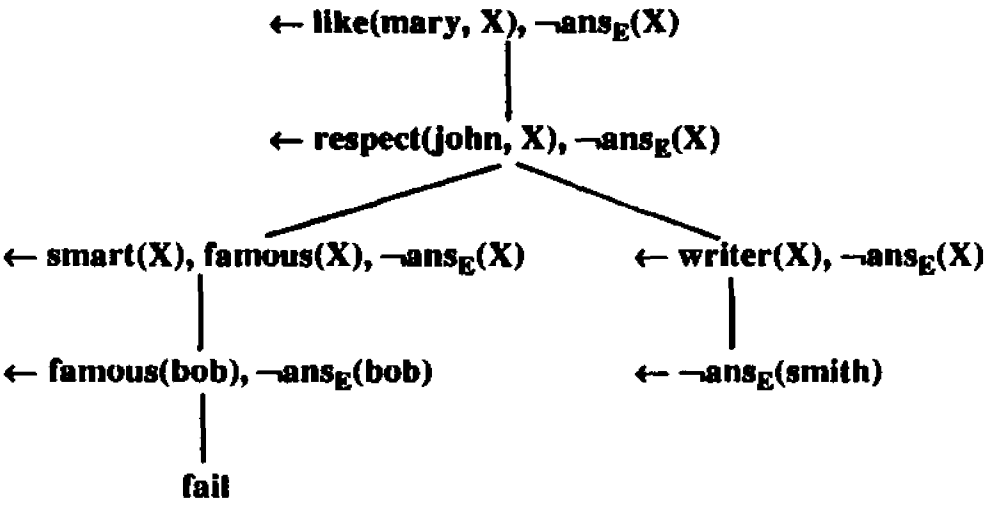


Figure 4.5.1 An SLD-EA tree for a LIKE database

Note that the negative clause $\leftarrow \neg ans_E(smith)$ is equivalent to $ans_E(smith)$, as was described in Section 4.1. There are two branches in this SLD-EA tree. Only the right branch is a success branch since it consists only of the answer literal. The left branch is a failure branch since it contains another literal, except the answer literal, that cannot be further resolved.

As has been shown by Examples 4.4.1 and 4.5.1, the SLD tree and the SLD-EA tree are the same except that a clause in each node is disjuncted by the answer literal $ans_E(X)$. Thus, in an SLD tree answers are implicitly obtained, while they are explicitly captured by the answer literal in each success branch of an SLD-EA tree.

4.6. SLD Resolution Tree for Intensional Answers (SLD-IA Tree)

In the previous sections, an SLD resolution with depth-first search and an SLD-

EA tree were defined and illustrated. In this section, first an SLD tree which derives intensional answers (SLD-IA tree) is discussed. And then, the problems of SLD-IA trees in deriving intensional answers are discussed.

4.6.1. SLD-IA Tree

The intensional answers to a query Q(X), denoted as $ans_I(X)$, has been defined by (3.3.1) as follows:

$$T \vdash \forall X(ans_I(X) \rightarrow Q(X)) \qquad (3.3.1)$$

In (3.3.1) a theorem $\forall X(ans_I(X) \rightarrow Q(X))$ is derived from the database theory T. The literal $ans_I(X)$ has been defined such that, under the theory T, any element X, where $X \in ans_I(X)$, must satisfy the Q(X).

Using resolution to generate intensional answers has been discussed in Section 2.5.1. The basic idea is applying resolution to a set of clauses that consist of both the database theory T and the negated theorem given in (3.3.1). Since SLD-IA resolution is also a resolution method, this basic procedure is also the same when an SLD-IA tree is used for the derivation of intensional answers.

To apply resolution principle, the theorem is first negated and transformed into clause form as follows:

$$\neg\forall X(ans_I(X) \rightarrow Q(X))$$
$$\equiv \exists X(ans_I(X) \wedge \neg Q(X))$$

Then the clause form of negated theorem is:

$$\{ans_I(x0), \neg Q(x0)\}$$

x0 is a tuple of Skolem constants, which is introduced to remove existential quantifier. Suppose **S** is a set of clauses transformed from the theory **T**. Then applying the resolution to

$$S \cup \{ans_I(x0), \neg Q(x0)\} \qquad (4.6.1)$$

should result in the empty clause □. However, initially we do not know what $ans_I(x0)$ are. Thus resolving $S \cup \{\neg Q(x0)\}$ without $ans_I(x0)$ will result in a resolvent, say **R(x0)**. Resolving **R(x0)** together with $ans_I(x0)$ therefore will result in the empty clause.

Thus an initial set of clauses for resolution to derive the intensional answers is given as follows:

$$S \cup \{\neg Q(x0)\} \qquad (4.6.1)$$

where **x0** is a Skolem constant which is introduced to remove an existential quantifier. Thus, in this case, a goal clause **G** in the form of a negative clause is defined below.

$$\leftarrow Q(x0) \qquad (4.6.2)$$

Since a goal clause in this case contains a Skolem constant, it can unify with the variable only, but not with constants. Also we impose one restriction that the resolution will stop if the resolvent consists only of non-intensional (extensional and comparison) literals. That is, we do not resolve the extensional literals or comparison literals. The selection function in an SLD-IA tree will always choose *the leftmost intensional literal*, unless otherwise specified. Based on these relationships, an SLD-IA tree for the clause set (4.6.1) is defined as follows:

Definition 4.6.2:

Let S be a set of rules and Q(X) be a query formula. Let G be a goal clause defined by $\leftarrow Q(x0)$, where x0 is a tuple with Skolem constants. Then the SLD-IA tree of root G, with a selection function SF, is defined as follows:

(a) Each node of the tree is either a goal clause or its resolvent.

(b) The root node is G.

(c) Let $\leftarrow p_1, \cdots, p_m, \cdots, p_k$ $(k \geq 1)$ be a node in the tree and suppose that p_m is a selected literal for resolution. Then the node has a descendent for each input clause $p \leftarrow q_1, \cdots, q_n$ $(n \geq 1)$ such that p_m and p are unifiable. The descendent is

$$\leftarrow (p_1, \cdots, p_{m-1}, q_1, \cdots, q_n, p_{m+1}, \cdots, p_k)\,\theta$$

where θ is the most general unifier of p_m and p.

(d) A node which consists only of non-intensional literals have no descendents. This node is represented as R^n and called *the last resolvent.*

Note that if a comparison literal is used as the head of a rule, then the comparison literal is treated as an intensional literal. This assumption is necessary to define a unique literal in a database theory T.

Definition 4.6.3:

A *success branch* of an SLD-IA tree is a branch with a leaf that consists only of non-intensional literals, which are either extensional literals or comparison literals. A *failure branch* of an SLD-IA tree is a branch with a leaf that contains at least one intensional literal which cannot be further resolved.

From this definition, we can see that the last resolvent is computed from a success branch, but not from a failure branch.

Definition 4.6.4:

A *candidate* for an intensional answer is the formula negated from the last resolvent of a branch of an SLD-IA tree.

Relationships between the last resolvent and a candidate for intensional answer are discussed in Section 5.5 and Section 5.6.

We note that a main difference among the SLD trees, the SLD-EA trees, and the SLD-IA trees lies in the definition of a success branch. The leaf of a success branch of SLD trees are the empty clause, while it is the answer literal in the SLD-EA trees and non-intensional literals in the SLD-IA trees.

Example 4.6.1:

The same database and query as in Example 4.4.1 will be used. The rules and facts are shown again for the convenience of discussion.

EDB and IDB:

like(mary, X) ← respect(john, X)
respect(john, X) ← smart(X), famous(X)
respect(john, X) ← writer(X)
writer(smith)
smart(bob)
famous(jack)

For a query *like(mary, X)?*, an SLD-IA tree which selects the leftmost intensional literal is shown below:

Goal: ←like(mary, x0)

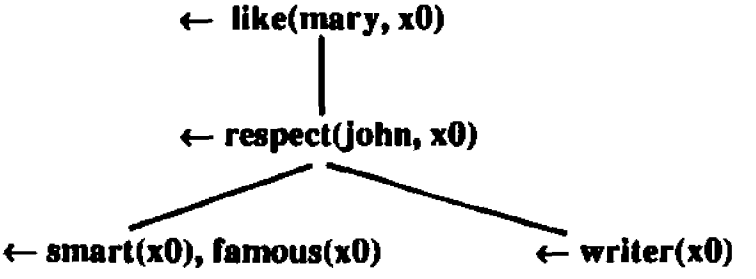


Figure 4.6.1 An SLD-IA tree for a LIKE database

In this tree, the resolution procedures stopped when the resolvents consist only of non-intensional literals. There are also two branches as in Examples 4.4.1 and 4.5.1. However, there is an important difference between this tree and those. Both branches of the SLD-IA tree are successful ones, while only the right branch was a successful one in the SLD tree and the SLD-EA tree. The reason is that intensional answers are independent of the current database state and represent the semantics of rules captured in an intensional database*. That is, an extensional database does not contain the fact **famous(bob)**. Thus, this node was a failure node in the SLD tree and the SLD-EA tree. In the SLD-IA tree, this node is not further resolved since it consists of an extensional literal. As a matter of fact, this clause cannot be further resolved even if the extensional database contains the fact **famous(bob)**. This is because the clause **famous(x0)** contains a Skolem constant **x0** that cannot be unified with any other constant. Note that Figure 4.6.1 has two last resolvents, and thus there are two candidates for intensional answers. Since a candidate for an intensional answer is the negation of $\mathbf{R^n}$, we have:

* This problem is further discussed in Section 5.6.3.

$$ans_I^1(X) = \neg R^n$$
$$= \neg (\leftarrow smart(x0), famous(x0))$$
$$= \neg (\neg smart(x0) \vee \neg famous(x0))$$
$$= smart(X), famous(X)$$

Note that a Skolem constant x0 was instantiated to a free variable X. The second candidate for an intensional answer can be computed similarly.

$$ans_I^1(X) = smart(X), famous(X)$$
$$ans_I^2(X) = writer(X)$$

These candidates are actually intensional answers. The usefulness of candidates is discussed in Section 5.6.

In the next section, we show examples that an SLD-IA tree results in a meaningless intensional answer*.

4.6.2. Examples of Problems

It has been discussed that most Prolog implementation of SLD resolution is not complete because of recursive rules with unbound depth-first search strategy. However, when an IDB does not have recursive rules, SLD resolution in Prolog is complete. Since it has been assumed that there are no recursive rules in an IDB, the SLD resolution with depth-first search can be applied to generate the intensional answers. However, the blind application of SLD resolution (or any other resolution) may result in meaningless intensional answers to a query. These undesirable cases can happen when an IDB has rules whose heads are the same comparison literals and

* The precise definition of a meaningless intensional answer is given in Chapter 5. For the time being, it can be understood as an *incorrect* intensional answer or a *non-acceptable* intensional answer to a given query.

whose bodies have different semantics each other. This point is illustrated below.

Example 4.6.2:

Suppose we have the following EDB schema and IDB about car-dealership. This database is explained in Appendix B.

EDB Schema:

(d1): emp(Name, Salary, Job-type)
(d2): car(Cno, Model, Year, Price)
(d3): sold(Name, Cno)

IDB:

(r1): expensive-car(C1) ← car(C1, M1, Y1, P1), gt(P1, 20)
(r2): economic-car(C2) ← car(C2, M2, Y2, P2), ¬ gt(P2, 5)
(r3): gt(S3, 20) ← emp(N3, S3, manager)
(r4): gt(P3, 20) ← car(C3, benz, Y3, P3)

Now let us consider the query *find expensive cars that are sold out.* The query can be written as **Q(N, C) = sold(N, C), expensive–car(C)?**. Then the goal clause by (4.6.2) is

← sold(n0, c0), expensive–car(c0),

where **n0** and **c0** are Skolem constants. An SLD-IA tree, defined by Definition 4.6.1, can lead to a meaningless intensional answer as shown below.

Goal:←sold(n0, c0), expensive–car(c0)

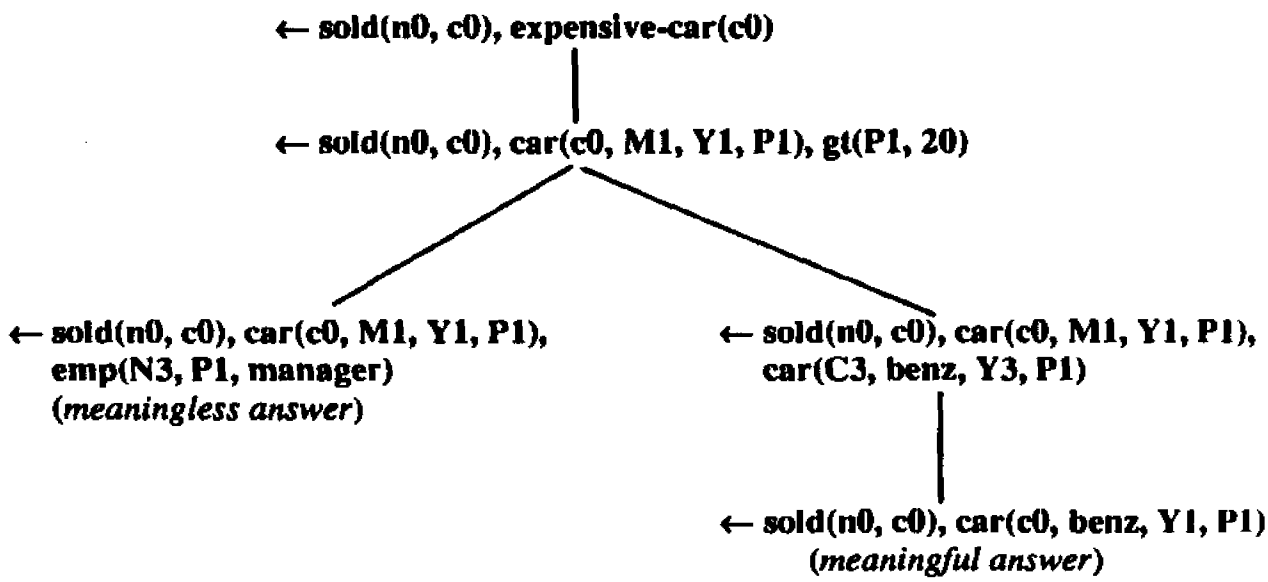


Figure 4.6.2 An SLD-IA tree that derives a meaningless intensional Answer

In the last resolvent of the left branch, semantically different variables are associated in two different extensional literals. That is, **p1** in the literal **car** is the price of a car, while **p1** in the literal **emp** is the salary of an employee. Thus this resolution branch must be discarded, even though the leaf of this branch satisfies the definition of a successful branch of an SLD-IA tree.

An SLD-IA tree in Example 4.6.2 derived a meaningless intensional answer. This kind of meaningless resolution step can be avoided by considering the semantics of literals in database. Note that rules (r3) and (r4) have the same comparison literals in the heads, but their bodies have semantically different literals. Thus, rules (r3) and (r4) must be semantically treated based on their semantics represented in the bodies.

In Section 5.2, the notions of relevant literals and relevant clauses are introduced to solve this problem. In Chapter 6, SLD resolution based on these notions is discussed.

There is another example in which SLD-IA trees cannot derive intensional answers even though intensional answers exist. This is the case when a rule has a constant in the head of the rule.

Example 4.6.3:

Suppose we have the following EDB schema and IDB about departmental database. This database is explained in Appendix C.

EDB Schema:

teaches(Tname, Dname, Cno)
enrolled(Sname, Dname, Cno)
dept(Dname, Cno, CHrs)

IDB:

(r1): teach(allen, math, Cno) ← dept(math, Cno, CHrs)
(r2): teach(baker, csc, Cno) ← dept(csc, Cno, CHrs)
(r3): teacher_of(S, T) ← enrolled(S, D, Cno), teach(T, D, Cno)
(r4): teach(T, D, Cno) ← teaches(T, D, Cno)

Let us consider a query *who are Gray's teachers?* This query can be written as **Q(T) = teacher_of(gray, T)?** The SLD-IA tree for this example is shown below:

Goal: ←teacher_of(gray, t0)

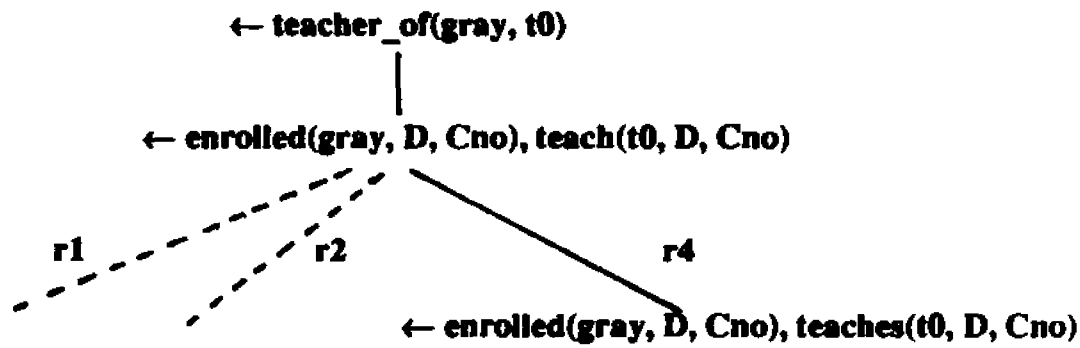


Figure 4.6.3 An SLD-IA tree that cannot derive an intensional answer

Here the second resolvent has a literal **teach(t0, D, Cno)**. This literal can be resolved with only rule (r4) in the IDB. It cannot be resolved with the rules (r1) and (r2), which have the same semantic information with rule (r4). It is the constants **allen** in rule (r1) and **baker** in rule (r2) that prevent unification with the Skolem constant **t0** in literal **teach(t0, D, Cno)**. To allow unification between them, rules (r1) and (r2) must be transformed into proper form. This problem is discussed in Section 5.2.

CHAPTER 5

FORMALIZATION OF INTENSIONAL ANSWERS

5.1. Introduction

In a logical sense, an intensional answer defined by the statement (3.3.1) in Section 3.3.1 can be any formula regardless of the formula's truth value. Thus, it is necessary to impose some conditions to $\mathbf{ans}_I(X)$ to make them acceptable answers to a given query. Such conditions will be called *restrictions* imposed on $\mathbf{ans}_I(X)$. In this chapter, such restrictions leading to acceptable meaningful intensional answers are formalized.

In Chapter 4, SLD-IA resolutions were used to derive intensional answers. As has been shown in Section 4.6.2, the blind application of an SLD-IA resolution can lead to meaningless intensional answers. Thus, the restriction strategies discussed in this chapter are imposed on SLD-IA resolutions so that they can derive meaningful intensional answers. A refined resolution strategy for deriving meaningful intensioanl answers, called an SLD-RC resolution, is discussed in Chapter 6.

The process of obtaining a meaningful intensional answer set can be described in three stages - pre-resolution stage, resolution stage, and post-resolution stage.

In the pre-resolution stage, two major actions will be taken - rule transformations and the identification of relevant literals and relevant clauses. At the stage of rule transformations, literals are checked for the assumption of unique intensional literals. Then a category of rules called non-term-restricted rules is transformed into term-

restricted rules or extended term-restricted rules. Non-term-restricted rules prevent the derivation of intensional answers for some queries. A transformation rule into extended term-restricted rule is defined and proved. The query type which requires the transformation is discussed in Section 5.2. Second, an initial set of clauses is reduced to the set of clauses which are necessary only to process a given query. The notions of relevant literals and relevant clauses are introduced for this purpose and discussed in Section 5.3.

In the resolution stage, after resolution is performed, resolvents can be either simplified or discarded. Once a new resolvent is derived from two parent clauses, it may have unifiable literals *within* a resolvent. In this case, it is said that a clause has a *factor* and a factoring rule can be applied to simplify the resolvent. Resolvents can also be checked to see whether they contain any literals which can be evaluated. If all the arguments of a comparison literal are instantiated, then the literal can be evaluated. Based on this evaluation, the resolvent clause can be either simplified or discarded. These inference rules are discussed in Section 5.4.

In the post-resolution stage, the last resolvents in success branches of a resolution tree will be negated and be taken as candidates for intensional answers. This idea is justified in Section 5.5. Evaluable literals in each candidate formula will be tested against the EDB to remove the contradictory formulas and to remove the subformula which is a tautology. This strategy is discussed in Section 5.6.

Any answer from a success branch of an SLD-RC resolution tree is a meaningful intensional answer. However, the different branches of a resolution tree might have redundant formulas in the sense that one formula can be deduced from others. Based

on these strategies, *meaningful* intensional answers and *minimal* intensional answers are formally defined in Section 5.7.

Section 8 discusses the constants which can appear in intensional answers. Finally, all these procedures are summarized in Section 5.9.

5.2. Rule Transformations

This section discusses the types of rules which are necessary for intensional query processing. Section 5.2.1 justifies the assumption of unique intensional literals. Section 5.2.2 defines a category of rules called *term-restricted* rules and extended term-restricted rules. Also the transformation of non-extended term-restricted rules into extended term-restricted rules is proved.

5.2.1. Transformation into Unique Intensional Literals

In Section 1.5, it has been assumed that a literal is either existentially defined or intensionally defined, but not both. Minker and Nicolas[Mink83] showed that one can always obtain such a partition by renaming the existential literal to p^* and introducing a new rule $p \leftarrow p^*$.

Without this assumption, an SLD-IA tree may not derive a unique tree. This is illustrated below.

Example 5.2.1:

Suppose we have the following artificial database.

EDB Schema:

p1, p2, p3, p6

IDB:

p1 ← p2, p4
p1 ← p3, p5
p4 ← p6
p5 ← p2

Note that a leaf of an SLD-IA tree consists solely of non-intensional literals. Since **p1** was both extensionally and intensionally defined, two SLD-IA trees are possible for a query clause **←p4, p1**. In a tree (1), the literal **p1** in the second resolvent is treated as an extensional literal. In a tree (2), the literal **p1** in the second resolvent is treated as an intensional literal.

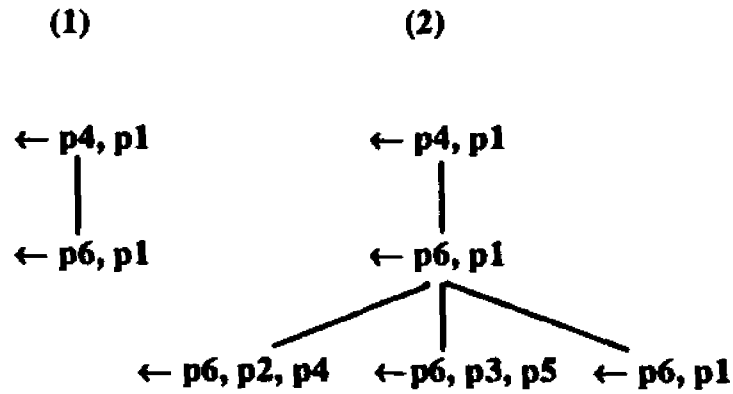


Figure 5.2.1 An example of two different SLD-IA trees for a query

In order to prevent the derivation of two different SLD-IA trees for a given query, the literal **p1** can be redefined by renaming **p1** in the EDB to **p1*** and by introducing a new rule **p1 ← p1*** in the IDB as in Example 5.2.1.

Example 5.2.2:

From the database of Example 5.2.1, **p1** in the EDB is renamed to **p1*** and a new rule **p1 ← p1*** is introduced to the IDB.

EDB Schema:
p1*, p2, p3, p6

IDB:
p1 ← p2, p4
p1 ← p3, p5
p4 ← p6
p5 ← p2
p1 ← p1*

Now, an SLD-IA tree is uniqly defined for the query **← p4, p1** below.

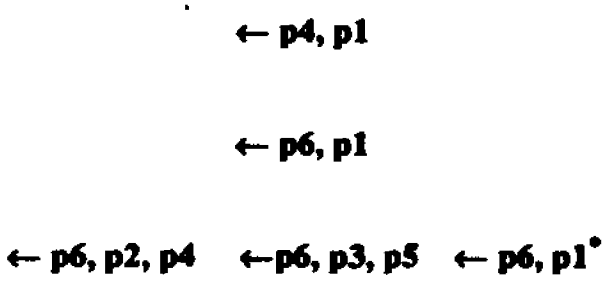


Figure 5.2.2 A unique SLD-IA tree of Figure 5.2.1 after the transformation of rules

5.2.2. Transformation into Extended Term-Restricted Rules

5.2.2.1. Term-Restricted Rules and Extended Term-Restricted Rules

A *term* in first-order logic is a constant, a variable, or a function. However, terms are usually assumed to be function-free in logic database community [Gall84]. Thus, a term is a constant or a variable in this dissertation.

The term-restricted rules are in a more strict form than are the range-restricted

rules in their syntactic requirement. *Range-restricted* rules are first defined.

Definition 5.2.1:

A rule is called *range-restricted* if all the variables in the head of a rule appear in the body.

Example 5.2.3:

The first rule is range-restricted, while the second is not.

(r1): cannot_take(Sname, Cno) ← student(Sname, Degree_Pgm, 1), Cno = 3102
(r2): love(X, Y) ← good_person(X)

Definition 5.2.2:

A rule is *term-restricted* if all the terms in the head of a rule appear in the body of the rule.

Note that all term-restricted rules are range-restricted rules, but not vice versa.

Example 5.2.4:

The first rule below is term-restricted, while the second is not.

(r1): cannot_take(Sname, Cno) ← student(Sname, under, 1), Cno > 4000
(r2): teach(baker, csc, Cno) ← dept(csc, Cno, CreditHr)

The rule (r2) is range-restricted but not term-restricted, since the constant *baker* does not appear in the body of (r2).

Definition 5.2.3:

A rule is *extended term-restricted* if it is term-restricted and does not have any constant in the head of a rule. Constants appearing in the head of a rule are called *head constants*.

All extended term-restricted rules are both range-restricted rules and term-restricted rules, but not vice versa.

Example 5.2.4:

The following rule is a term-restricted rule, but not an extended term-restricted rule.

teach(X, csc, Cno) ← dept(csc, Cno, CreditHr), (X = baker)

The head of a rule has a constant *csc* which appears both in the head and body. The constant *csc* is a head constant.

The reason we are defining a term-restricted rule is that we want all of the information in the head of a rule appear in the body. By this requirement, all the information in the head of a rule will appear in the subsequent resolvent when the rule is used in resolution. Using non-term-restricted rules or non-extended term-restricted rules in resolution for the derivation of intensional answers will prevent the derivation of intensional answers for some queries, even though intensional answers exist. This problem is discussed in Section 5.2.2.3. Thus, any rule that is not in extended term-restricted form in an IDB will be transformed into extended term-restricted rules.

5.2.2.2. Transformation into Extended Term-Restricted Rules

In this section, we are interested in the transformation of non-term-restricted rules into extended term-restricted rules. Since term-restricted rules are a subclass of extended term-restricted rules, the transformation of rules into extended-term-restricted rules will automatically put them into term-restricted rule. In these rules all variables in the head appear in the body, but one or more constants do not. In this case

we will transform the non-term-restricted rules into extended term-restricted rules as follows: Suppose a constant **a** is such a head constant. Then we replace **a** by the new variable **X** in the head, and append the equality $(X = a)$ in the body.

Suppose we have a non-term-restricted rule as follows:

$$p(X_1, \cdots, X_m, a_1, \cdots, a_n, b_1, \cdots, b_k) \leftarrow$$
$$q(X_1, \cdots, X_m, a_1, \cdots, a_n), \tag{5.2.1}$$

where X_i, $i = 1, \cdots, m$ are variables, a_i, $i = 1, \cdots, n$ are constants which appear both in the head and the body, and b_i, $i = 1, \cdots, k$ are constants which appear in the head only. We transform the rule as follows:

$$p(X_1, \cdots, X_m, Z_1, \cdots, Z_n, Y_1, \cdots, Y_k) \leftarrow$$
$$q(X_1, \cdots, X_m, a_1, \cdots, a_n), (Y_1 = b_1), \cdots, (Y_k = b_k),$$
$$(Z_1 = a_1), \cdots, (Z_n = a_n), \tag{5.2.2}$$

where Y_i and Z_i, $i = 1, \cdots k$ and $i = 1, \cdots n$ are different variables from X_j, $j = 1, \cdots, m$. It is proved that the formulas (5.2.1) and (5.2.2) are logically equivalent.

The proof of equivalence between the two formulas requires the notion of *paramodulation* [Chan73]. Paramodulation, introduced by Robinson and Wos [Robi69], is a special resolution procedure which can handle the equality literal.

Here, we briefly describe it enough to prove our theorem and leave the details to the references [for example, Chapter 8 of Chan73]. Paramodulation is a generalization of equality substitution which causes an equality substitution to take place from one clause into another. The resulting clause inferred from paramodulation is called *paramodulant*. For example, from the two parent clauses **p(a)** and **(a = b)**, paramodulation infers the paramodulant **p(b)**. The equality literal

can also have other literals in the same clause and they are inherited by the derived clause. For example, from the two clauses, $p(b) \vee \neg q(b)$ and $(b = a) \vee \neg r(Y)$, paramodulation infers the paramodulant $p(a) \vee \neg q(a) \vee \neg r(Y)$.

Theorem 5.2.1:

The formula (5.2.1) and (5.2.2) are equivalent.

Proof:

We will establish (5.2.1) $\Leftrightarrow$ (5.2.2).

($\rightarrow$) We prove (5.2.1) $\rightarrow$ (5.2.2). By deduction theorem[Ende72], it suffices to show that (5.2.1) $\wedge \neg$ (5.2.2) $\vdash \square$, where $\square$ is the empty clause. The clause form of (5.2.1) is:

$$\neg q(X_1, \cdots, X_m, a_1, \cdots, a_n) \vee p(X_1, \cdots, X_m, a_1, \cdots, a_n, b_1, \cdots, b_k) \quad (a)$$

The formula (5.2.2) after removing the implication symbol is:

$$\neg\{q(X_1, \cdots, X_m, a_1, \cdots, a_n), (Y_1 = b_1), \cdots, (Y_k = b_k), (Z_1 = a_1), \cdots, (Z_n = a_n)\} \vee p(X_1, \cdots, X_m, a_1, \cdots, a_n, Y_1, \cdots, Y_k) \quad (b)$$

The negation of the clause (b) for refutation is:

$$\exists X_1 \cdots \exists X_m \exists Z_1 \cdots \exists Z_n \exists Y_1 \cdots \exists Y_k \{q(X_1, \cdots, X_m, a_1, \cdots, a_n), (Y_1 = b_1), \cdots, (Y_k = b_k), (Z_1 = a_1), \cdots, (Z_n = a_n), \neg p(X_1, \cdots, X_m, Z_1, \cdots, Z_n, Y_1, \cdots, Y_k)\} \quad (c)$$

After skolemization, (c) becomes the following set of clauses.

$$\{q(\omega_1, \cdots, \omega_m, a_1, \cdots, a_n), (\upsilon_1 = b_1), \cdots, (\upsilon_k = b_k), (\tau_1 = a_1), \cdots, (\tau_n = a_n), \neg p(\omega_1, \cdots, \omega_m, \tau_1, \cdots, \tau_n, \upsilon_1, \cdots, \upsilon_k)\}, \quad (d)$$

where, ω_i, $i = 1, \cdots, m$, τ_i, $i = 1, \cdots, n$, and υ_i, $i = 1, \cdots, k$ are distinct Skolem constants. From the clause set of the union of (a) and (d), we apply the paramodulation. The two clauses, by paramodulation,

$(\upsilon_1 = b1)$

$\neg p(\omega_1, \cdots, \omega_m, \tau_1, \cdots, \tau_n, \upsilon_1, \upsilon_2, \cdots, \upsilon_k)$

derives a new clause (e) below.

$$\neg p(\omega_1, \cdots, \omega_m, \tau_1, \cdots, \tau_n, b_1, \upsilon_2, \cdots, \upsilon_k) \qquad (e)$$

Similarly, the repeated applications of paramodulation between the equality literals related to in clause set (d) and clause (e) will eventually derive the clause (f) as follows:

$$\neg p(\omega_1, \cdots, \omega_m, \tau_1, \cdots, \tau_n, b_1, b2, \cdots, b_k) \qquad (f)$$

The similar procedures of deriving (e) and (f) can also be applied to clause (f) and equality literals related to τ in (d). Thus, the following two clauses

$(\tau_1 = a_1)$

$\neg p(\omega_1, \cdots, \omega_m, \tau_1, \cdots, \tau_n, b_1, b_2, \cdots, b_k)$

will derive a new clause (g), by paramodulation, below.

$$\neg p(\omega_1, \cdots, \omega_m, a_1, \tau_2, \cdots, \tau_n, b_1, \upsilon_2, \cdots, \upsilon_k) \qquad (g)$$

Similarly, the repeated application of paramodulation between equality literals related to τ in (d) and (g) derives clause (h) below.

$$\neg p(\omega_1, \cdots, \omega_m, a_1, \cdots, a_n, b_1, b_2, \cdots, b_k)$$

Then, the remaining clauses are (a), (h), and (d), except equality literals, as follows:

$\{ \neg q(X_1, \cdots, X_m, a_1, \cdots, a_n) \vee p(X_1, \cdots, X_m, a_1, \cdots, a_n, b_1, \cdots, b_k),$

$\neg p(\omega_1, \cdots, \omega_m, a_1, \cdots, a_n, b_1, b2, \cdots, b_k),$

$q(\omega_1, \cdots, \omega_m, a_1, \cdots, a_n)\}$

Resolution among the three clauses above will derive the empty clause.

($\leftarrow$) We prove (5.2.1) $\leftarrow$ (5.2.2). Similarly, it suffices to show that (5.2.2) $\wedge \neg$(5.2.1)

$\vdash \square$. The clause form of (5.2.2) is:

$$\neg q(X_1, \cdots, X_m, a_1, \cdots, a_n) \vee \neg(Y_1 = b_1) \vee \cdots \vee \neg(Y_k = b_k) \vee$$
$$\neg(Z_1 = a_1) \vee \cdots \vee \neg(Z_n = a_n) \vee p(X_1, \cdots, X_m, a_1, \cdots, a_n, Y_1, \cdots, Y_k) \quad (a)$$

The clause form of (5.2.1) is:

$$\neg q(X_1, \cdots, X_m, a_1, \cdots, a_n) \vee p(X_1, \cdots, X_m, a_1, \cdots, a_n, b_1, \cdots, b_k) \quad (b)$$

The negation of clause (b) for refutation is:

$$\exists X_1 \cdots \exists X_m \, (q(X_1, \cdots, X_m, a_1, \cdots, a_n), \neg p(X_1, \cdots, X_m, a_1, \cdots, a_n, b_1, \cdots, b_k)) \quad (c)$$

After skolemization, formula (c) becomes the following two clauses:

$$q(\omega_1, \cdots, \omega_m, a_1, \cdots, a_n), \neg p(\omega_1, \cdots, \omega_m, a_1, \cdots, a_n, b_1, \cdots, b_k), \quad (d)$$

where ω_i, $i = 1, \cdots, m$ are distinct Skolem constants. The resolution from the clause set of the union of (a) and (d) will result in

$$\neg(b_1 = b_1) \vee \cdots \vee \neg(Y_k = b_k),$$

which is again reduced to the empty clause. ■

Note that every fact is a non-term-restricted rule, but we do not convert it to a term-restricted rule.

Example 5.2.6:

A non-term-restricted rule in Example 5.2.4 was:

teach(baker, csc, Cno) ← dept(csc, Cno, CreditHr)

This rule can be transformed into extended term-restricted rule by the rule (5.2.2) described above.

teach(X, Y, Cno) ← dept(csc, Cno, CreditHr), (X = baker), (Y = csc)

Actually this rule can also be written as follows:

teach(X, Y, Cno) ← dept(Y, Cno, CreditHr), (X = baker), (Y = csc)

5.2.2.3. Types of Queries which Require Term-Restricted Rules or Extended term-Restricted Rules

Not all intensional queries require rules to be in a term-restricted form. Example 4.6.3 in Section 4.6.2 is an example that an SLD-IA tree could not derive intensional answers because rules were not in a term-restricted form. Let us study the relationships between queries and rules to see what types of queries require rules to be in a term-restricted form or extended term-restricted form.

Let us consider two queries using the database in Example 4.6.3.

Q_1(T) = teacher_of(gray, T) *Who are Gray's teachers ?*
Q_2(S) = teacher_of(S, baker) *Who are Baker's students ?*

Note that **Q_1(T)** has a variable in the second argument, while **Q_2(S)** has a variable in the first argument. The goals for SLD-IA trees are:

G_1 = ←teacher_of(gray, t0)
G_2 = ←teacher_of(s0, baker)

Note that the SLD-IA tree for **G_1** is the same as Figure 4.6.3, but it is reproduced for the convenience of comparison with the SLD-IA tree for **G_2**. The SLD-IA trees for the goals are:

Goal: ←teacher_of(gray, t0)

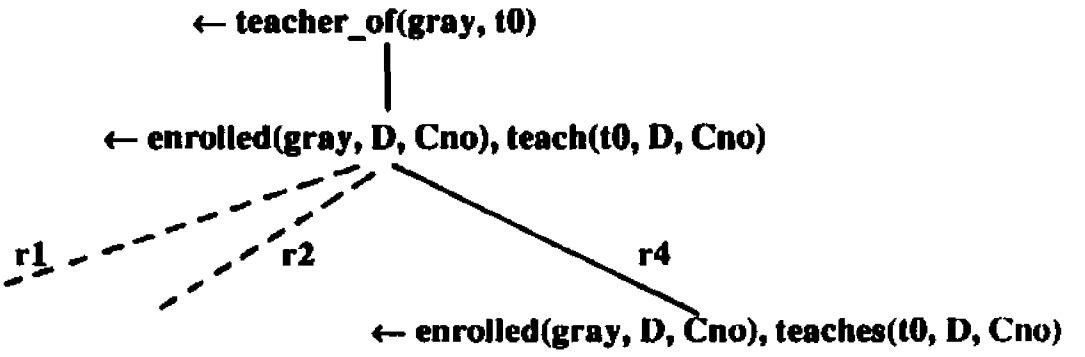
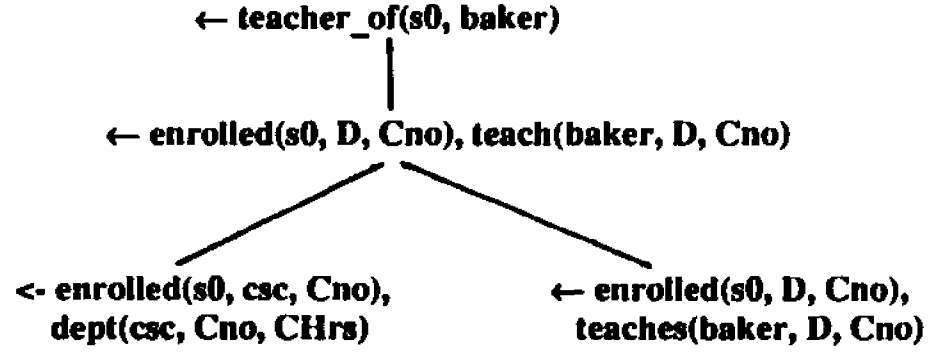


Figure 5.2.3 An SLD-IA tree for a query *teacher_of(gray, T)* in DEPARTMENT database

Goal: ←teacher_of(s0, baker)



Intensional Answers for this tree
If a student* S *is enrolled in Course* Cno *of* csc *department, then* S *is a Baker's student.
$\mathbf{ans_I^1(S) = \exists Cno\ \exists CHrs\ enrolled(S, csc, Cno), dept(csc, Cno, CHrs)}$

If a student* S *is enrolled in Course* Cno *of* D *department which Baker teaches, then* S *is a Baker's student.
$\mathbf{ans_I^2(S) = \exists D\ \exists Cno\ enrolled(S, D, Cno), teaches(baker, D, Cno)}$

Figure 5.2.4 An SLD-IA tree for a query *teacher_of(S, baker)* in DEPARTMENT database

Note that both queries use non-term-restricted rules (r2) in Figure 5.2.3 and Figure 5.2.4. While the second query in Figure 5.2.4 could derive intensional answers with rule (r2), the first query could not derive intensional answers with this rule. Also note that a variable in a query becomes a Skolem constant in a goal of SLD-IA trees. Thus the variable T in $Q_1(T)$ introduces a Skolem constant **t0** and the variable S in $Q_2(S)$ introduces a Skolem constant **s0**. Rule (r2) is a non-term-restricted rule because of the constant *baker* in the head of (r2). The constant *baker* in rule (r2) fails to unify with **t0** in Figure 5.2.3, while the constant *baker* in (r2) can unify with the same constant *baker* in Figure 5.2.4. Thus, *if the term which makes a rule non-term-restricted is to be unified with Skolem constants (i.e., variables appearing in a query formula), then the rule should be converted into term-restricted rules.*

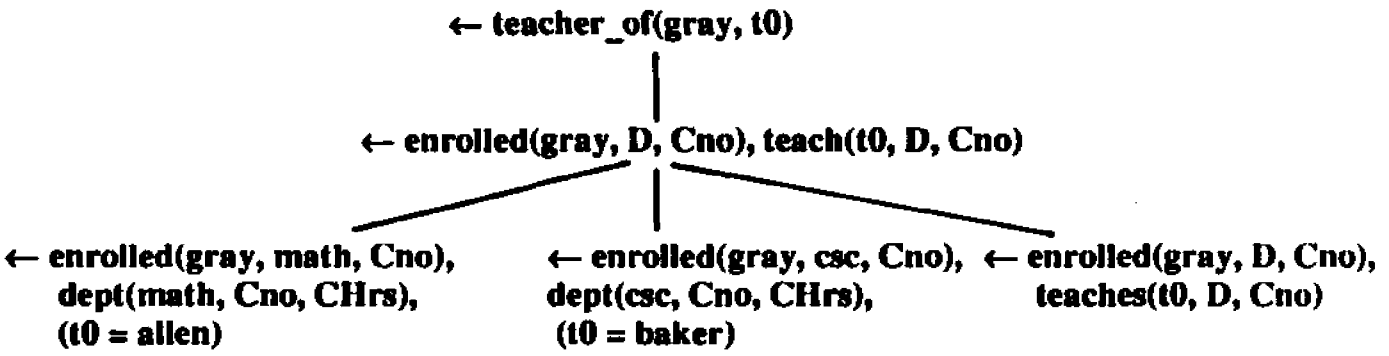
By transforming the non-term-restricted rules into term-restricted rules, all intensional answers for the query can be derived. This is illustrated below.

Example 5.2.7:

Rules after transformation into term-restricted rules:

(r1): teach(T, math, Cno) ← dept(math, Cno, CHrs), (T = allen)
(r2): teach(T, csc, Cno) ← dept(csc, Cno, CHrs), (T = baker)
(r3): teacher_of(S, T) ← enrolled(S, D, Cno), teach(T, D, Cno)
(r4): teach(T, D, Cno) ← teaches(T, D, Cno)

Query: *who are Gray's teachers?* **Q(T) = teacher_of(gray, T)?**



Intensional Answers for this tree

$ans_I^1(T) = \exists Cno\ \exists CHrs$ **enrolled(gray, math, Cno), dept(math,Cno, CHrs), (T = allen)**

$ans_I^2(T) = \exists Cno\ \exists CHrs$ **enrolled(gray, csc, Cno), dept(csc, Cno, CHrs), (T = baker)**

$ans_I^3(T) = \exists D\ \exists Cno$ **enrolled(gray, D, Cno), teaches(T, D, Cno)**

Figure 5.2.5 An SLD-IA tree with term-restricted rules in DEPARTMENT database

Three intensional answers have been derived after transforming the non-term-restricted rules into term-restricted rules.

In Example 5.2.7, transforming rules into term-restricted rules solved the problem. However, we can generalize this discussion to extended term-restricted rules. Note that if a query contains a variable, this variable becomes a Skolem constant in the root of a SLD-IA tree. Thus, any constant in the head of a rule cannot be unified with this Skolem constant. It is therefore the variables appearing in a query that require rules to be in either term-restricted or extended term-restricted form. Since a query always contains variables in our problem, we will transform any rule which has at least one constant in the head of a rule into extended term-restricted form.

5.3. Reducing an Initial Set of Rules

Obviously, not all the rules in an IDB and facts in an EDB are used to process a given query. In this section, two strategies are discussed to reduce an initial set of rules to a smaller set of rules which are actually needed to derive the intensional answers for a given query. We first discuss pure literals and then propose the notions of relevant literals and relevant clauses for this purpose. We will show that the notion of a pure literal is not powerful enough to reduce a set of clauses to those clauses of interest in the processing of a given query. That is, we show that a set of pure literals is a subset of a set of non-relevant literals. Especially, we argue that the notion of pure literals cannot be used to avoid semantically meaningless resolution. We show that the notions of relevant literals and relevant clauses can be used to avoid the meaningless intensional answers.

5.3.1. Pure Literals

The conventional method to reduce the clauses which will never be used during resolution is to use the notion of *pure literals* [Love78]. We give a definition for the *pure literal* first.

Definition 5.3.1: (Pure Literal)
A literal p is pure if and only if the clause set S does not have its complementary literal $\neg$p.

We call a clause with at least one pure literal a *pure clause* and a clause without any pure literals a *non-pure clause*. We use L_p to represent a set of pure literals, C_p for a set of pure clauses, and C_{np} for a set of non-pure clauses.

From the view point of refutation a pure clause can never be resolved into the empty clause, since it does not have a complementary literal. Thus we can remove the clauses having any number of pure literals.

Example 5.3.1:

Suppose we have the following database and a query as follows:

EDB Schema:

(r1): p1 **(r2): p2**
(r3): p3 **(r4): p7**

IDB:

(r5): p4 ← p2, p5 **(r6): p5 ← p6, p1**
(r7): p6 ← p3

Query:

← p1, p5

The set of clauses for resolution for the database above is:

(s1): p1 **(s2): p2**
(s3): p3 **(s4): p7**
(s5): p4 ∨ ¬p2 ∨ ¬p5 **(s6): p5 ∨ ¬p6 ∨ ¬p1**
(s7): p6 ∨ ¬p3 **(Q): ¬p1 ∨ ¬p5**

In the above clause set, since **p4** and **p7** do not have negative literals, the set L_p of pure literals is **{p4, p7}**. Since the clauses (s4) and (s5) contain pure literals **p7** and **p4** respectively, the set C_p of pure clauses is **{s4, s5}**.

However, the definition of a pure literal given in Definition 5.3.1 fails in a set of Horn clauses using the meta rule NAF. The example is shown below.

Example 5.3.2:

This example illustrates the conventional definition of pure literal fails in a Horn clause system that uses the NAF rule. Suppose we have the following EDB, IDB, and a query. Note that the following database is the same as that in Example 5.3.1 except

rule (r6), which contains a $\neg$**p6** instead **p6**.

EDB Schema:

(r1): p1 **(r2): p2**
(r3): p3 **(r4): p7**

IDB:

(r5): p4 $\leftarrow$ p2, p5 **(r6): p5 $\leftarrow \neg$ p6, p1**
(r7): p6 $\leftarrow$ p3

Query:

$\leftarrow$ p1, p5

The clause form of the rule (r6) is **p5 $\vee$ p6 $\vee \neg$p1** which becomes a non-Horn clause. Furthermore, **p6** becomes a pure literal because **p6** in a rule (r7) is also a positive literal. Thus, the rules (r6) and (r7) becomes a pure clause and will be removed. The set L_p of pure literals is **{p1, p4, p6}**. Hence, we cannot solve the query **$\leftarrow$ p1, p5** if we remove the pure clauses.

A pure literal in a Horn clause system using NAF can be stated by Proposition 5.3.1 below.

Proposition 5.3.1:

In a set **S** of a Horn clause system using the meta rule NAF, a literal **p** is pure iff it is never used in the body of any rules or in a given query clause **Q**.

Proof:

In deciding the pureness of a literal in a Horn clause system, the fact that we are using NAF means that we need to ignore a negative symbol $\neg$ appearing in the body of a rule. Thus we can consider that a set S has no negative symbols in the body of rules. Then, we apply the definition of pure literal given in Definition 5.3.1.

(←)

(a) Suppose the literal **p** appears in the query clause **Q**. The clause form of **Q** is ← **Q** or ¬ **Q**. That is, **p** in a query clause is a negative literal. If **p** is an EDB-defined or an IDB-defined literal, then **p** is not a pure literal. However, if **p** is neither an EDB-defined nor an IDB-defined literal, then we are using a literal which is not defined in the database. Hence, the literals appearing in a query clause are not pure.

(b) Suppose the literal **p** appears in the body of a rule. If this **p** is defined either in an EDB or in an IDB, then **p** is not pure and resolvable. Again, if **p** is neither an EDB-defined nor an IDB-defined literal, then we are using a literal which is not defined in the database. Hence, the literals appearing in the body of a rule are not pure.

(→)

Suppose the literal **p** is pure. Since **p** is pure, it is either a positive literal or a negative literal. Since every literal in a database has a positive literal (EDB-defined or IDB-defined literals are positive literals), **p** does not have negative literals. In a Horn clause system, negative literals can come only from a query clause or body of rules. Hence, **p** appears neither in a query clause nor in bodies of rules. ■

Example 5.3.2:

(a) If we apply Proposition 5.3.1 to Example 5.3.1, the literals **p4** and **p7** are pure literals since they do not appear either in a query clause or in a bodies of rules.

(b) If we apply Proposition 5.3.1 to Example 5.3.2, still the literals p4 and p7 are pure literals.

These two examples show that Proposition 5.3.1 always works in a set of Horn clause systems regardless of the existence of negation in the bodies of rules.

5.3.2. Relevant Literals and Relevant Clauses

In this section, we define the relevant literals and the relevant clauses which are only literals and clauses involved in resolution derivations for a given query. There are two major roles of relevant literals. First, they are used to eliminate unnecessry rules which are not used for the derivation of answers. Second, we have shown that, in Section 4.6.2, using comparison literals in the head of rules may derive a meaningless intensional answer. Thus, the notion of relevant literal is used to eliminate rules that are semantically unrelated rules to a given query.

Definition 5.3.2:

Let **S** be a set of database clauses consisting of {IDB} ∪ {EDB}, Q(X) be a query clause, and L_Q be a set of literals contained in Q(X). Then a set L_r of *relevant literals* to a query Q(X) is defined recursively as follows:

(1) For any literal $p \in L_Q$, $p \in L_r$

(2) Consider a rule $r \in S$ which is in the form of $a \leftarrow b_1, b_2, \cdots, b_m$, where $m \geq 0$, and a literal $p \in L_r$. Suppose there exists a most general unifier θ between a and p.

(a) If a is not a comparison literal, then both $a \in L_r$ and $b_i \in L_r$, where $i = 0, \cdots, m$.

(b) If a is a comparison literal and there exists at least one b_i such that $b_i \in L_r$, then both $a \in L_r$ and $b_i \in L_r$, where $i = 0, \cdots, m$.

(3) No other literals are relevant.

The conditions (1) and (2) of the above definition can be informally stated as follows:

(1) Any literals in a query clause are relevant literals.

(2) Suppose the head of a rule can unify with other relevant literals.

(a) If the head of the rule is not a comparison literal, then all the literals in the rule are relevant literals.

(b) If the head of the rule is a comparison literal and the body of the rule contains at least one relevant literal, then all other literals in the body of the rule are relevant literals.

The literals defined by the condition (1) are called *directly relevant literals* and those defined by the condition (2) are called *indirectly relevant literals*. Any literals which are not relevant literals are called *non-relevant literals*. A set of non-relevant literals is represented by L_{nr}.

Note that the facts in an EDB are tested for relevant literals by the condition (2)-(a), since a fact is a rule without body.

In the definition of a relevant literal we have treated the comparison literals in a special way. The reason is that the comparison literals themselves have no semantic meaning. Their semantics depends on other literals. They just perform the comparison operation using their arguments. Thus we also call the comparison

literals *semantically incomplete literals* for this reason. More generally, if the semantics of a literal is dependent on other literals, we call it a semantically incomplete literal.

Since the *pureness* of a literal is decided by the syntactic criteria - whether its complementary literal occurs or not in a clause set - the definition of pure literals is syntactical rather than semantical. On the other hand, the definition of a relevant literal is semantical, since the *relevance* of a literal is decided by the sequence of resolution derivation invoked by a query.

It is also worth noting that relevant literals always occur as a pair in the set L_r: a positive literal and a negative literal, because of the unification between them. Thus it is convenient to use only the predicate names without their signs. Hence, we represent the set of relevant predicate names as L_r.

Definition 5.3.3:

Let S be a set of database clauses consisting of {IDB} ∪ {EDB}, Q(X) be a query clause, and L_r be a set of relevant literals. Then a set C_r of *relevant clauses* to a query Q(X) is defined recursively as follows:

(1) $Q(X) \in C_r$.

(2) Consider a literal $p \in L_r$ and a rule $r \in S$ which is in the form of $a \leftarrow b_1, b_2, \cdots, b_m$, where $m \geq 0$. Suppose there exists a most general unifier θ between a and p.

(a) If a is not a comparison literal, then $r \in C_r$.

(b) If a is a comparison literal and there exists at least one b_i such that $b_i \in L_r$,

then $r \in C_r$.

(3) No other clauses are relevant.

Informally, the conditions (1) and (2) of above definition of relevant clauses can be stated as follows:

(1) A query clause is a relevant clause.

(2) Suppose the head of a rule can unify with other relevant literals.

(a) If the head is not a comparison literal, then the rule is a relevant clause.

(b) If the head is a comparison literal and the body of the rule contains at least one relevant literal, then the rule is a relevant clause.

All the other clauses are called *non-relevant clauses*.

A set of relevant clauses is represented by C_r and a set of non-relevant clauses is represented by C_{nr}. The set C_r of relevant clauses are *interesting* clauses to a given query, while the set C_{nr} of non-relevant clauses are not interesting clauses to a given query. The relevant clauses are those which are actually used in deriving the answers for a given query. We also note that the relevant clauses must contain at least one relevant literal, but not all clauses containing relevant literals are relevant clauses Example 5.3.3 below shows this.

Example 5.3.3:

To show the notions discussed in this section, we use the database of Example 5.3.1.

EDB Schema:

(r1): p1	**(r2): p2**
(r3): p3	**(r4): p7**

IDB:

(r5): p4 ← p2, p5 **(r6): p5 ← p6, p1**
(r7): p6 ← p3

Query:

(Q): ← p1, p5

The set of parameters discussed so far is shown below:

The set L_p of pure literals = **{p4, p7}**

The set C_p of pure clauses = **{(r4), (r5)}**

The set L_r of relevant literals = **{p1, p3, p5, p6}**

The set L_{nr} of non-relevant literals = **{p2, p4, p7}**

The set C_r of relevant clauses = **{(Q), (r1), (r3), (r6), (r7)}**

The set C_{nr} of non-relevant clauses = **{(r2), (r4), (r5)}**

Note that rule (r5) contains a relevant literal **p5**, but it is not a relevant clause since its head literal **p4** is not a relevant literal. That is, a clause in which the relevant literals appear only in the body, and not in the head, is not a relevant clause.

5.3.3. Relationships Between Pure Literals and Relevant Literals

We show that the set of pure literals and the set of pure clauses to a query are subsets of the set of relevant literals and the set of relevant clauses to the query, respectively. This shows that, in simplifying a set of clauses for resolution, our notion of a relevant literal is more powerful than the conventional method of a pure literal. Thus, once non-relevant clauses are removed, the pure clauses are automatically removed.

Theorem 5.3.2:

Let **S** be a set of Horn clauses, L_p be a set of pure literals, L_{nr} be a set of non-relevant literals, C_p be a set of pure clauses, C_{nr} be a set of non-relevant clauses, and **Q** be a query clause. Then we have:

$$\text{(i): } L_p \subseteq L_{nr} \tag{5.3.1}$$

$$\text{(ii): } C_p \subseteq C_{nr} \tag{5.3.2}$$

Proof:

(i) $L_p \subseteq L_{nr}$

Let **p** be a literal. We show that if $p \in L_p$, then $p \in L_{nr}$. Since **p** is a pure literal, by Proposition 5.3.1, it does not appear in a query clause **Q** or in the bodies of rules. Since **p** does not appear in **Q**, it is not a directly relevant literal. Since **p** does not appear in bodies of rules, it can never become an indirectly relevant literal. Since it is neither a directly relevant nor an indirectly relevant literal, it is not a relevant literal. Hence, $p \in L_{nr}$.

(ii): $C_p \subseteq C_{nr}$

Let **c** be a clause such that $c \in C_p$. Then **c** is a clause with at least one pure literal. Then there are two possibilities.

(a) **c** *is an extensionally-defined unit clause*. Since **c** itself is a pure literal, it is never used in the bodies of rules or in a query. The only way for the extensional literal to become a relevant literal is to be used in a query clause or bodies of rules. Hence, **c** is a non-relevant clause.

(b) **c** *is a clause whose head is a pure literal in the form of*

$p \leftarrow q1, q2, \cdots, qn$, where $n \geq 1$. In order for a clause to be a relevant clause, its head literal must be able to unify with any of the existing relevant literals. Since p is pure, it does not have its negative literal. Thus, c cannot be a relevant literal. Hence, c is a non-relevant clause.

■

The implication of this theorem is that once a set of non-relevant clauses is removed, a set of pure clauses is automatically removed. Hence, from now on, we are only concerned with relevant clauses.

Example 5.3.4:

Look at Example 5.3.3.

5.3.4. Avoiding Meaningless Resolution Using Relevant Clauses

It has been discussed that when an IDB contains rules whose heads are comparison literals meaningless intensional answers may be derived. Cholvy and Demolombe do not discuss the strategy of avoiding meaningless resolution among non-interesting predicates or different domains, in those cases. To solve this problem, we use the notion of relevant literals and relevant clauses defined in the previous section.

In Section 4.6.3, we showed an example of an SLD-IA tree which derives a meaningless intensional answer. In this section, we show that if we restrict the introduced clauses (defined in Section 4.1) to relevant clauses we can avoid deriving such answers. This is shown below.

Example 5.3.5:

We use the same database and the query of Example 4.6.2 in Section 4.6.3 to illustrate the idea.

The query was: ← **sold(N, C), expensive-car(C)** ?, asking to *find expensive cars that are sold out*. Now we compute the parameters for non-relevant clauses. They are:

Pure Literals L_p = { economic-car }

Relevant Literals L_r = { sold, expensive-car, car, gt }

Relevant Clauses C_r = { d2, d3, r1, r4, Q }

Non-relevant Clauses C_{nr} = {d1, r2, r3 }

Pure Clauses C_p = { r2 }

Note that the rule (r3) is not a relevant clause since **emp** $\notin L_r$, while (r4) is a relevant clause since **car** $\in L_r$. Thus (r3) will not be used in deriving the intensional answers for the given query. An SLD-IA tree which takes only relevant clauses as introduced clauses is shown below.

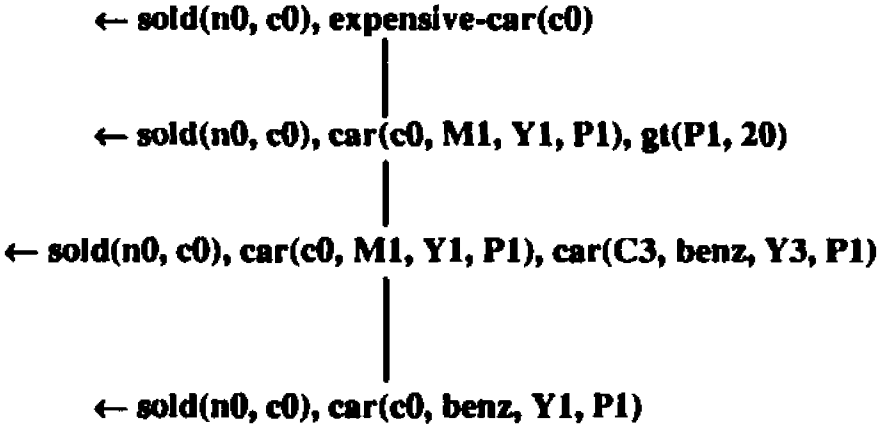


Figure 5.3.1 An augmented SLD-IA tree with a notion of relevant clauses to a query *sold(N, C), expensive-car(C)* ? in a CAR-DEALERSHIP database.

The intensional answer from the tree of Figure 5.3.1 is:

$$\mathbf{ans_I(N, C) = \exists Y1\ \exists P1\ sold(N, C) \wedge car(C, benz, Y1, P1).}$$

It can be interpreted as *If a salesman N sold out the car C whose model is benz, then it is a expensive car that is sold out.*

If we compare Figure 5.3.1 with Figure 4.6.2, Figure 5.3.1 has only one branch while Figure 4.6.2 has two branches. Since the rule (r3) is not a relevant clause, it is not resolved with **gt(P1, 20)** in the second resolvent. Hence, we removed a meaningless intensional answer by using the notion of relevant literals.

5.3.5. Relevant Literals and Intensional Answers

In this section, a relationship between relevant literals and intensional answers is discussed. It has been argued that it is necessary to impose some restrictions on the definition of intensional answers given by (3.3.1) to make them meaningful.

If only relevant clauses are used as introduced clauses for an SLD-IA tree, then all the literals appearing in subsequent resolvents will consists of relevant literals. Since relevant literals are those which are necessary to derive meaningful intensional answers, this restriction does not affect our answers. Thus, one of our restriction to $\mathbf{ans_I(X)}$ is to enforce the condition that *intensional answers must consist of relevant literals*. It can be written as follows:

Restriction 1 to the $\mathbf{ans_I(X)}$:

Let **T** be a database theory, $\mathbf{L_r}$ be a set of relevant literals in **T** to a query **Q(X)**, and $\mathbf{L_{ans_I(X)}}$ be a set of literals in $\mathbf{ans_I(X)}$ to a query **Q(X)**. Then a set $\mathbf{ANS_I^1(Q)}$ of intensional answers with the restriction 1 is defined as follows:

$$ANS_I^1(Q)=\{\ ans_I(X):\ T \vdash \forall X(ans_I(X) \rightarrow Q(X))\ \text{and}\ (L_{ans_I(X)} \subseteq L_r)\ \} \qquad (5.3.3)$$

5.4. Simplification of Resolvents

In this section, we consider the strategies to remove the redundant resolution steps during the derivation of intensional answers. Three strategies are discussed: factoring, subsumption, and evaluation of comparison literals. Factoring is an inference rule which is used *within* a clause. Subsumption is an inference rule which deletes a clause based on *another* clause. However, we show that SLD resolutions do not need subsumption test. A comparison literal whose arguments are all instantiated can be evaluated.

5.4.1. Factoring

This is an inference rule which can simplify a clause which has a *factor*. A factor is a simplified clause obtained by applying the most general unifier to the clause and removing all but one of the unified literals. For example, the factor of a clause $p(X, a) \vee p(b, Y) \vee q(W, Z)$ is $p(b, a) \vee q(W, Z)$ with the unifier of {b/X, a/Y}.

However, the blind application of factoring might result in a contradictory answer as in the example below.

Example 5.4.1:

Let us take a familiar example defining *grandfather* relation, gf(X, Y), saying that X is a Y's *grandfather* if there is a person Z such that X is a father of Z and Z is a father of Y. Here, the rule is defined as follows:

$$gf(X, Z) \leftarrow father(X, Y), father(Y, Z).$$

Suppose the query is **gf(U, V)** asking who is the grandfather of whom. Then resolution between **¬gf(U, V)** and **gf(X, Z) ∨ ¬father(X, Y) ∨ ¬father(Y, Z)** results in the resolvent **¬father(X, Y) ∨ ¬father(Y, Z)** by the unifier of **{X/U, Z/V}**. Applying factoring to the above resolvent results in **¬father(Z, Z)** by the most general unifier **{Y/X, Z/Y}**. Then the intensional answer will be a contradictory formula **f(Z, Z)**, which means a person is a *father* of oneself.

The problem in Example 5.4.1 comes from the fact that an argument in each position of a literal has a designated semantics in a database. For example, in **father(X, Y)**, **X** is a father and **Y** is a **X**'s son. As in this case, most database literals and their arguments have a specific meaning. It is evident from these observations that *we must limit the factoring operation to those cases where the result of factoring is consistent with the database semantics*. (In conventional database systems, this problem is solved by accessing the factual data of a database.) One way to implement this strategy is not to perform the factoring operation *when the same term appears in different position of argument in the literal.* We give several examples.

Example 5.4.2:

Let us consider Example 5.4.1 again. The resolvent was:

¬father(X, Y) ∨ ¬father(Y, Z)

Since the variable **Y** appears in the different position of the same literal, we do not perform factoring.

Example 5.4.3:

Suppose we have the following resolvent: **man(X, Y)** meaning that **X** is a manager

of Y.

$$\neg man(x0, Y) \vee \neg man(x0, Y)$$

In this case we can perform factoring since the variable Y appears in the same position. The result is $\neg man(x0, Y)$.

5.4.2. Subsumption

Generally, resolution process can produce not only tautologies but also subsuming clauses. Thus we need to check for both tautologies and subsumptions as we perform resolution. We discuss subsumption first and show that SLD resolutions do not need subsumption tests.

The subsumption strategy allows the deletion of a clause based on other clauses. A clause S_1 subsumes a clause S_2 if there exists a substitution σ such that the literals of $S_1\sigma$ are a subset of the literals of S_2 [Chan73]. In this case, we can delete the subsumed clause S_2, since every model of S_1 is also a model of S_2. For example, let S_1 be $p(U, V) \vee q(Z, Y)$ and S_2 be $p(X, a) \vee q(b, Y) \vee r(Z, b)$, where a and b are constants and X, Y, Z, U, and V are variables. Then the clause S_1 subsumes the clause S_2, since there is a substitution $\{X/U, a/V, b/Z\}$ which makes $S_1\sigma \subseteq S_2$.

Removing a subsumed clause does not change the satisfiability of a clause set. That is, the set of clauses after eliminating a subsumed clause is satisfiable iff the original set is satisfiable [Chan73].

As we can see from the above example, the subsumed clause has a large number of literals than the subsuming clause does. Thus the number of literals of the

resolvent from the subsumed clause is larger than that of the resolvent from the subsuming clause. The more literals the resolvents have, the greater the number of resolution steps we need to derive the empty clause. Therefore, we want to remove subsumed clause for efficient resolution.

In SLD resolutions, since we are always performing resolution between either a query clause or a resolvent and a database clause, we do not need the subsumption tests. More specifically, this can be shown below.

Example 5.4.4:

Suppose we have a resolvent $\leftarrow p, q$ in an SLD-like resolution. The clause form of the resolvent is $\neg p \vee \neg q$. The possible form of subsumed clause by the resolvent $\neg p \vee \neg q$ can be either of the following:

(a) *A clause that contains a complementary literal of subsuming clause.* For example, $\neg p \vee \neg q \vee p$ or $\neg p \vee \neg q \vee q$. In this case, these are tautologies. They will be removed by the test of a tautology.

(b) *A clause that does not contain a complementary literal of subsuming clause.* For example, $\neg p \vee \neg q \vee r$, where r is a disjunction of literals that does not contain p or q. In this case, this clause will not be involved in subsequent resolutions since it does not have any complementary literal of p or q.

Thus, we will not be further concerned with the subsumption test.

5.4.3. Evaluable Comparison Literals

The comparison literal in a resolvent of an SLD-IA tree can be evaluated to truth value if all the arguments of the literal are instantiated. For example, **gt(15, 20)** can

be evaluated while **gt(X, 20)** cannot be evaluated. We say that a literal like **gt(15, 20)** is *evaluable* and a literal like **gt(X, 20)** is not evaluable.

Suppose a resolvent **r** contains a evaluable literal **p**. Then, **r** can be written as a negative clause **← p, q**, where **q** is a conjunction of other literals. The clause form of **r** is **¬p ∨ ¬q**. Now there are three possibilities to handle **¬p ∨ ¬q**.

(1) *Reduce* **← p, q** *to* **← q** *when* **p** *is evaluated to true.* This is shown below.

← p, q
≡ ¬p ∨ ¬q
≡ ¬(True) ∨ ¬q
≡ (False) ∨ ¬q
≡ ¬q

(2) *Discard the branch of the resolution tree when* **p** *is evauated to false.*

← p, q
≡ ¬p ∨ ¬q
≡ ¬(False) ∨ ¬q
≡ (True) ∨ ¬q
≡ True

Since the resolvent is evaluated to *true*, the clause set is satisfiable and cannot derive the empty clause. Thus the resolution branch can be discarded.

(3) *Keep* **p** *when* **p** cannot be evaluated.

5.5. The Last Resolvent in a Resolution Tree

In this section, we show first-order logical representations of the last resolvent in several way. Then we show that the last resolvent is a logical consequence of a goal and an IDB. As a result, we argue that we can ignore all intermediate resolvents and, for the intensional answers to a given query, take only the last resolvents from a

resolution tree.

5.5.1. The Last resolvent

In Section 4.6, we defined the last resolvent as the leaf of the success branch of an SLD-IA resolution tree. The last resolvent therefore consists only of non-intensional literals that are either extensional literals or comparison literals. While we are deriving the intensional answers, we might have many intermediate resolvents before we reach to a leaf of an SLD-IA tree. By the definition of intensional answers, all the intermediate resolvents could be intensional answers. However, in this section, we argue that we can take only the last resolvents as the candidates of intensional answers and ignore all other intermediate resolvents. In conventional query processing systems, the extensional answers to a query is computed from an EDB. Thus, it is reasonable to derive the intensional answers that consist of either extensional literals or comparison literals.

In Section 4.1, we represented $\mathbf{R}^l$ as a resolvent of the l^{th} resolution derivation from a query clause. Let us assume that $\mathbf{C}^{l-1} \in \{\mathbf{IDB}\}$ can be unified with $\mathbf{R}^{l-1}$. Then we have:

$$\mathbf{Res}(\mathbf{R}^{l-1}, \mathbf{C}^{l-1}) = \mathbf{R}^l \tag{5.5.1}$$

This just shows a representation between $\mathbf{R}^l$ and $\mathbf{R}^{l-1}$, but not a logical relationship between them. However, the soundness of resolution states that any formula derived by a resolution principle is a logical consequence of two parent clauses. That is, $\mathbf{R}^l$ is a logical consequence of $\mathbf{R}^{l-1}$ and $\mathbf{C}^{l-1}$:

$$\mathbf{R}^{l-1} \wedge \mathbf{C}^{l-1} \rightarrow \mathbf{R}^l \tag{5.5.2}$$

Or equivalently in the form of a rule:

$$R^l \leftarrow R^{l-1}, C^{l-1} \tag{5.5.3}$$

Note that comma is used as a conjunction in the form of a rule. Based on this principle, the last resolvent can be formalized in the next section.

5.5.2. A Representation for the Last Resolvent

Let R^n be the last resolvent of a branch of an SLD-IA resolution tree, where n is the number of resolution derivations from the query clause to the last resolvent. Since the leaf of an SLD-IA tree consists of non-intensional literals, the last resolvent R^n has the following property.

A property of the last resolvent:

Let p and q be literals in R^n and R^{n-1}, respectively. Let L_{IDB} be the set of literals defined in an IDB. Then, for any $p \in R^n$, $p \notin L_{IDB}$. Also there exists at least one $q \in R^{n-1}$ such that $q \in L_{IDB}$.

This property directly comes from the definition of the leaf of an SLD-IA tree. This property states that the last resolvents consist only of non-intensional literals and R^{n-1} contains at least one intensional literal.

In the following, we derive a representation for the last resolvent based on the axiom (5.5.3). Let R^0 be the goal clause G and $C^l \in$ IDB. (Note that the following derivation uses the form of a rule, and thus comma in a body of a rule is a conjunction.) Then we have:

$$R^1 \leftarrow R^0, C^0$$
$$R^2 \leftarrow R^1, C^1$$

$$\vdots$$

$$R^n \leftarrow R^{n-1}, C^{n-1}$$

Therefore,

$$R^n \leftarrow R^{n-1}, C^{n-1}$$
$$R^n \leftarrow (R^{n-2}, C^{n-2}), C^{n-1}$$
$$R^n \leftarrow (R^{n-3}, C^{n-3}), C^{n-2}, C^{n-1}$$
$$\vdots$$
$$R^n \leftarrow R^0, C^0, C^1, \cdots, C^{n-2}, C^{n-1}$$

That is, as we could expect from the soundness of resolution, the last resolvent R^n is a logical consequence of the union of negated query clause and {IDB}. Also R^n is a logical consequence of both any intermediate resolvent and {IDB}. (Note that this does not mean there is a direct relationship between two subsequent resolvents without involving {IDB}. That is, we need {IDB} to represent the relationships between any two resolvents.) Thus, we just choose the negation of the last resolvent as a candidate for an intensional answer. Therefore, let R_i^n be the last resolvent of i^{th} success branch of an SLD-IA tree. Then we define a candidate of the i^{th} intensional answer as the negation of R_i^n as follows:

$$S_i = \neg R_i^n \qquad (5.5.4)$$

Thus, the formula which corresponds to the clause form S_i becomes a candidate of an intensional answer. This restriction allows us to choose at most one intensional answer in each branch of a resolution tree. No formula is taken as an intensional answer if the candidate evaluates to contradictory. This problem is discussed in Section 5.6.

According to the property of the last resolvent, every intermediate resolvent contains at least one intensional literal. Thus, if we take intermediate resolvents as intensional answers and, in addition, if we want to derive extensional answers from the intensional answers, then we need another inference step to derive those intensional answers to new intensional answers that consist only of non-intensional literals. Furthermore, the new intensional answers are the one that is converted from the last resolvent. Thus, when there is a possibility that we might want to compute extensional answers from intensional answers, taking intermediate resolvents needs extra inference to generate extensional answers.

5.5.3. A Generalized Representation for the Last Resolvent

In this section, we show a generalized representation of the last resolvent using the form of (5.5.1). Let $\mathbf{R^n}$ be the last resolvent, $\mathbf{R^0}$ be the query clause, and $\mathbf{C^l \in \{IDB\}}$, where $\mathbf{C^l}$ is unifiable with $\mathbf{R^l}$ by their most general unifiers. Then, we have:

$$\mathbf{Res(Res(\cdots(Res(Res(R^{n-l}, C^{n-l}), C^{n-l+1}), \cdots), C^{n-2}), C^{n-1}) = R^n} \quad (5.5.5)$$

Note that $\mathbf{R^{n-l}}$ is an arbitrary intermediate resolvent. Thus, (5.5.5) shows the relationship between an arbitrary intermediate node to the last resolvent. We explain the superscript in detail below.

(1) The number *i* is the number of resolution derivations which are necessary to derive the last resolvent from a given intermediate resolvent.

(2) The number *n-i* is the number of resolution derivations which are necessary to derive *the* intermediate resolvent from the query clause.

Let us look at two extreme cases.

(1) If we take i = n, (5.5.5) shows a complete branch from the query clause to the last resolvent as follows:

$$\mathbf{Res(Res(\cdots (Res(Res(R^0, C^0), C^1), \cdots), C^{n-2}), C^{n-1}) = R^n} \qquad (5.5.6)$$

(2) If we take i = 1, then (5.5.5) becomes a functional form which derives the last resolvent.

$$\mathbf{Res(R^{n-1}, C^{n-1}) = R^n}$$

In (5.5.5), a sequence of $C^{n-i}, C^{n-i+1}, \cdots, C^{n-1}$ is determined by Q, IDB, and a database search function. The sequence fixes a complete branch leading to the last resolvent. That is, in a given sequence $C^{n-i}, C^{n-i+1}, \cdots, C^{n-1}$, any intermediate resolvents eventually leads to the same last resolvent.

5.5.4. The Last Resolvent and Intensional Answers

In a logical sense, any intermediate resolvents can be intensional answers. However, it has been shown that the last resolvent is a logical consequence of the union of the query clause and the {IDB}. Also the last resolvent is a logical consequence of both any intermediate resolvent and the {IDB}. Thus we take only the last resolvent as a candidate of intensional answer.

In Section 5.3.5, we imposed a restriction to $\mathbf{ans_I(X)}$. Now we add one more restriction to $\mathbf{ans_I(X)}$ as follows:

Restriction 2 to the $\mathbf{ans_I(X)}$:

Let T be a database theory, L_r be a set of relevant literals in T to a query Q(X), $L_{ans_I(X)}$ be a set of literals in $\mathbf{ans_I(X)}$ to a query Q(X), and R^n be the last resolvent of

a branch in an SLD-IA resolution tree. Then a set $ANS_I^2(Q)$ of intensional answers with the restrictions 1 and 2 is defined as follows:

$$ANS_I^2(Q)=\{\ ans_I(X):\ T \vdash \forall X(ans_I(X) \rightarrow Q(X)) \text{ and } (L_{ans_I(X)} \subseteq L_r)\ \} \quad (ans_I(X) = \neg R^n)\} \tag{5.57}$$

5.6. Evaluation of EDB-defined Formulas

Facts defined in an EDB can be involved in intensional query processing. They can be used to define rules in an IDB or used in a query clause. Generally, we call the literal defined in an EDB an *EDB-defined formula*. In this section, we describe a *modified compiled approach* to handle EDB-defined formulas.

Cholvy and Demolombe [Chol86] suggest that all facts be changed into rules. For example, they argue that **man(smith)** must be converted to $man(X) \leftarrow (X = smith)$. In general, they rewrite the fact $R(a_1, \cdots, a_n)$ into $R(X_1, \cdots, X_n) \leftarrow ((X_1 = a_1), \cdots, (X_n = a_n))$; X_i are assumed to be universally quantified.

However, we argue that this conversion is inefficient and impractical where there is a large number of facts in EDB relations. The purpose of their transformation is to obtain intensional answers even when a query consists only of extensional literals. However, this approach is not consistent in the sense that these rules, transformed from facts, are not always used in resolution, when we infer from the examples used in [Chol86]. They use the rules converted from the facts for the derivation of intensional answers from the query containing a fact; they do not use those rules in other cases. Note that using those transformed rules whenever it can be

resolved introduces a lot of redundant resolution steps.

Here, we take a modified compiled approach for handling EDB-defined formulas during the derivation of intensional answers. That is, during resolution we do not evaluate EDB-defined formulas, but resolve only IDB-defined literals until we can not resolve further. When we finish a complete resolution process, we access EDB-defined relations and evaluate EDB-defined formulas, if they are evaluable. This strategy, known as compiled approach, is generally known to be more efficient than the interpreted approach, which accesses the EDB whenever needed, since compiled approach allows a query processor to access the EDB in a globally optimized manner by conventional database technology.

We call our approach a *compiled approach* in the sense that we defer the access to an EDB until we cannot resolve further. However, we call it *modified* in that we access an EDB relations only to evaluate EDB-defined formulas, if evaluable after accessing the EDB, not to instantiate the values of unknown attributes. This means we just look up the EDB relations to evaluate EDB-defined formulas so that any contradictory formulas can be removed and any tautological subformulas can be reduced.

5.6.1. Types of EDB-defined Formulas

In this section, we use the term *argument* when we mention a formula, while we use the term *attribute* when we mention database relations corresponding to the formula. Also we use the term *key arguments*, which correspond to the term *key attributes*. Note that we use the Prolog convention that variables begin with an upper

case letter and string constants with a lower case letter.

We distinguish four types of EDB-defined formulas depending on the instantiated information in the arguments of a relation. Let us use the relation, **student(Sname, Degree_Pgm, Year)** as an example, to demonstrate these types. The tuples in the relation *student* describes the student name, his/her degree program, and the number of years of studies. In the following we discuss the evaluation of EDB-defined formulas based on the instantiated values of arguments in the formula. There are four cases:

(a) *When the values of all the arguments are instantiated*

When all the arguments in an EDB-defined formula are instantiated, we can evaluate the formula by accessing the corresponding relation in the database. For example, suppose we want to evaluate the formula **student(david, under, 1)**. If **student(david, under, 1)** is a tuple in the relation *student*, then it evaluates to *true*. Otherwise it evaluates to *false*. Likewise **student(david, phd, 1)** becomes *false* since the second argument has a *false* value.

(b) *When only the values of key arguments are instantiated with non-key arguments uninstantiated.*

In this case, we cannot evaluate the formula since we do not know the values of other uninstantiated attributes. For example, **student(david, Degree_Pgm, Year)** cannot be evaluated since the values of *Degree_Pgm* and *Year* are unknown.

(c) *When the values of key arguments are instantiated with non-key arguments partially instantaited.*

In this case we can evaluate the formula only when at least one instantiated argument has a *false* value. Even when the instantiated arguments have *true* values, the value of the formula still depends on the value of uninstantiated arguments. For example, **student(david, phd, Year)** is evaluated to *false*, if *david* is in the *undergraduate* program as assumed in the previous example. However, **student(david, under, Year)** cannot be evaluated until the value of *Year* is known.

d) *When the values of key arguments are unbounded*

In this case, we cannot evaluate the formula regardless of the boundness of the non-key arguments, since we cannot identify the fact corresponding to the specific formula. For example, in the formula **student(Sname, under, 1)** we do not know whom we are talking about.

5.6.2. Removing contradictory formulas

From the logical point of view, the answer $\mathbf{ans_I(X)}$ defined by (3.3.1) can be any formula, since the *false* value of $\mathbf{ans_I(X)}$ still makes the formula $\forall \mathbf{X(ans_I(X) \rightarrow Q(X))}$ *true*.

Thus, such contradictory answers should be removed from an answer set. There are three sources of contradictory formulas. Two of them have already been discussed. The blind application of the factoring operation can introduce contradictory answers as shown in Example 5.4.1. This type of contradictory formulas can be avoided by the performing factoring operation carefully as discussed in Section 5.4.1. Comparison literals which are evaluated to *false* are also

contradictory, as discussed in Section 5.4.3. Contradictory formulas can also be found when EDB-defined formulas in the candidates of intensional answers turn out to be *false* after accessing the EDB. We want to remove these kinds of contradictory formulas from our answer set.

A candidate for intensional answers consists of one or more conjunctions of literals. An intensional answer is contradictory when at least one literal contained in the candidate of an intensional answer is evaluated to *false* against the EDB. Constants that can appear in intensional answers are only from either a query formula or rules. Thus we need to evaluate only those subformulas whose arguments have at least one instantiated value. When either the subformulas do not have any constants or the subformulas are classified as unevaluable by the criteria discussed in previous section, we do not need to access the database. Note that we assumed that constants appearing in a query formula (*query constants*) exist in the database. Without this assumption, we need to check all subformulas that have the query constants against the corresponding relation in the EDB.

Example 5.6.1:

We use example 5.2.7 in Section 5.2.2.3. There are three candidates for intensional answers derived for the query **teacher_of(gray, T)?**.

$\mathbf{ans_I^1(T) = \exists Cno\ \exists CHrs\ enrolled(gray, math, Cno), dept(math, Cno, CHrs), (T = allen)}$
$\mathbf{ans_I^2(T) = \exists Cno\ \exists CHrs\ enrolled(gray, csc, Cno), dept(csc, Cno, CHrs), (T = baker)}$
$\mathbf{ans_I^3(T) = \exists D\ \exists Cno\ enrolled(gray, D, Cno), teaches(T, D, Cno)}$

Suppose EDB relation *enrolled* has the following tuples.

enrolled	Sname	Dname	Cno
	gray	math	100
	gray	phil	300
	haas	math	200
	haas	csc	200
	james	hist	100

Subformula **enrolled(gray, math, Cno)** in $\mathbf{ans}_I^1(T)$ is not removed since *Gray* is actually enrolled in *math* department. However, subformula **enrolled(gray, csc, Cno)** in $\mathbf{ans}_I^2(T)$ is evaluated to *false* since *Gray* is not taking any *csc* courses as shown in the relation *enrolled* above. Thus the acceptable intensional answers in this case are only $\mathbf{ans}_I^1(T)$ and $\mathbf{ans}_I^3(T)$, but not $\mathbf{ans}_I^2(T)$ under the database above.

Note that the fact that we are removing $\mathbf{ans}_I^2(T)$ based on *the fact* in the EDB does not mean intensional answers are database-dependent. We remove it because the query is asking specifically about *Gray's teacher*. We must return answers only in relation to *Gray*. That is, even though intensional answers are independent of database state, they are *query dependent*.

Note that, after removing all contradictory formulas, all intensional answers can be removed and the set of intensional answers becomes the empty set. Since the empty intensional answer set can also be considered contradictory, this type of answers will be removed too.

5.6.3. Query-constant-dependent Intensional Answers and Query-constant-independent Intensional Answers

In the previous section, $\mathbf{ans}_I^2(T)$ has been removed from the set of intensional answers, since it is not related to the given query. Thus, even though $\mathbf{ans}_I^2(T)$ is independent of particular database state, it is not considered meaningful to the given query. Answers like $\mathbf{ans}_I^1(T)$ and $\mathbf{ans}_I^3(T)$ are called *query-constant-dependent (QCD) intensional answers*; while answers like $\mathbf{ans}_I^2(T)$ *is called query-constant-independent (QCI) intensional answers*. Note that if a given query does not contain any constant, then QCD intensional answers are the same as QCI intensional answers.

However, there are some advantages including QCI intensional answers to the set of intensional answers. First, the set of intensional answers including QCI intensional answers becomes completely *database-state-independent* by ignoring query constants. However, we argued that this answer is *not meaningful* to the given query. Second advantage is in computational aspects. We do not need to access an EDB to check whether each intensional answer is contradictory against the EDB. Intensional answers can be derived using only an IDB without accessing an EDB at all. However, if QCI intensional answer is evaluated against the EDB, it derives an empty set of extensional answers since it is not related to the given query.

5.6.4. Removing tautological subformulas

A clause with a tautology is the one which has a literal **p** and and its complementary literal $\neg$**p**. Since a tautology in a set does not affect the result of that set's satisfiability, we can remove any tautologies from a clause. Likewise, a

subformula which is a tautology can be reduced from the formula.

There are two types of tautologies during the derivation of intensional answers. First, if an evaluable comparison literal is evaluated to *true*, it can be reduced as discussed in Section 5.4.3. Second, if a ground subformula in a candidate of intensional answers is evaluated to *true*, then it can also be removed from the formula. A ground subformula is evaluated to *true* if the fact corresponding to the ground subformula exist in the EDB. We argue that this subformula can be removed from the answer formula as shown below.

The definition of intensional answers is:

$$\mathbf{T} \vdash \forall \mathbf{X}(\mathbf{ans}_1(\mathbf{X}) \rightarrow \mathbf{Q}(\mathbf{X}))$$

Suppose $\mathbf{ans}_1(\mathbf{X}) = \mathbf{p1}, \mathbf{p2}$. Note that comma in **p1, p2** is a conjunction in our notation. Then we have:

$$\mathbf{T} \vdash \forall \mathbf{X}((\mathbf{p1}, \mathbf{p2}) \rightarrow \mathbf{Q}(\mathbf{X}))$$

Suppose **p2** is evaluated to *true* under the current database. Then we have:

$$\mathbf{T} \vdash \forall \mathbf{X}([\mathbf{p1}, (\mathbf{true})] \rightarrow \mathbf{Q}(\mathbf{X}))$$
$$\equiv \mathbf{T} \vdash \forall \mathbf{X}(\mathbf{p1} \rightarrow \mathbf{Q}(\mathbf{X}))$$

For example, suppose a candidate for intensional answer is **q(X, a), p(a, b), r(b, Y)** and database relation **p** has a tuple **p(a, b)**. Then subformula **p(a, b)** is removed from the candidate for intensional answers and the intensional answer becomes **q(X, a), r(b, Y)**.

5.6.5. EDB-defined formulas and Intensional Answers

In Section 5.3.5, we imposed restriction 1 stating that an intensional answer must consist of relevant literals. In Section 5.5.4, we imposed restriction 2 stating that we can take only the last resolvent in an SLD-IA resolution tree as the candidate of an intensional answer.

In Sections 5.6.2 and 5.6.4, we discussed that an intensional answer must not be contradictory and must not contain any subformula which is a tautology. We designate these two conditions as restriction 3 and 4, respectively. We define a meaningful intensional answer based on these four restrictions in the next section.

5.7. Meaningful Intensional Answers and Minimal Intensional Answers

Various inferences rules and strategies to formalize intensional answers were discussed in previous sections. In this section, we formally define meaningful intensional answers and minimal intensional answers.

Definition 5.7.1:

An intensional answer, $\mathbf{ans_I(X)}$, is *meaningful* if:

(1) it is *relevant* to a given query,

(2) it is the negated formula of the last resolvent of an SLD-IA resolution tree.

(3) it is a *non-contradictory formula*, and

(4) it does not have any subformula which is a tautology.

Otherwise, it is *meaningless* (or *non-meaningful*).

We will use $\mathbf{ans_I^{mf}(X)}$ when we need to emphasize the meaningful intensional

answer. A set $ANS_I^{mf}(Q)$ of meaningful intensional answers can now be defined as follows:

Definition 5.7.2:

Let T be a database theory. Let Q(X) be a given query and G be a goal clause defined by $\leftarrow Q(x0)$, where x0 is a tuple of Skolem constant. Let L_r be the set of relevant literals to Q(X) and let L_Q be the set of literals contained in a given query clause (i.e., a set of directly relevant literals). Let $L_{ans_I(X)}$ be the set of literals contained in $ans_I(X)$. Let R^n be the last resolvent of a branch in a resolution tree whose root is G, where R^n consists only of non-intensional literals. Then, a set $ANS_I^{mf}(Q)$ of meaningful intensional answers is:

$$
\begin{aligned}
ANS_I^{mf}(Q) = \{\ & ans_I(X): T \vdash \forall X(ans_I(X) \rightarrow Q(X)) \text{ and} \\
& (1). \quad (L_{ans_I(X)} \subseteq L_r) \text{ and} \\
& (2). \quad (ans_I(X) = \neg R^n) \text{ and} \\
& (3). \quad (ans_I(X) \text{ is not a contradictory formula }) \text{ and} \\
& (4). \quad (ans_I(X) \text{ does not contain any subformula} \\
& \qquad \text{which is a tautology }) \qquad (5.7.1)
\end{aligned}
$$

The assigned numbers in $ANS_I^{mf}(Q)$ are the restriction numbers corresponding to those in Definition 5.7.1.

Note that we removed all of the intermediate nodes of a resolution tree from the definition of meaningful intensional answers. Also removed are any formulas which are contradictory or tautologies. Note that the empty formula corresponding to the empty clause in a resolvent is also removed when the contradictory formula is removed. Furthermore, a meaningful intensional answer must consist of relevant

literals. The resolution methods discussed in Chapter 4 cannot avoid introducing non-relevant clauses into resolution derivations. A new resolution strategy - which can avoid introducing non-relevant clauses and whose success branch derives a meaningful intensional answer - is defined and discussed in Chapter 6.

A set $ANS_I^{mf}(Q)$ of meaningful intensional answers, however, might contain redundant answers which are implied by other meaningful intensional answers. Thus, these redundant intensional answers can be further removed from $ANS_I^{mf}(Q)$ by considering dependencies among the answers. Let us define redundant answers.

Definition 5.7.3:

Let T be a database theory and Q be a given query. In a set $ANS_I^{mf}(Q)$ of meaningful intensional answers to the query Q, an intensional answer $ans_I^j(X)$ is redundant if there exist $ans_I^1, \cdots, ans_I^j(X)$ such that

$$T \vdash \forall X\ (F(ans_I^1(X), \cdots, ans_I^j(X)) \rightarrow ans_I^j(X)), \qquad (5.7.2)$$

where $i \geq 1$, $j > i$, and $F(a, \cdots, b)$ is first-order logic language whose predicate names are $a, \cdots, b$.

We say a set of intensional answers are *independent* if there are no redundant answers in the set. A set $ANS_I^m(Q)$ of minimal intensional answers can be defined in terms of both $ANS_I^{mf}(Q)$ and independent answers.

Definition 5.7.4:

A set $ANS_I^m(Q)$ of intensional answers is *minimal* if all of its members are *meaningful* and *independent* each other. Otherwise, it is not minimal.

$$ANS_I^m(Q)=\{\ ans_i(X):\ (ans_i(X) \in ANS_I^{mf}(Q)\ )\ \text{and}$$
$$(\text{no } ans_i(X) \text{ is a redundant intensional answer})\} \qquad (5.7.3)$$

These definitions of intensional answers will be used in SLD-RC trees, which are discussed in the next chapter.

In the following, we compare our definitions of intensional answers to that of Cholvy and Demolombe's given in (2.5.3).

(1) Their intensional answers are defined as first-order language built from a set of database predicates. However, this restriction can not remove any meaningless intensional answers to a given query. Our condition corresponding to this restriction is that intensional answers must consist of relevant literals. Since relevant literals are those which are actually used to derive meaningful intensional answers, our restriction is a more refined one.

(2) They do not remove, by definition, any intermediate resolvents as intensional answers. We argued that the last resolvent in a branch of resolution tree is a logical consequence of both a given query clause and a set of intensional database clauses. Also the last resolvent is a logical consequence of both an intermediate resolvent and a set of intensional database clauses. For this reason, we imposed the restriction that we will take only the last resolvent of a resolution tree.

(3) They do not remove any tautological subformulas. However, we argued, in Section 5.6.5, that we can remove any subformula which is a tautology from an intensional answer.

(4) We have classified intensional answers by several criteria. An intensional answer is query-constant-independent (QCI), if the validity of query constants in the intensional answer is not checked against an EDB; query-constant-dependent (QCD), if the validity of query constants in the intensional answer is checked against an EDB; meaningful, if it satisfies the four conditions discussed in this chapter; minimal; if it is not redundant in the meaningful intensional answer set.

5.8. Constants Appearing in Intensional Answers

We note that the constants appearing in intensional answers are only those which appear in a query clause or rules. Since we do not instantiate the variables in formulas against EDB, even after accessing the EDB, no other constants can appear in the meaningful intensional answers and hence in the minimal intensional answers.

We say that our meaningful intensional answers are *query-dependent* in the sense that query constants are allowed to appear in meaningful intensional answers.

5.9. Summary of Formalization

In this chapter, we have discussed a three-stage formalization process to derive meaningful intensional answers and the minimal intensional answers.

Before starting the resolution, we remove all the non-relevant clauses. Since pure clauses are a subset of non-relevant clauses, removing non-relevant clauses will automatically remove pure clauses. Therefore, we do not need to compute pure clauses. In addition, in any rule which contains a constant **a** in the head of a rule, the constant **a** is replaced by a new variable **X**, and the equality predicate $(X = a)$ is conjuncted to the body of the rule.

In the resolution stage, we check three restrictions for each resolvent. First, the two restrictions are related with any evaluable comparison literals in the resolvent. We check whether the resolvent contains any comparison literals in which all of their arguments are instantiated (i.e., evaluable)*. Then for each evaluable comparison literal, we discard the resolution branch if it is evaluated to *false*; while if it is evaluated to *true* we only remove the evaluable comparison literal from the resolvent. On the other hand, if a comparison literal is not evaluable, we keep it within the resolvent. We then check for a factor in the resolvent. If there is a factor in the resolvent, we perform the factoring operation as long as the result of factoring is consistent with the database semantics. We do not need to check for a subsumed clause, however, since an SLD-like resolution do not need this inference rule.

When the last resolvent consists only of non-intensional literals, we stop resolution. Then we negate the last resolvent and take it as a candidate for an intensional answer.

Finally, we access the EDB and evaluate EDB-defined formulas. If an EDB-defined formula in the candidate of an intensional answer is evaluable and is evaluated to *false*, then the candidate is discarded. If any subformula of the candidate is evaluated to *true*, then we reduce the candidate formula by removing the subformula from the candidate. Also removed is the empty set of intensional answers, since it is contradictory.

---

* Note that we represent the goal and each node of a resolution tree by a negative clause which is defined in (4.1.1). Thus each literal in a negative clause is a negative literal in a clause form as in (4.1.2). Therefore, the fact that a comparison literal evaluates to false in a node of a resolution tree implies that it evaluates to true in the clause form of (4.1.2), which means the clause is satisfiable. Hence we need to discard the resolution branch. Refer to Section 5.4.3 for detail.

Note that we checked twice for the contradictory and the tautological formulas - in the middle of resolution and after the end of resolution by accessing the EDB. In the first case we check whether a resolvent contains any evaluable comparison literal which can be evaluated without accessing the EDB; in the second case, we check whether an intensional answer can be evaluated after accessing the EDB.

These procedures are summarized below.

Steps for Formalization of Intensional Answers

1 (Pre-resolution)
Transform rules which have constants in their heads into extended term-restricted rules
Negate the query and convert it into the clause form
Compute non-relevant clauses and remove them

2 (Resolution)
Repeat for all branches of a resolution tree
Perform resolution
For each evaluable comparison literal in the resolvent
If it is contradictory, then discard the current branch
If it is a tautology, then remove it from the resolvent
Perform factoring, if necessary and if
the result of factoring is consistent with database semantics
Until a resolvent consists only of non-intensional literals or it can not be further resolved

3 (Post-Resolution)
Generate the candidates of intensional answers by negating all the last resolvents of the success branches of the SLD-IA tree
For all candidates, evaluate evaluable EDB-defined formulas against the EDB
Remove any contradictory formulas
Remove any subformula which is a tautology
Now the answers are meaningful intensional answers

4 (Minimal intensional answers)
Remove redundant answers from the answer set

CHAPTER 6

INTENSIONAL QUERY PROCESSING (IQP)

In Chapter 4, the SLD-IA tree which derives intensional answers was defined. For a given query Q(X), an SLD-IA tree is an SLD tree whose root node is a goal $\leftarrow$ Q(x0) and whose leaf of the success branch consists only of non-intensional literals.

However, the definition of an SLD-IA tree could not exclude the possibility of deriving meaningless intensional answers. Thus, in Chapter 5, a three-stage formalization process leading to rule transformations and to the impositions of four restrictions on $\mathbf{ans_I(X)}$ has been discussed. Based on the four restrictions imposed on $\mathbf{ans_I(X)}$, a meaningful intensional answer has been defined in Section 5.7.

In this chapter, we refine an SLD-IA tree to an SLD-RC tree by imposing four restrictions to an SLD-IA tree so that a success branch of an SLD-RC tree can derive a meaningful intensional answer. Since introduced clauses are limited to relevant clauses in an SLD-RC tree, it is proved that an SLD-RC tree is a subtree of an SLD-IA tree.

In section 6.1, the finiteness of an SLD-RC tree is proved. Also proved are the soundness and the completeness of an SLD-RC tree for intensional query processing (IQP). In section 6.2, an algorithm to derive meaningful intensional answers is presented and the correctness of the algorithm is proved.

6.1. SLD-RC Resolution

6.1.1. SLD-RC Tree

In Section 4.6.3, we have showed an example in which an SLD-IA tree derives a meaningless intensional answer. In this section, we define an SLD-RC tree which can avoid deriving meaningless intensional answers by imposing four restrictions on an SLD-IA tree.

In an SLD-RC tree, introduced clauses are limited to relevant clauses as was discussed in Section 5.3. The relevant literals and relevant clauses are those which will be actually involved in resolution for the derivation of answers for a given query. They are determined by the relationship between the literals appearing in a query clause and database clauses.

In this approach, an SLD-RC tree is built based on the relevant literals and the relevant clauses. Every node of an SLD-RC tree consists of the relevant literals. Non-relevant literals do not appear in SLD-RC trees. Only relevant clauses will be introduced clauses for resolution. Non-relevant clauses are not introduced for resolution. That is, we search clauses in the database in a top-down fashion looking for relevant clauses, skipping non-relevant clauses even though they may be unifiable with the selected literal. As we stated in Chapter 4, we will use the selection function that chooses the first intensional literal from a goal clause. This is just for simplicity and does not affect SLD resolution derivation itself [Lloy84]. The SLD-RC tree is defined below.

Definition 6.1.1:

Let **S** be a set of non-recursive Horn clauses consisting of {IDB} ∪ {EDB}, where rules are in extended term-restricted form. Let **Q(X)** be a given query and **G** be the goal clause defined by ← **Q(x0)**, where **x0** is a tuple with Skolem constants . Then the SLD-RC tree for the root of **G**, with a given selection function **SF**, is defined as follows:

(a) Each node of the tree is either a goal clause or its resolvent.

(b) The root node is **G**.

(c) Let $\leftarrow a_1, \cdots, a_m, \cdots, a_k$ $(k \geq 1)$ be a node in the tree and suppose that a_m is a selected literal for resolution. Then the node has a descendent for each *relevant clause* $a \leftarrow b_1, \cdots, b_q$ such that a_m and a are unifiable. The descendent is

$$\leftarrow (a_1, \cdots, a_{m-1}, b_1, \cdots, b_q, a_{m+1}, \cdots, a_k)\,\theta$$

where θ is the most general unifier of a_m and a.

(d) A node which consists only of non-intensional literals has no descendents. It is denoted as R^n and called *the last resolvent.*

Note that the last resolvent in condition (d) consists only of non-intensional literals. A resolvent which contains at least one intensional literal but cannot be further resolved is *not* the last resolvent by the definition.

If a comparison literal is used as the head of rules, the comparison literal is treated as an intensional literal as in SLD-IA trees.

Now we define a success branch and a failure branch of an SLD-RC tree.

Definition 6.1.2:

Let R^n be defined as it was in Definition 6.1.1. A branch of an SLD-RC tree is a success branch if

(1) it contains R^n,

(2) $\neg R^n$ is not a contradictory formula under the database, and

(3) $\neg R^n$ does not contain any subformula which is a tautology under the database.

Other branches are failure branches.

The success branch of an SLD-RC tree returns a meaningful intensional answer. However, it does not consider dependency relationships with other success branches. Thus, the success branches of an SLD-RC tree might contain redundant intensional answers. To obtain minimal intensional answers, the redundant answers, if any, must be removed. That is, a meaningful intensional answer is defined in terms of a success branch of an SLD-RC tree, while a set of minimal intensional answers is defined in terms of an SLD-RC tree.

There are important differences between an SLD-IA tree and an SLD-RC tree. While, in the latter, introduced clauses can be any unifiable clauses, in the SLD-RC tree they are limited to relevant clauses. The two trees differ as well in the manner in which their success branches are defined.

In the following, we first illustrate the SLD-RC resolution tree and then discuss a relationship between an SLD-RC tree and an SLD-IA tree.

Example 6.1.1:

We use the same database and query as in Example 4.6.2, which are reproduced below for the convenience of discussion.

EDB Schema:

(d1): emp(Name, Salary, Job-type)
(d2): car(Cno, Model, Year, Price)
(d3): sold(Name, Cno)

IDB:
(r1): expensive-car(C1) ← car(C1, M1, Y1, P1), gt(P1, 20)
(r2): economic-car(C2) ← car(C2, M2, Y2, P2), ¬ gt(P2, 5)
(r3): gt(S3, 20) ← emp(N3, S3, manager)
(r4): gt(P3, 20) ← car(C3, benz, Y3, P3)

Note that rules (r3) and (r4) are not extended term-restricted rules. So we transform them into extended term-restricted rules as follows:

(r3′): gt(S3, W3) ← emp(N3, S3, manager), (W3 = 20)
(r4′): gt(P3, W4) ← car(C3, benz, Y3, P3), (W4 = 20)

Now let us consider a query *find expensive cars that are sold out.* The query formula is **Q(N, C) = sold(N, C), expensive–car(C)?**. Then the goal clause is:

← sold(n0, c0), expensive–car(c0),

where **n0** and **c0** are Skolem constants. The directly relevant literals are **{sold, expensive–car}**. The directly relevant literal **expensive–car** makes **car** and **gt** relevant literals and the clause (r1) a relevant clause. The new relevant literal **car** is an extensional literal, thus it does not introduce any new relevant literal. The literal **gt** in clause (r1) can unify with **gt** in the rules (r3′) and (r4′). However, by the definition of relevant literal, the rule (r3′) does not introduce any new relevant literals since it does not have any relevant literals in its body. Thus (r3′) is not a relevant clause either. The rule (r4′) is a relevant clause since its body contains a relevant literal by

the definition of a relevant clause. Thus, the set $\mathbf{L_r}$ of relevant literals is **{sold, expensive–car, car, gt}**. The rule (r2) is not a relevant clause, since its head can not be unified with any other relevant literals. The set $\mathbf{C_r}$ of relevant clauses is **{Q, d2, d3, r1, r4′}**. Since (r3′) is not a relevant clause it will not be used to build a SLD-RC tree, thus avoiding meaningless resolution. This tree is shown below.

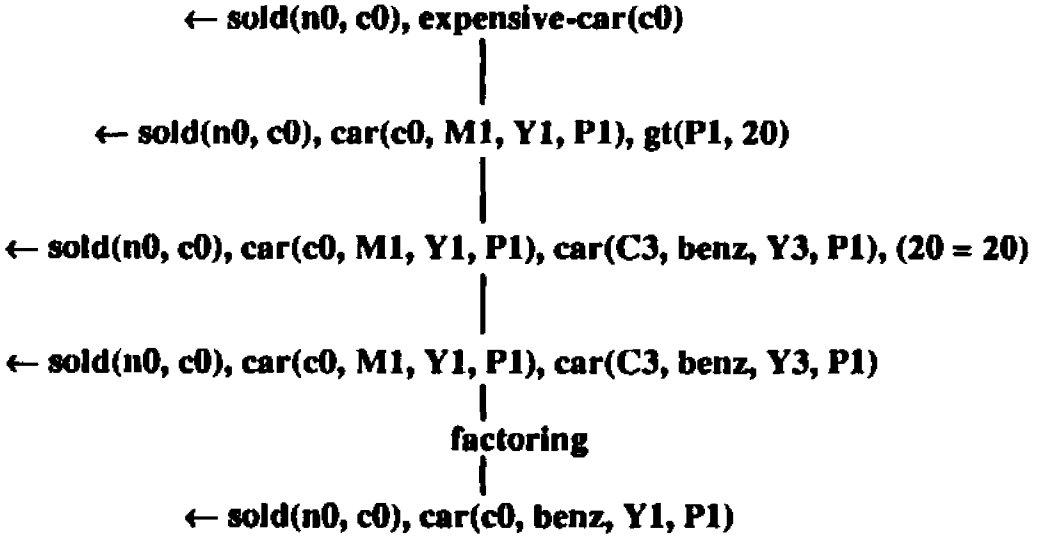


Figure 6.1.1 An SLD-RC tree to a query *sold(N, C), expensive-car(C) ?* **in a CAR-DEALERSHIP database.**

Note that this tree is similar to the augmented SLD-IA tree in Figure 5.3.1 in that both trees limit introduced clauses to relevant clauses. However, rules (r3) and (r4) were transformed to extended term-restricted rules in Figure 6.1.1, while they were not transformed in Figure 5.3.1. Since the leaf in Figure 6.1.1 consists only of extensional literals, it becomes the last resolvent. The candidate for an intensional answer is:

$$\mathbf{ans_I(X) = \exists Y1\ \exists P1\ sold(N, C), car(C, benz, Y1, P1)}$$

Since no subformula is evaluable against the EDB, the above formula becomes an

intensional answer. By the definition of intensional answer, we can interpret this answer as follows: *If a salesman N sold out the car C, whose model is benz, then it is an expensive car that is sold out.*

A relationship between an SLD-IA tree and an SLD-RC tree can be stated in the following proposition.

Proposition 6.1.1:

Let S be a set of non-recursive Horn clauses consisting of {IDB} ∪ {EDB}, where rules are in extended term-restricted form. Let Q(X) be a given query and G be the goal clause defined by ← Q(x0), where x0 is a tuple with Skolem constants. Then the SLD-RC tree of root G, with a selection function SF, is a subtree of the SLD-IA tree of root G.

Proof:

We need to show all the nodes of an SLD-RC tree is contained in an SLD-IA tree with the same edges for any given two nodes. We prove by induction on the height of the resolution tree. Clearly, the root of two trees is the same G by the definitions of an SLD-IA tree and an SLD-RC tree. This establishes the base of the induction. Suppose the hypothesis is true for the height n of an SLD-RC tree and an SLD-IA tree. This means, given a set S of database clauses, a goal G, and a selection function SF, the SLD-RC tree of the height n is a subtree of the SLD-IA tree of the height n. Note that at each node, except its leaves, the SLD-IA tree has descendents for any introduced clause which can unify with the selected literal in a node, while the SLD-RC tree has descendents only for the relevant clauses. That is, an SLD-RC tree does not have descendents for the clauses which can unify with the selected literal of the node but

which are not relevant clauses. Thus the branches in the SLD-RC tree from the height **n** to **n + 1** is a subset of those in SLD-IA tree from the height **n** to **n + 1**. Hence, all the nodes and the same edges between two nodes in an SLD-RC tree exist in an SLD-IA tree, but not vice versa. Thus the SLD-RC tree of root G is a subtree of the SLD-IA tree of the same root. ■

Example 6.1.2:

We use the database and query in Example 6.1.1. The SLD-RC tree for the query is shown in Figure 6.1.1. The SLD-IA tree using the database in Example 6.1.1 is shown as Figure 6.1.2 below. Note that Figure 6.1.2 is different from Figure 4.6.2 in that the database for Figure 6.1.2 is in extended term-restricted form while that in Figure 4.6.2 is not. Clearly Figure 6.1.1 is a subtree of Figure 6.1.2.

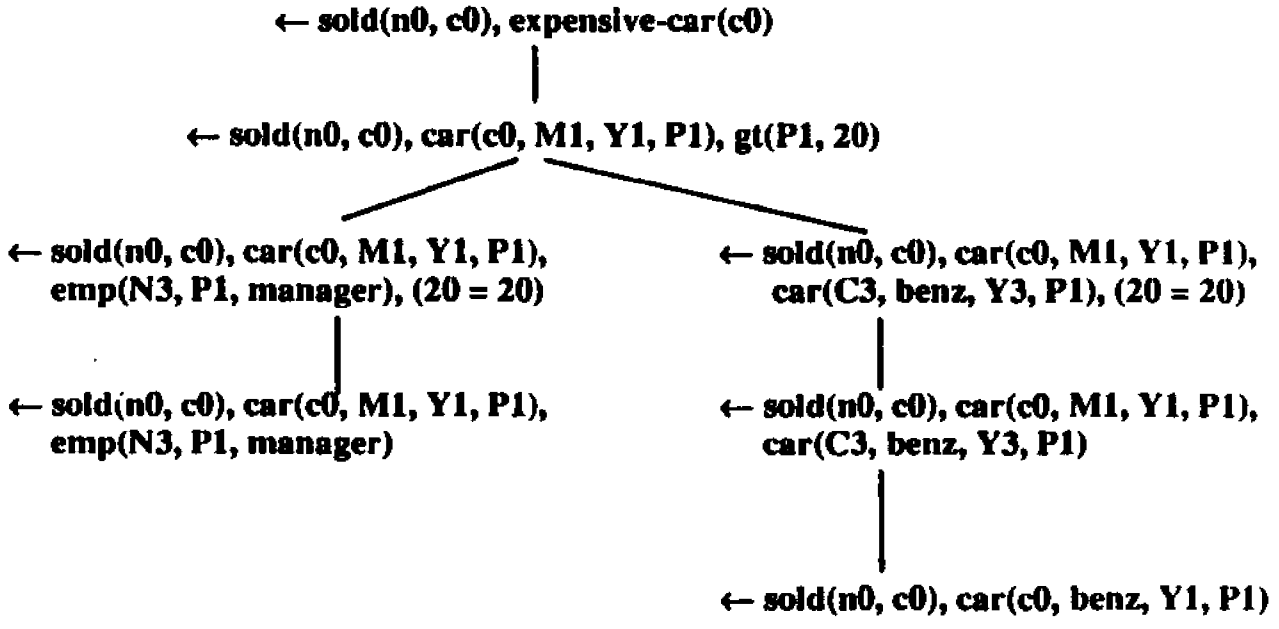


Figure 6.1.2 An SLD-IA tree using the database and query in Example 6.1.1

6.1.2. Finiteness of an SLD-RC Tree

In this section, we prove the finiteness of SLD-RC trees in the lemma below. The finiteness of SLD-RC trees can be easily seen since we are assuming that IDBs consist of non-recursive definite Horn clauses.

Lemma 6.1.2:

Let S be a set of database clauses consisting of non-recursive Horn clauses. Let G be a goal defined by $\leftarrow Q(x0)$, where x0 is a tuple with Skolem constants. Then the SLD-RC tree of root G, with a selection function SF, is always finite.

Proof:

Since the number of clauses in a set $S \cup \{G\}$ is finite, the set of intensional literals and the set of extensional literals are also finite. Since the set of intensional literals contained in a goal clause G is finite and the number of introduced clauses for each selected literal is finite, the number of branches from the root G is finite. Since each introduced clause has a finite number of intensional literals, the number of intensional literals contained in each resolvent is also finite. Since S is a set of non-recursive Horn clauses and the number of intensional literals contained in each resolvent is finite, the SLD-RC tree of root G is always finite. ■

The finiteness of an SLD-RC tree means that an SLD-RC resolution will eventually stop with a finite height of an SLD-RC tree.

6.1.3. Soundness of SLD-RC Resolution for IQP

The notions of relevant literals and relevant clauses have been defined to restrict literals and clauses to those which are actually used to generate the intensional

answers.

Based on these definitions, it has been shown in Section 5.3.4 that, using these notions, we can avoid deriving meaningless intensional answers. That is, non-relevant clauses are not necessary to derive meaningful intensional answers. We recall that the notion of relevant clauses was not used to define an SLD-IA tree, while it was used to define an SLD-RC tree. To justify using only relevant clauses in an SLD-RC tree, we claim that the intensional answers which are generated in an SLD-IA tree using non-relevant clauses are meaningless answers. This is stated as a lemma below.

Lemma 6.1.3:

Let **S** be a set of non-recursive Horn clauses consisting of {IDB} $\cup$ {EDB}, where rules are in extended term-restricted form. Let **Q(X)** be a given query and **G** be the goal clause, with a selection function **SF**, defined by $\leftarrow$ **Q(x0)**, where **x0** is a tuple with the Skolem constants. Then the SLD-IA tree of root **G**, using non-relevant clauses to a query **Q(X)** as introduced clauses, derives meaningless intensional answers.

Proof:

By the definition of meaningful intensional answers, they must consist of relevant literals to the query **Q(X)**. The intensional answers which contain non-relevant literals are not meaningful answers for the query. Since the non-relevant clauses contain non-relevant literals, resolution between a selected literal and a non-relevant clause introduces non-relevant literals into the resolvent. Thus the leaf node of this resolution branch will contain non-relevant literals. Hence it derives a meaningless

intensional answer for the query Q(X). ■

An SLD-RC tree was defined in such a way that whose introduced clauses are solely relevant clauses. It is shown that, using only relevant clauses as introduced clauses, the success branches of an SLD-RC tree do not derive meaningless intensional answers. That is, every success branch of an SLD-RC tree returns a meaningful intensional answer. This is stated as the soundness of an SLD-RC resolution for intensional query processing (IQP) below.

Theorem 6.1.4:

Let S be a set of non-recursive Horn clauses consisting of {IDB} ∪ {EDB}, where rules are in extended term-restricted form. Let Q(X) be a given query and G be the goal clause, defined by ← Q(x0), where x0 is a tuple with Skolem constants. Then any intensional answer $\mathbf{ans_I(X)}$ returned by a success branch of an SLD-RC tree of root G, with selection function SF, is a meaningful intensional answer to the query Q(X).

Proof:

We prove this theorem based on the fact that SLD resolution is sound [Lloy84]. It then suffices to show that a success branch of an SLD-RC tree derives a meaningful intensional answer to the query Q(X). That is, we need to show that a success branch of an SLD-RC tree satisfies the definition of a meaningful intensional answer to the query Q(X).

Since all literals in the query clause (i.e., G) are relevant literals by definition, the root node of an SLD-RC tree consists of a set of relevant literals. SLD-RC resolution uses only the relevant clauses as introduced clauses. By the definition of

relevant clauses, it does not contain any non-relevant literals. Thus, performing resolution between relevant literals and relevant clauses as introduced clauses does not bring any non-relevant literals to the resolvent. Since an SLD-RC tree is finite by Lemma 6.1.2, a leaf node eventually consists only of relevant literals. Thus, it satisfies the first condition of meaningful intensional answers. The last three conditions of meaningful intensional answers are satisfied by the definition of the success branch of the SLD-RC tree. Thus a success branch of an SLD-RC tree contains a meaningful intensioanl answer to the query Q(X). ■

6.1.4. Completeness of SLD-RC resolution for IQP

In the previous section, we proved the soundness of SLD-RC resolution for IQP by showing that every success branch of an SLD-RC resolution tree returns a meaningful intensional answer to a given query. In this section, we prove the completeness of SLD-RC resolution for IQP stating that a finite number of success branches of an SLD-RC tree return all meaningful intensional answers to a given query.

Theorem 6.1.5:

Let S be a set of *non-recursive Horn clauses consisting of* {IDB} ∪ {EDB}, where rules are in extended term-restricted form. Let Q(X) be a given query and G be the goal clause defined by ← Q(x0), where x0 is a tuple with Skolem constants. Then in the SLD-RC tree of root G, with a selection function SF, every meaningful intensional answer to the query Q(X), $ans_I^{mf}(X)$, has a success branch which returns an $ans_I^{mf}(X)$.

Proof:

We prove the theorem based on the fact that SLD resolution is complete [Lloy84]. It then suffices to show that any meaningful intensional answer $\mathbf{ans}_I^{mf}(X)$ has a success branch in a leaf of an SLD-RC tree. That is, if there is any $\mathbf{ans}_I^{mf}(X)$, we must find a sequence of resolution derivation, from a given goal G, which leads to an $\mathbf{ans}_I^{mf}(X)$.

We show that, from the root node of an SLD-RC tree, all branches of the SLD-RC tree are derived. From this derivation, we also show that no $\mathbf{ans}_I^{mf}(X)$ is removed from the SLD-RC tree.

A goal clause G becomes the root of an SLD-RC tree. In this root node, a literal is selected by a selection function. Suppose the selected literal can unify with k_1 relevant clauses, where $k_1 \geq 0$. Then there are k_1 descendents from G. These descendents become nodes of an SLD-RC tree. For those nodes, again a literal is selected for each node. Suppose the selected literal can unify with k_i relevant clauses, where $k_i \geq 0$. Then there will be k_i descendents for the node. Resolution derivation continues until all nodes consist only of non-intensional literals or nodes cannot be further unified. Since an SLD-RC tree is finite (Lemma 6.1.2), the resolution will stop after a height h of an SLD-RC tree. Since SLD-RC resolution is sound, the leaf nodes consist only of relevant literals. By the definition of the success branch of an SLD-RC tree, we take the clauses of the last resolvent and negate them. Among them, any contradictory formulas are removed and any subformula which is a tautology is removed.

Now we show that we do not remove any meaningful intensional answers from the SLD-RC tree derived above. First, we need to show that cutting non-relevant clauses as introduced clauses does not remove any possibility of deriving meaningful

intensional answers to the query. Since the SLD-RC tree of root G is a subtree of the SLD-IA tree of root G, Lemma 6.1.3 is used to prove this. The next two places, that cut the branches, are removing contradictory formulas from resolvents and from candidates for the intensional answers, respectively. However, since contradictory formulas are removed by the definition of meaningful intensional answers, no meaningful intensional answers are removed by these removals. The two places that simplify resolvents or formulas are also from resolvents and candidates for intensional answers. Again since tautological formulas are also removed by the definition of meaningful intensional answers, no meaningful intensional answers are removed by these simplifications. Thus, no meaningful intensional answers to the given query are removed during the derivation of an SLD-RC tree. Thus all meaningful intensional answers to the query Q(X) can be derived using an SLD-RC tree. ■

6.2. Algorithm for IQP

6.2.1. Algorithm for IQP

In this section, we present Algorithm 6.2.1 that derives a set $ANS_I^{mf}(X)$ of meaningful intensional answers. Mainly, Algorithm 6.2.1 consists of three parts: pre-resolution steps, resolution steps, and the test of meaningful intensional answers.

In the pre-resolution stage, a given query Q(X) is tested for two-trivial cases. First, if Q(X) does not contain any intensional literal, then Q(X) itself is a trivial intensional answer. Also if the query is checked for whether it contains any variable. If Q(X) does not contain a variable, then Q(X) is not an intensional query. In these

two trivial cases, the algorithm stops. If the query is not trivial, then it is negated and transformed into clause form. Also the set $\mathbf{L_r}$ of relevant literals and the set $\mathbf{C_r}$ of relevant clauses to the query **Q(X)** are computed. Then the set $\mathbf{C_{nr}}$ of non-relevant clauses are computed and removed from the set **S** of database clauses. These are done in steps 1-3 in Algorithm 6.2.1. In step 4, a variable, for the set **ASLR(Q)** of the last resolvents in an SLD-RC tree, is initialized.

Resolution steps are described by the recursive procedure **EVAL** in step 5, which accepts two input parameters: **ASLR(Q)** and $\mathbf{R^l}$, where $\mathbf{R^l}$ is a current goal to be resolved.

In the procedure **EVAL**, SLD-RC resolutions are performed and resolvents are derived. And the resolvent is set to $\mathbf{R^l}$. We check whether the resolvent contains any comparison literals in which all of their arguments are instantiated (i.e., evaluable). Then, for each evaluable comparison literal, if any of them is evaluated to *false*, $\mathbf{R^l}$ is set to the empty clause, $\square$; if it is evaluated to *true**, we remove the evaluable comparison literal. Then $\mathbf{R^l}$ is checked for a factor. If $\mathbf{R^l}$ contains a factor and no variables occur in the different positions in the same literal, then factoring is performed. These steps are repeated until $\mathbf{R^l}$ either consists of non-intensional literals or contains the empty clause. In the former case, $\mathbf{R^l}$ is unioned to the set **ASLR(Q)**, while in the latter case the resolution branch is discarded, where **ASLR(Q)** is the set of last resolvents in an SLD-RC tree of root **G**. Note that if there is no unifiable relevant clauses, step 5.2 is not executed and returned to the caller.

---

* See the foot note in Section 5.10 for the detail.

Post-resolution steps are described in step 6, 7, and 8 in Algorithm 6.2.1. Each member of the set **ASLR(Q)** is negated first and assigned to the set $ANS_I(Q)$ of intensional answers. Then, the EDB is accessed to check $ANS_I(Q)$ against the EDB. For each $F_I \in ANS_I(Q)$, if F_I is contradictory against the EDB, then F_I is removed. If F_I contains a subformula which is a tautology, then the subformula is removed. The set of each F_I becomes the set $ANS_I^{mf}(Q)$ of meaningful intensional answers to the query **Q(X)**.

Algorithm 6.2.1 (IQP)

Input: A set S of non-recursive Horn clauses consisting of {EDB} ∪ {IDB}, where rules are in extended term-restricted form, and a query Q(X), where X is a tuple of free variables, and a selection function SF.
Output: A set $ANS_I^{mf}(X)$ of meaningful intensional answers.

```
procedure EVAL (ASLR(Q), R^I)

  /* ASLR(Q): the set of last resolvents. */
  /* R^I: a given goal to be resolved. */

begin
  5.1  Choose a selected literal p from R^I by the selection function SF
  5.2  For each relevant clause c ∈ C_r, where c can unify with p do
         begin
           5.2.1  perform resolution between c and R^I
                  i := i + 1
           5.2.2  For each evaluable comparison literal q in R^I
                    begin
                      if q is a tautology then
                        remove q from R^I
                      else if q is a contradictory then
                        begin
                          R^I := □
                          return (ASLR(Q))
                        end
                    end
           5.2.3  If R^I contains a factor and no variables occur
                  in the different positions of same literal, then
```

```
                perform factoring
          5.2.4 If R^i consists only of non-intensional literals then
                begin
                   ASLR(Q) := ASLR(Q) ∪ {R^i}
                   return (ASLR(Q))
                end
                else
          5.2.5    EVAL (ASLR(Q), R^i)
        end
end; /* EVAL */


begin /* main program of Algorithm 6.2.1 */
  /* PRE-RESOLUTION STEPS */
  1  If Q(X) consists only of non-intensional literals then
       begin
          ANS_I^mf(Q) = { Q(X) }
          stop
       end
     If Q(X) does not contain any variable
       begin
          writeln ("Query is not an intensional")
          ANS_I^mf(Q) = ∅
          stop
       end
     else /* Negate the query, convert it into the clause form, and call it ¬ Q(x0),
       where x0 is a tuple of Skolem constants */
       R^0 := ¬ Q(x0)

  2  Compute a set L_r of relevant literals, a set C_r of relevant clauses,
     and a set C_nr of non-relevant clauses.

  3  Remove C_nr

  4  /* Initialize the set of candidates for the intensional answers */
     ASLR(Q) := ∅
     n := 0

  /* RESOLUTION STEP */
  5  /* Perform SLD-RC resolution */
     EVAL (ASLR(Q), R^i)

  /* POST-RESOLUTION STEPS */
  6  /* Negate the candidates of intensional answers and assign it to ANS_I(Q) */
     ANS_I(Q) := ∅
```

```
    For each S_I ∈ ASLR(Q) do
        ANS_I(Q) := ANS_I(Q) ∪ {¬ S_I}
7   /* Access the EDB */
    For each F_I ∈ ANS_I(Q) do
        begin
            If F_I is contradictory against EDB then
                ANS_I(Q) := ANS_I(Q) – F_I
            For each literal p ∈ F_I, if p is true in EDB then
                remove p from F_I
        end
8   ANS_I^mf(Q) := ANS_I(Q)
end.
```

6.2.2. Correctness of the Algorithm IQP

Algorithm 6.2.1 computes the set $ANS_I^{mf}(Q)$ of meaningful intensional answers for a given query and the database. The algorithm is actually an implementation of an SLD-RC resolution tree.

The finiteness of an SLD-RC tree has been given by Lemma 6.1.2, which means Algorithm 6.2.1 will eventually stop with a finite height of the SLD-RC tree. The soundness and the completeness of SLD-RC resolution for IQP have been proved in Theorems 6.1.4 and 6.1.5, respectively. Those two theorems state that any meaningful intensional answer computed by SLD-RC resolution is a correct meaningful intensional answer to a given input query, and all meaningful intensional answers to the query are computed by SLD-RC resolution. Since Algorithm 6.2.1 is an implementation of SLD-RC resolution, the theoretical aspects of the algorithm is justified. Now we prove the correctness of the algorithm below.

Theorem 6.2.1:

Let S be a set of non-recursive Horn clauses consisting of {EDB} ∪ {IDB}, where

rules are in extended term-restricted form. Let Q(X) be a query, where X is a tuple of free variables, and SF be a selection function for the goal of an SLD-RC tree. Algorithm 6.2.1 derives the set $ANS_I^{mf}(X)$ of all meaningful intensional answers for a query Q(X).

Proof:

To prove the correctness of the algorithm, we show that it computes all meaningful intensional answers defined by Definition (5.7.1), and it terminates for any input query Q(X).

First, we prove that the algorithm terminates for any given input query Q(X). The algorithm consists of three parts. Steps 1-4 are pre-resolution steps, step 5 is a resolution step, and steps 6-8 are steps for the test of meaningful intensional answers. Steps 1-4 and 6-8 are sequential statements, while step 5 is a procedure calling statement which invokes a recursive procedure **EVAL**. Thus it suffices to show that the procedure **EVAL** terminates for any given query Q(X). The procedure **EVAL** has two input parameters ASLR(Q) and R^I, where ASLR(Q) is a set of the last resolvents of an SLD-RC tree computed so far and R^I is a current goal to be resolved. The recursive procedure **EVAL** stops recursion when all R^Is consist only of non-intensional literals or R^I contains the empty clause. In step 5.2, since the number of relevant clause C_r is finite, the number of branches generated in this step is finite. Furthermore, since the number of intensional literals in each relevant clause is finite, R^I in step 5.2.4 will eventually consists only of non-intensional literals. When a resolvent contains an intensional literal that cannot be further resolved, step 5.2 is not executed. Hence, the procedure **EVAL** terminates the recursive calling. If R^I contains

□, then the procedure returns to the caller.

Next we prove that once the algorithm 6.2.1 terminates then it correctly computes all meaningful intensional answers. In step 1, the algorithm checks whether a query Q(X) contains any intensional literals. If Q(X) consists only of non-intensional literals, the algorithm stops processing and returns Q(X) as a trivial intensional answer. If Q(X) does not contain any variable, then it also stops with the empty set of meaningful intensional answers. If Q(X) contains any intensional literals then it is negated and converted into a clause form. In step 2, a set L_r of relevant literals and a set C_r of relevant clauses to a query Q(X) are computed. In step 3, a set C_{nr} of non-relevant clauses is removed. Thus, only relevant clauses can be involved in resolution and only relevant literals will appear in the resolvents, enforcing the condition (1) of the definition of meaningful intensional answers. In step 4, a variable ASLR(Q) for the candidates for intensional answers to the query Q(X) and the height of resolution tree i are initialized to the empty set and zero, respectively. In step 5, resolution is performed. There is one branch of a resolution tree for each relevant clause c which can unify with the selected literal p in the goal clause R^i. In step 5.2.2, we check for a tautological literal and for contradictory literals. If a resolvent contains an evaluable comparison literal which is a tautology, the literal can be removed as is shown below.

$$
\begin{aligned}
& \leftarrow F_1, \text{True} \\
\equiv\ & \neg(F_1, \text{True}) \\
\equiv\ & \neg F_1 \vee \text{False} \\
\equiv\ & \neg F_1 \\
\equiv\ & \leftarrow F_1
\end{aligned}
$$

If a literal is a contradictory one, the whole branch of a resolution tree is discarded.

Step 5.2.3 simplifies a clause when the resolvent contains a factor. Step 5.2.4 either calls the procedure **EVAL** again or takes the last resolvent and returns to the previous recursive caller.

In the post-resolution step, the candidates of intensional answers are negated in step 6. Thus the candidates satisfy the condition (4) of the meaningful intensional answers by the step 5.2.4 and step 6. In step 7, the EDB is accessed and any contradictory formulas and tautological subformulas are removed. Thus, by step 5.2.2 and step 7, answers in $\mathbf{ANS_I(Q)}$ satisfy the condition (2) and (3) of the definition of meaningful intensional answers. Any $\mathbf{ans_I(X)}$ in $\mathbf{ANS_1(Q)}$ hence is a meaningful intensional answer, and Algorithm 6.2.1 computes all meaningful intensional answers to a query **Q(X)**. ■

Example 6.1.1 in Section 6.1.1 actually used the idea in Algorithm 6.2.1. Example 7.3.1 in Section 7.3.2 is another example that uses the algorithm.

CHAPTER 7

PROPERTIES OF INTENSIONAL ANSWERS

In this chapter, we first revisit the definitions of extensional answers and intensional answers given in Chapter 3. We show that, when we use resolution to derive extensional answers and intensional answers, the definitions given in Axioms (3.2.4) and (3.3.1) are only those we can define. For example, we show that, when we use resolution for the derivation of intensional answers in Horn clause systems, we cannot derive intensional answers which are logically equivalent to a given query.

In Section 7.2, we discuss two relationships between extensional answers and intensional answers. We prove that, under the definitions of answers, given by Axioms (3.2.4) and (3.3.1), intensional answers are sufficient conditions to derive extensional answers. We show another relationship between them using new definitions defined in Section 7.1.2.

In Section 7.3, we introduce the notions of the global/local completeness of an IDB based on the relationships between extensional answers and intensional answers. Finally, we prove that all incomplete IDBs can be transformed into globally complete IDBs.

7.1. Revisit: Definitions of Answers

7.1.1. Revisit: Extensional Answers

Extensional answers, $ans_E(X)$, was defined in (3.2.4) in Chapter 3 as an axiom as follows:

$$\mathbf{T} \wedge \forall \mathbf{X}(\mathbf{Q}(\mathbf{X}) \rightarrow \mathbf{ans_E}(\mathbf{X})) \qquad (3.2.4)$$

The axiom states that any X that satisfies query Q(X) is an extensional answer, where X can be any constant in the database theory T. We note that this axiom came from the notion of Green's literal, $\mathbf{ans_E}(\mathbf{X})$, and is widely accepted as the definition of extensional answers [Gree69, Luck71, Chan73, Nils80, Gene87, Maie88].

The meaning of Axiom (3.2.4) states that, in database theory T, *any* X *which satisfies* Q(X) *is an extensional answer.* However, from this definition, *we do not know, whether all the extensional answers are derived by evaluating the* Q(X). That is, this definition itself does not give us the unique minimal set of extensional answers, though the minimality is usually assumed when we use Horn clauses [VanE76]. However, logically speaking, we must be able to compute not only all the extensional answers from the query, but also only those extensional answers satisfied by the query. In this sense, the definition of extensional answers should have logically equivalent relationship between Q(X) and $\mathbf{ans_E}(\mathbf{X})$ as follows:*

$$\mathbf{T} \wedge \forall \mathbf{X}(\mathbf{Q}(\mathbf{X}) \leftrightarrow \mathbf{ans_E}(\mathbf{X})) \qquad (7.1.1)$$

However, this logically equivalent definition does not allow us to use resolution to derive the extensional answers as can be shown below.

$$\begin{aligned}
&\mathbf{T} \wedge \forall \mathbf{X}(\mathbf{Q}(\mathbf{X}) \leftrightarrow \mathbf{ans_E}(\mathbf{X})) \\
\equiv\ &\mathbf{T} \wedge \forall \mathbf{X}\{(\mathbf{Q}(\mathbf{X}) \rightarrow \mathbf{ans_E}(\mathbf{X})) \wedge (\mathbf{ans_E}(\mathbf{X}) \rightarrow \mathbf{Q}(\mathbf{X}))\} \\
\equiv\ &\mathbf{T} \wedge \{\ \forall \mathbf{X}(\mathbf{Q}(\mathbf{X}) \rightarrow \mathbf{ans_E}(\mathbf{X})) \wedge \forall \mathbf{X}(\mathbf{ans_E}(\mathbf{X}) \rightarrow \mathbf{Q}(\mathbf{X}))\ \} \\
\equiv\ &\{\ \mathbf{T} \wedge \forall \mathbf{X}\ (\mathbf{Q}(\mathbf{X}) \rightarrow \mathbf{ans_E}(\mathbf{X}))\ \}\ \wedge\ \{\ \mathbf{T} \wedge \forall \mathbf{X}(\mathbf{ans_E}(\mathbf{X}) \rightarrow \mathbf{Q}(\mathbf{X}))\}
\end{aligned}$$

* Note that comma is used to denote a logical conjunction in rules, queries, and intensional answers, while ∧ is used to denote logical conjunction in *first-order logical formulas.*

Thus, we need to solve the following two axioms.

$$\mathbf{T} \wedge \forall \mathbf{X}\ (\mathbf{Q(X)} \rightarrow \mathbf{ans_E(X)}) \tag{7.1.2}$$

$$\mathbf{T} \wedge \forall \mathbf{X}(\mathbf{ans_E(X)} \rightarrow \mathbf{Q(X)}) \tag{7.1.3}$$

Note that the Axiom (7.1.2) is the same as the definition of extensional answers given in (3.2.4). Thus, we know we can derive extensional answers from (7.1.2). However, we show that we cannot use resolution to derive the extensional answers from Axiom (7.1.3).

Let S be a set of clauses corresponding to T. Then a set of clauses for resolution for Axiom (7.1.3) is:

$$\{\ \mathbf{S}, \neg\mathbf{ans_E(X)} \vee \mathbf{Q(X)} \tag{7.1.4}$$

We cannot prove the above goal $\neg\mathbf{ans_E(X)} \vee \mathbf{Q(X)}$ using resolution. For example, let **Q(X)** in (7.1.4) be **p1**. In order to resolve **p1** with rules, we need a rule whose head is **¬p1**. However, since we are assuming that the rules in IDB are Horn clauses, we do not have any rule whose head is a negative literal. Thus, we cannot prove a goal clause $\neg\mathbf{ans_E(X)} \vee \mathbf{Q(X)}$ corresponding to $\forall \mathbf{X}(\mathbf{ans_E(X)} \rightarrow \mathbf{Q(X)})$. Since we cannot prove $\forall \mathbf{X}(\mathbf{ans_E(X)} \rightarrow \mathbf{Q(X)})$ using resolution, neither can we prove $\forall \mathbf{X}(\mathbf{Q(X)} \leftrightarrow \mathbf{ans_E(X)})$ using resolution.

Example 7.1.1:

We use Example 3.2.2 in Section 3.2 to show the idea above. Since Axiom (7.1.2) has been proved in Example 3.2.2, we attempt to prove Axiom (7.1.3) only. The set of rules are:

at(mary, X) ← at(john, X)
at(john, X) ← at(bob, X)
at(bob, school)

Let the query **Q(X)** be **at(mary, X)?** asking *where is Mary?*. According to clause set (7.1.4), the goal to be proved is **at(mary, X)** ∨ **¬ans$_E$(X)** . Thus the set of clauses for the resolution proof is as follows:

(s1) **¬ at(john, X) ∨ at(mary, X)**
(s2) **¬ at(bob, Z) ∨ at(john, Z)**
(s3) **at(bob, school)**
(Q′) **at(mary, Y) ∨ ¬ ans$_E$(Y)**

Note that, from the above clause set, the literal **at(mary, X)** is a pure literal that does not have its complementary literal. Hence, the goal clause can not be resolved.

7.1.2. Revisit: Intensional Answers

The definition of intensional answers given by (3.3.1) in Section 3.3 is:

$$\mathbf{T} \vdash \forall \mathbf{X}(\mathbf{ans_I(X)} \rightarrow \mathbf{Q(X)}) \qquad (3.3.1)$$

That is, we want to derive a formula **ans$_I$(X)**, which is a sufficient condition for **Q(X)**, under the database theory **T**. In this section, we discuss why we have to choose only sufficient conditions, not logically equivalent conditions to **Q(X)**.

In fact, it is best if we derive intensional answers that are logically equivalent to a given query. Note that, in Axiom (3.3.1), intensional answers are sufficient conditions for **Q(X)**, meaning that *any* **X** *that satisfies* **ans$_I$(X)** *satisfies* **Q(X)** *under the database theory* **T**. However, this definition itself does not say *every* **X** *that satisfies* **Q(X)** *also satisfies* **ans$_I$(X)**. Thus, we might want to define intensional answers that

are both sufficient and necessary condition for Q(X) as follows:

$$\mathbf{T} \vdash \forall \mathbf{X}(\mathbf{ans_I(X)} \leftrightarrow \mathbf{Q(X)}) \tag{7.1.5}$$

In this case, we can say that, under the database theory T, every X that satisfies $\mathbf{ans_I(X)}$ satisfies Q(X) and those Xs are only answers that satisfy Q(X). Thus, $\mathbf{ans_I(X)}$ in (7.1.5) is logically equivalent to Q(X) under the database theory T.

We discussed intensional query processing based on the assumption that we are using resolution to derive the answers, for resolution is a simple and efficient way to derive the answers. However, proving the theorem $\forall \mathbf{X}(\mathbf{ans_I(X)} \leftrightarrow \mathbf{Q(X)})$ using resolution cause a problem as shown below.

$$\begin{aligned}
&\mathbf{T} \vdash \forall \mathbf{X}(\mathbf{ans_I(X)} \leftrightarrow \mathbf{Q(X)}) \\
\equiv\ &\mathbf{T} \vdash \forall \mathbf{X}\ \{\ (\mathbf{ans_I(X)} \rightarrow \mathbf{Q(X)}) \wedge (\mathbf{Q(X)} \rightarrow \mathbf{ans_I(X)})\ \} \\
\equiv\ &\mathbf{T} \vdash \{\ \forall \mathbf{X}(\mathbf{ans_I(X)} \rightarrow \mathbf{Q(X)}) \wedge \forall \mathbf{X}(\mathbf{Q(X)} \rightarrow \mathbf{ans_I(X)})\ \} \\
\equiv\ &\{\ \mathbf{T} \vdash \forall \mathbf{X}(\mathbf{ans_I(X)} \rightarrow \mathbf{Q(X)})\ \} \wedge \{\ \mathbf{T} \vdash \forall \mathbf{X}(\mathbf{Q(X)} \rightarrow \mathbf{ans_I(X)})\ \}
\end{aligned}$$

Thus, we need to solve the following two axioms.

$$\mathbf{T} \vdash \forall \mathbf{X}(\mathbf{ans_I(X)} \rightarrow \mathbf{Q(X)}) \tag{7.1.6}$$

$$\mathbf{T} \vdash \forall \mathbf{X}(\mathbf{Q(X)} \rightarrow \mathbf{ans_I(X)}) \tag{7.1.7}$$

Here, Axiom (7.1.6) is the same as our definition of intensional answers given in (3.3.1). We have already showed that we can prove the theorem in (7.1.6). However, we can show that we cannot prove the theorem in (7.1.7) using resolution.

The negation of theorem in (7.1.7) is:

$$\begin{aligned}
&\neg \forall \mathbf{X}\ (\mathbf{Q(X)} \rightarrow \mathbf{ans_I(X)}) \\
\equiv\ &\neg \forall \mathbf{X}\ (\neg \mathbf{Q(X)} \vee \mathbf{ans_I(X)}) \\
\equiv\ &\exists \mathbf{X}\ (\mathbf{Q(X)} \wedge \neg \mathbf{ans_I(X)})
\end{aligned}$$

The clause form after skolemization with Skolem constant **x0** is :

$$\mathbf{Q(x0) \wedge \neg ans_I(x0)}$$

Thus, the set of clauses to prove the theorem in (7.1.7) for resolution becomes:

$$\mathbf{\{S, Q(x0), \neg ans_I(x0)\}}, \tag{7.1.8}$$

where S is a set of clauses corresponding to T. Note that **Q(x0)** in (7.1.8) cannot be resolved in an intensional database (IDB) based on Horn clauses. Suppose we have a query **Q(X) = p1(X), p2(X)**. Then the goal **Q(x0)** becomes **p1(x0), p2(x0)**. In order to solve the goal **p1(x0)**, we need a rule whose head is a **¬p(Y)**. However, no head of a rule, in Horn clause systems, contains a negative literal. Thus it can not be resolved.

Thus, using resolution, we cannot prove the theorem $\mathbf{\forall X(Q(X) \rightarrow ans_I(X))}$ in (7.1.7) under the database theory T. Since we cannot prove $\mathbf{\forall X(Q(X) \rightarrow ans_I(X))}$ under the database theory T, we cannot prove $\mathbf{\forall X(ans_I(X) \leftrightarrow Q(X))}$.

Hence, if we use resolution as a computational strategy, we can derive intensional answers that are only sufficient conditions for a given query. We can not derive intensional answers that are logically equivalent to the query **Q(X)** in Horn clause systems.

Example 7.1.2:

We use the same example database and query as in Example 7.1.1.

The query **Q(X)** is **at(mary, X)?** asking *where is Mary?*. According to (7.1.8), the goal to be proved is **Q(x0) = at(mary, x0)**. Thus the set of clauses for the resolution proof is as follows:

(s1) $\neg$ at(john, X) $\vee$ at(mary, X)
(s2) $\neg$ at(bob, Z) $\vee$ at(john, Z)
(s3) at(bob, school)
(Q1) at(mary, x0)

Again the literal at(mary, x0) is a pure literal that does not have its complementary literal $\neg$at(mary, X). Hence, we cannot resolve it.

7.2. Relationships between Extensional Answers and Intensional Answers

7.2.1. Syntactic Relationship between Extensional Answers and Intensional Answers

In this section a relationship between the extensional answers and the intensional answers is shown under the definitions of answers given in (3.2.4) and (3.3.1).

Theorem 7.2.1:

Let T be a database theory and Q(X) be a query, where X is a tuple with free variables. Also let the extensional answers and the intensional answers be defined by (3.2.4) and (3.3.1), respectively. Then any X that satisfies an intensional answer is an extensional answer in database theory T:

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash (\forall X(ans_I(X) \rightarrow ans_E(X))) \qquad (7.2.1)$$

Proof:

Let Axiom (3.2.4) be (a) and Axiom (3.3.1) be (b) as follows:

$$T \wedge \forall X(Q(X) \rightarrow ans_E(X)) \qquad (a)$$
$$T \vdash \forall X(ans_I(X) \rightarrow Q(X)) \qquad (b)$$

Then by the implication tautology we have the following relationships:

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash T \qquad (c)$$

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash \forall X(Q(X) \rightarrow ans_E(X)) \qquad (d)$$

From (b) and (c), we have:

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash \forall X(ans_I(X) \rightarrow Q(X)) \qquad (e)$$

From (d) and (e), we have:

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash$$
$$(\forall X(ans_I(X) \rightarrow Q(X)) \wedge \forall X(Q(X) \rightarrow ans_E(X))) \qquad (f)$$

Thus, by the rule of transitivity, we have:

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash (\forall X(ans_I(X) \rightarrow ans_E(X))) \qquad (g)$$

■

Corollary 7.2.2:

$$(T \wedge \forall X(Q(X) \rightarrow ans_E(X))) \vdash (\forall X(ans_I^{mf}(X) \rightarrow ans_E(X))) \qquad (7.2.2)$$

Proof:

The set $ANS_I^{mf}(Q)$ of meaningful intensional answers has been defined in (5.7.1) by imposing four more restrictions on $ANS_I(Q)$. Thus, clearly, $ANS_I^{mf}(Q) \subseteq ANS_I(Q)$. Hence, Corollary 7.2.2 follows. ■

Theorem 7.2.1 implies that intensional answers are sufficient conditions to derive extensional answers. Corollary 7.2.2 implies that evaluating non-meaningful intensional answers against an EDB does not compute any additional extensional answers.

By Theorem 7.2.1, we mean that we may or may not compute all extensional

answers by evaluating intensional answers against an EDB. It implies that any X which is computed by evaluating $\mathbf{ans_I(X)}$ against the EDB must belong to extensional answers. Theorem 7.2.1 itself, however, does not tell us whether all extensional answers can be generated by evaluating all intensional answers. That is, in *some* databases with a given query Q(X), it is possible that there is at least one extensional answer which cannot be derived by evaluating the intensional answers against the EDB. However, it is also possible that all extensional answers may also be derived by evaluating intensional answers against the EDB.

Actually examples used in Chapters 3 and 4 can be used to show this theorem. Since an example used in Chapter 3 was already used in Example 3.3.2 to show the relationship between intensional answers and extensional answers, we rediscuss the example used in Chapter 4 to show the theorem.

Example 7.2.1:

The database used in Examples 4.4.1, 4.5.1, and 4.6.1 is:

EDB:

writer(smith)
smart(bob)
famous(jack)

IDB:

like(mary, X) ← respect(john, X)
respect(john, X) ← smart(X), famous(X)
respect(john, X) ← writer(X)

The query was *like(mary, X)?*. The extensional answer computed in Example 4.5.1 was:

$\mathbf{ans_E(smith)}$

The intensional answers computed in Example 4.6.1 was

$$\mathbf{ans_I^1(X) = smart(X), famous(X)}$$
$$\mathbf{ans_I^2(X) = writer(X)}$$

If we evaluate intensional answers against the EDB, $\mathbf{ans_I^1(X)}$ generates no extensional answers and $\mathbf{ans_I^2(X)}$ generates an extensional answer **smith**.

However, in this example, intensional answers were both sufficient and necessary conditions in the derivation of extensional answers. An example which intensional answers are only sufficient, not both sufficient and necessary, conditions are shown in Example 7.3.1.

We call the relationship defined by Theorem 7.2.1 a *syntactic relationship* in that the definitions of answers used to derive Theorem 7.2.1 are based on a particular computational methodology, resolution, and that a logically equivalent relationship cannot be derived.

7.2.2. Semantic Relationship Between Extensional Answers and Intensional Answers

In Section 7.1, we argued that our definitions of extensional answers and intensional answers are based on resolution. Logically the definition of extensional answers should be (7.1.1) rather than (3.2.4) as shown here:

$$\mathbf{T} \wedge \forall \mathbf{X}(\mathbf{Q(X)} \leftrightarrow \mathbf{ans_E(X)}) \qquad (7.1.1)$$

Also the logical definition of intensional answers, as discussed in Section 7.1.2, should be (7.1.5), rather than (3.3.1):

$$T \vdash \forall X(ans_I(X) \leftrightarrow Q(X)) \qquad (7.1.5)$$

Although, under Horn clause systems, resolution method does not allow us to derive intensional answers under these definitions, it is nontheless interesting to derive by these definitions the logical relationship between intensional answers and extensional answers.

Theorem 7.2.3:

Let T be a database theory and Q(X) be a query, where X is a tuple with free variables. Also let the extensional answers and the intensional answers be defined by (7.1.1) and (7.1.5), respectively. Then, any X that satisfies an intensional answer is an extensional answer and all extensional answers can be derived by evaluating all intensional answers against the EDB:

$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash (\forall X(ans_I(X) \leftrightarrow ans_E(X))) \qquad (7.2.3)$$

Proof:

Let Axiom (7.1.1) be (a) and Axiom (7.1.5) be (b) as follows:

$$T \wedge \forall X(Q(X) \leftrightarrow ans_E(X)) \qquad (a)$$
$$T \vdash \forall X(ans_I(X) \leftrightarrow Q(X)) \qquad (b)$$

Then by the implication tautology we have the following relationships:

$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash T \qquad (c)$$
$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash \forall X(Q(X) \leftrightarrow ans_E(X)) \qquad (d)$$

From (b) and (c), we have:

$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash \forall X(ans_I(X) \leftrightarrow Q(X)) \qquad (e)$$

From (d) and (e), we have:

$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash$$
$$(\forall X(ans_I(X) \leftrightarrow Q(X)) \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \qquad (f)$$

Thus, by the rule of transitivity, we have:

$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash (\forall X(ans_I(X) \leftrightarrow ans_E(X))) \qquad (g)$$

■

Corollary 7.2.4:

$$(T \wedge \forall X(Q(X) \leftrightarrow ans_E(X))) \vdash (\forall X(ans_I^{mf}(X) \leftrightarrow ans_E(X))) \qquad (7.2.4)$$

Proof:

Similar to the proof of Corollary 7.2.2. ■

Theorem 7.2.3 implies that intensional answers are sufficient and necessary conditions to derive extensional answers under the logical definitions of answers, given in (7.1.1) and (7.1.5). That is, under these definitions, not only are all extensional answers able to be derived by evaluating all intensional answers, but also those extensional answers are the only answers which are implied by the query Q(X).

Also note that the existence of Theorem 7.2.3 does not harm the validity of Theorem 7.2.1, since Theorem 7.2.1 is a special case of Theorem 7.2.3.

We cannot give an example showing the idea of this theorem, since we cannot compute intensional answers and extensional answers based on the definition given by (7.1.1) and (7.1.5).

We call the relationship defined by Theorem 7.2.3 a *semantic relationship* in that the definitions of answers used to derive Theorem 7.2.3 is independent of any

particular computational methodology and based on the semantics of terms.

7.2.3. Syntactic Relationship, Semantic relationship, and Completeness of IDB

In the previous section, we argued that the syntactic relationship defined by Theorem 7.2.1 relies on a particular computational methodology, while the semantic relationship defined by Theorem 7.2.3 is independent of any particular computational methodology and completely based on their semantics of terms. Therefore, we can say that it is the semantic relationship that is the true relationship between extensional answers and intensional answers.

Hence, if the database semantics in both an EDB and an IDB is correctly modeled, then the semantic relationship must always hold. Thus, we can say that, while the syntactic relationship always hold regardless of the correctness of the contents of database, the semantic relationship holds only when the database is correctly modeled. That is, when the semantic relationship holds, we can always derive all extensional answers by evaluating all extensional answers against an EDB and those extensional answers are only correct answers that satisfy the query. When the syntactic relationship hold, we can derive extensional answers by evaluating all intensional answers, but we do not know whether these extensional answers are the only extensional answers that satisfy the query.

Based on the notions above, we discuss the completeness and incompleteness of an IDB to a given query. Here, by the completeness of an IDB, we mean whether a database contains all the information we need. Note that we want to relate an *IDB* to the notions of completeness and incompleteness to a given query. In fact, the notion

of completeness of database must rely on the contents of both the EDB and the IDB in a DDB, but not on a query. However, if we just consider the completeness of the IDB, then it relies on both the EDB and a given query. Since the completeness of an EDB is a complex problem involving issues such as database modeling, normalization, and data entry, it is reasonable to assume that an EDB is complete and correct. Once we assume that an EDB is complete and correct, the completeness of the IDB relies on a given query. Since a query uses a subset of rules in the IDB, the same IDB can be complete for a query and incomplete for another query. Thus we discuss the notion of the completeness of an IDB only in relation to a given query on the assumption that EDB is complete and correct.

Based on the discussion above, we say that an IDB is *complete* if the semantic relationship holds. Furthermore, we say that an IDB is *globally complete* if semantic relationship always holds regardless of a given query; we say that an IDB is *locally complete to a given query* if the completeness of the IDB depends on the given query (i.e., the validity of the semantic relationship depends on the given query). Note that the syntactic relationship always holds regardless of the global or local completeness. We discuss these ideas in more detail in the next section.

7.3. Completeness of Intensional Databases (IDBs)

In Section 7.2.3, we have discussed the notion of completeness of an IDB based on the semantic relationship and the syntactic relationship between intensional answers and extensional answers. In this section, we formally define the completeness of an IDB and give an example, showing that the properties of intensional answers

can be used to detect a locally complete IDB. We also prove that every incomplete IDB can be transformed into a globally complete IDB.

7.3.1. Completeness and Consistency

In this section, we compare the completeness and consistency of an IDB.

The completeness of an IDB is different from the consistency of IDB. *The completeness of an IDB is concerned with whether the IDB has enough rules to correctly relate the IDB with the EDB.* That is, the rules defined in the IDB should correctly reflect their semantics based on the relations defined in the EDB. However, we can not tell whether the IDB fullly describes the intended database semantics of the Universe of Discourse. The fullness of description depends on the IDB designer and real-world semanticsm being modeled. We can only say that the rules, once they are defined, must correctly reflect their semantics based on the EDB. Thus, by the completeness of an IDB, we are talking about the structure of the IDB in relation to the EDB. Hence, the completeness of an IDB addresses the problem of an IDB design.

For example, given an EDB, suppose an IDB designer wants to create an intensional literal **p**. Note that whether creating **p** or not is the problem of application needs and is decided by the IDB designer. However, once the intensional literal **p** is defined in the IDB, it must fully describe the semantics of **p** in relation to other EDB relations and other intensional literals.

On the other hand, the consistency of an IDB is concerned with whether the rules in the IDB are contradictory among themselves or against EDB. Checking

consistency in deductive database systems is much more complicated than doing so in conventional database systems, because of the existence of rules in an IDB. Checking consistency in deductive database systems is widely studied. This problem therefore will not be further considered in this work.

In a sense, the completeness of an IDB can be considered as a first step to the consistency of the IDB. The consistency of an IDB, without the completeness of the IDB, is only partially correct. An IDB must correctly represent the semantics of the IDB related to the EDB.

7.3.2. Global Completeness and Local Completeness

In this section, we formally define the completeness of an IDB to a given query based on the semantic relationship between intensional answers and extensional answers.

We first define $ext(ANS_I^{mf}(Q))$ as a set of all extensional answers generated by evaluating all meaningful intensional answers against an EDB. That is, $ext(ANS_I^{mf}(Q))$ is:

$$ext(ANS_I^{mf}(Q)) = \cup\, ext(ans_I^{mf}(X)), \tag{7.3.1}$$

where $ext(ans_I^{mf}(X))$ is a set of extensional answers generated by evaluating a meaningful intensional answer against the EDB.

Before the formal definition of the completeness of an IDB, we first discuss the idea behind the definition. The test of completeness of an IDB is only available with queries. Furthermore, in order to test, we must know the exact extensional answers to the queries. Therefore, we first assume that there is a method which can compute the

set $ANS_E'(Q)$ of correct extensional answers to a query Q(X) satisfying Definition (7.1.1). Note that, using resolution, we cannot compute extensional answers based on Definition (7.1.1). Thus, we are assuming a method that can compute all correct extensional answers, based on the semantics of database and a given query. One way to compute them is to examine the contents of the EDB and IDB related to a given query.

We then compute a set $ANS_I^{mf}(Q)$ of meaningful intensional answers by Algorithm 6.2.1. If $ext(ANS_I^{mf}(Q)) = ANS_E'(Q)$, then we say that the IDB is locally complete to a given query Q(X). For a given IDB and a query, we know that the syntactic relationship always holds, but we do not know whether the semantic relationship holds. Thus, if $ext(ANS_I^{mf}(Q)) = ANS_E'(Q)$, then we know that the semantic relationship holds for the query Q(X) and the IDB. Otherwise, the semantic relationship does not hold for the query Q(X). Now we define the completeness of IDB to a given query based on the discussion above.

Definition 7.3.1:

Let $ANS_E'(Q)$ be a set of all the correct extensional answers, which satisfy Definition (7.1.1), to a query Q(X). Let $ANS_I^{mf}(Q)$ be a set of meaningful intensional answers to the query Q(X). Then an IDB is *locally complete* to a given query Q(X) iff

$$ANS_E'(Q) = ext(ANS_I^{mf}(Q)), \tag{7.3.2}$$

where $ext(ANS_I^{mf}(Q))$ is defined by (7.3.1). Otherwise, the IDB is *incomplete* to the query Q(X). We say that the IDB is *globally complete* iff relationship (7.3.2) always holds for any query Q(X).

We give examples for the incompleteness and local completeness of an IDB for two different queries below:

Example 7.3.1:

Suppose we have the following EDB schema and IDB that are described in Appendix C.

EDB Schema:

teaches(Tname, Dname, Cno)
enrolled(Sname, Dname, Cno)
dept(Dname, Cno, CHrs)

IDB

(r1): teach(allen, math, Cno) ← dept(math, Cno, CHrs)
(r2): teach(baker, csc, Cno) ← dept(csc, Cno, CHrs)
(r3): teacher_of(S, T) ← enrolled(S, D, Cno), teach(T, D, Cno)

Suppose the query is **Q(T) = teacher_of(gray, T)?** asking *who are Gray's teachers?*. After examining the EDB and IDB in Appendix C, we see that the set $ANS_E'(Q)$ of extensional answers for the query **Q(T)** is **{allen, baker, cook}**.

Before we apply Algorithm 6.2.1, which requires rules to be in extended term-restricted form, we note that the rules (r1) and (r2) are not in extended term-restricted form. Thus, we first transform them into extended term-restricted rules as described in Section 5.2.2. They are:

(r1′): teach(T, M, Cno) ← dept(M, Cno, CHrs), (T = allen), (M = math)
(r2′): teach(T, C, Cno) ← dept(C, Cno, CHrs), (T = baker), (C = csc)

Then, the SLD-RC tree for the query **teacher_of(gray, t0)?** is:

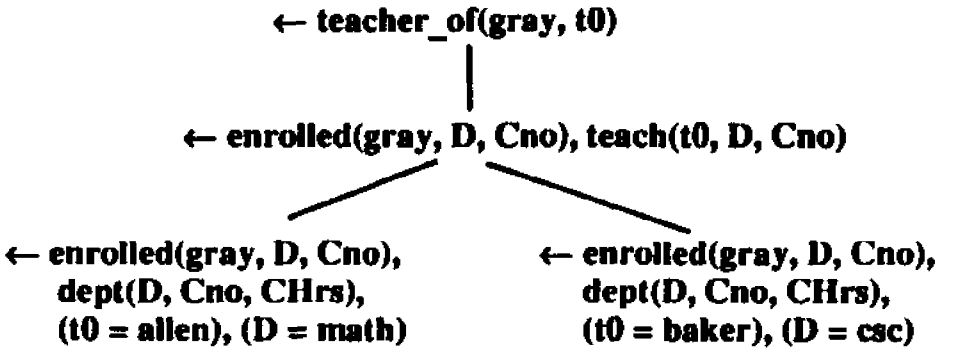


Figure 7.3.1 An SLD-RC tree with the incompleteness of an IDB for a query *teacher_of(gray, T) ?* in DEPARTMENT database

Then, we have two last resolvents; and thus we have two candidates for meaningful intensional answers. They are:

∃Cno ∃CHrs enrolled(gray, math, Cno), dept(math, Cno, CHrs), (T = allen)
∃Cno ∃CHrs enrolled(gray, csc, Cno), dept(csc, Cno, CHrs), (T = baker)

To check whether these candidates are query-constant-dependent (QCD) meaningful intensional answers (Section 5.6.3), we access the EDB to check whether *Gray* is actually enrolled in *math* and *csc* courses. In the relation *enrolled* in Appendix C, *Gray* is actually enrolled in *math* and *csc* courses. Thus, we have two meaningful intensional answers in a set $ANS_I^{mf}(Q)$ as follows:

$ans_I^1(T)$ = ∃Cno ∃CHrs enrolled(gray, math, Cno), dept(math, Cno, CHrs), (T = allen)
$ans_I^2(T)$ = ∃Cno ∃CHrs enrolled(gray, csc, Cno), dept(csc, Cno, CHrs), (T = baker)

Now we compute **$ext(ANS_I^{mf}(Q))$**. Since **$ext(ans_I^1)$ = {allen}** and **$ext(ans_I^2)$ = {baker}**, **$ext(ANS_I^{mf}(Q))$ = {allen, baker}**. Thus, clearly, an extensional answer **{cook}** is in the set $ANS_E'(Q)$, but it is not generated by the evaluation of

intensional answers. Thus the IDB in this case is incomplete to the query **teacher_of(gray, T)?**.

Example 7.3.2:

We illustrate a locally complete IDB to a given query using the same IDB as in Example 7.3.1. Suppose we have a **Q(S) = teacher_of(S, baker)**, asking *who are Baker's students ?* After examining the EDB and IDB in Appendix C, we see that the set $ANS_E'(Q)$ of extensional answers for the query Q(S) is, **{gray, haas}**. Then SLD-RC tree for the query **teacher_of(S, baker)?** is:

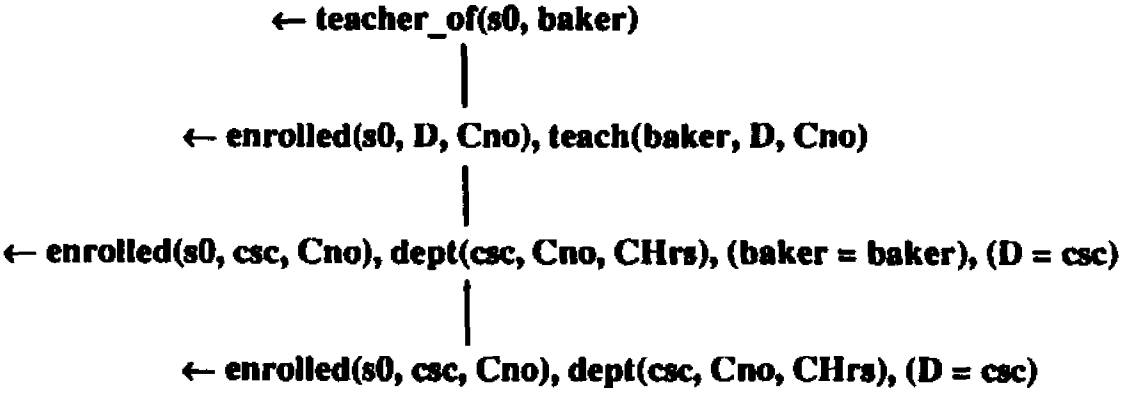


Figure 7.3.2 An SLD-RC tree with the local completeness of an IDB for the query *teacher_of(S, baker)* in DEPARTMENT database

Here we have only one last resolvent; thus we have only one candidate for a meaningful intensional answer. However, no subformula in the candidate can be evaluable against the EDB. Thus, we have an intensional answer for the query *teacher_of(S, baker)* as follows:

$$ans_I^1(S) = \exists Cno\ \exists CHrs\ enrolled(S, csc, Cno), dept(csc, Cno, CHrs)$$

Note that this answer is interpreted as *if a student* S *is enrolled in Course* Cno *of* csc *department, then* S *is a Baker's student*. Now we compute $\mathbf{ext(ANS_I^{mf}(Q))}$. Since $\mathbf{ext(ans_I^1) = \{gray, haas\}}$, $\mathbf{ext(ANS_I^{mf}(Q)) = \{gray, haas\}}$. Thus, $\mathbf{ANS_E'(Q) = ext(ANS_I^{mf}(Q))}$. Thus the IDB in this case is locally complete to the query **teacher_of(S, baker)?**. But this IDB is not globally complete, since the IDB was incomplete for a query **teacher_of(gray, T)?** in Example 7.3.1.

Discussion of Example 7.3.1 and Example 7.3.2

In the following, we discuss the database used in these examples. The database in Example 7.3.1 satisfies the unique intensional literal assumption stating that a literal must not be defined in both EDB and IDB. This assumption, discussed in Section 5.2.1, prevents the derivation of two different trees for a given query. Note that, however, in our example the literals having the same semantics have been defined both in the EDB as *teaches* and in the IDB as *teach*. The reason for this can be explained as follows: In rule (r1), *Allen* is teaching all courses in the department of Mathematics. Suppose the department has total 20 courses. If we want to add these facts into the EDB, we need to store 20 tuples to the relation *teaches*. However, these facts can be abstracted into one rule (r1) as in the example database. However, since our unique intensional literal assumption does not allow us to use the same literal name, we use different literal name *teach* in the IDB from that *teaches* in the EDB. Thus, having two different literal names, one in the EDB and another in the IDB, with the same semantics can be justified.

However, the IDB of the example does not contain any rule which connects the EDB and the IDB. That is, the IDB defines the relationship *teach incompletely*. If the IDB is very complex or contains many rules, this kind of problem may result. Thus, we need to add one rule to the IDB that connects the semantics of the EDB with that of the IDB as follows:

teach(T, D, Cno) ← teaches(T, D, Cno)

Now the complete IDB of the example database looks as follows:

(r1): teach(allen, math, Cno) ← dept(math, Cno, CHrs)
(r2): teach(baker, csc, Cno) ← dept(csc, Cno, CHrs)
(r3): teacher_of(S, T) ← enrolled(S, D, Cno), teach(T, D, Cno)
(r4): teach(T, D, Cno) ← teaches(T, D, Cno)

We interpret these rules in relation to *teaches* and *teach* as follows: *For the teaching information, we first check rule (r1) and (r2), since they abstract teaching information related to Allen* and *Baker. Then all other teaching information is referred to the EDB via rule (r4).* For this reason, we can call rule (r4) a *catch-all* rule. Without rule (r4), the same semantics of teaching information in the EDB and the IDB is not connected. Thus, the general rule is that when the same semantics is distributed over both an EDB and an IDB, there must be a catch-all rule which connects the EDB and the IDB. This idea also agrees to the transformation rule of the unique intensional literal assumption.

Since we are assuming that there is a method that can compute all extensional answers by checking the semantics of the database and the given query, we agree that our approach is not completely general, and theoretical rather than practical. However, we believe that it will be a first step to research on the notions of completeness and

incompleteness of an IDB. However, we believe that this idea can be useful in the design of deductive databases, and more specifically, in the design of an IDB which correctly connects and represents the semantics of an EDB.

7.3.3. Transformation into Complete IDB

In the previous section, we have shown that the idea of intensional answers can be used to relate the notion of the completeness of an IDB to a given query.

In this section, we show that, theoretically, all incomplete IDBs can be converted into complete IDBs as stated below.

Theorem 7.3.1:

Let the completeness of an IDB be defined as in Definition 7.3.1. Then all incomplete IDBs can be transformed into globally complete IDBs.

Proof:

In the worst case, we convert all facts in an EDB into rules. Let $r(a_1, \cdots, a_n)$ be a tuple in an arbitrary EDB-defined relation r, where a_i is a constant. Then we convert the tuple into the following rule.

$$r(X_1, \cdots, X_n) \leftarrow (X_1 = a_1), \cdots, (X_n = a_n)$$

By this transformation, resolution can clearly be connected with all tuples in an EDB for any given query. Hence, the new IDB is globally complete. The proof of equivalence for this transformation is similar to the proof of Theorem 5.2.1. ■

Converting all EDB tuples into rules,* however, is very inefficient and impractical. Checking the completeness of an IDB and the efficient transformation of an incomplete IDB into a complete IDB should be studied further. We illustrate the modification of an incomplete IDB shown in Example 7.3.1 into a complete IDB below:

Example 7.3.3:

After adding the catch-all rule discussed in the previous section, we use the same query and databases as in Example 7.3.1.

Query: *who are Gray's teachers?* **Q(T) = teacher_of(gray, T)?**

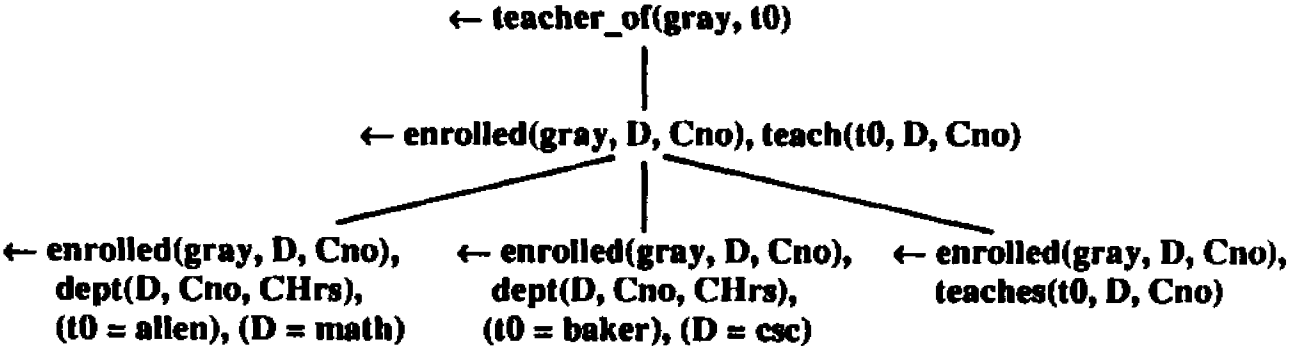


Figure 7.3.3 An SLD-RC tree with complete IDB in the DEPARTMENT database

The meaningful intensional answers are:

$ans_I^1(T)$ = **∃Cno ∃CHrs enrolled(gray, math, Cno), dept(math, Cno, CHrs), (T = allen)**

$ans_I^2(T)$ = **∃Cno ∃CHrs enrolled(gray, csc, Cno), dept(csc, Cno, CHrs), (T = baker)**

---

* Note that Cholvy and Demolombe also use this transformation to convert a fact into a rule. The purpose of their transformation is to obtain intensional answers even when a query consists only of extensional literals. However, their approach is not consistent because they use these rules, converted from the facts, only for the resolution of extensional literals in a query, but they do not use those rules in other cases. In our approach, we do not attempt to resolve the query that consists only of extensional literals. Furthermore, we stop resolution when a resolvent consists only of non-intensional literals.

$\mathbf{ans_I^3(T) = \exists D\ \exists Cno\ enrolled(gray, D, Cno), teaches(T, D, Cno)}$

Now we compute $\mathbf{ext(ANS_I^{mf}(Q))}$. Since $\mathbf{ext(ans_I^1) = \{allen\}}$, $\mathbf{ext(ans_I^2) = \{baker\}}$, and $\mathbf{ext(ans_I^3) = \{cook\}}$, $\mathbf{ext(ANS_I^{mf}(Q)) = \{allen, baker, cook\}}$. Thus, $\mathbf{ANS_E'(Q) = ext(ANS_I^{mf}(Q))}$. The IDB in this case is locally complete to the query **teacher_of(gray, T)?**. It is easy to see that the new IDB is also locally complete for the query **teacher_of(S, baker)?**

CHAPTER 8

IQP WITH RECURSIVE RULES

8.1. Recursive Query Processing

Recursive query processing is answering queries on relations defined by recursive Horn clauses. Recursive query processing in deductive database systems has received a lot of attentions recently [Hens84, Ullm85, Lozi85, Viei86, Yoko86, Han86, Banc86b, Balb87, Sacc87]. Among them, Ullamn [Ullm85] provides a basic framework for the understanding of recursive query processing strategies such as top-down, bottom-up, and sideway rules using rule/goal graph formalism. Han [Han86] provides an extensive classification of the recursive rules and several algorithms to process recursive queries. Bancilhon and Ramakrishnan [Banc86b] survey and compare some known strategies for the processing of recursive queries. Even though many different strategies in processing the recursive queries in DDBs have been proposed, it is still believed that there is no complete general scheme, yet.

The problem of recursive query processing is in determining the termination of applying recursive rules. Query processing algorithms have difficulty in determining the termination of recursion, because it cannot tell when complete answers have been found. That is, for a given recursive query, an intensional processor usually controls the recursivity in the rules, but only a DBMS is able to know when the recursion must stop, based on the facts in an EDB.

8.2. IQP with Recursive Rules

One of the drawbacks of IQP is its inherent difficulty in handling recursive rules. The source of difficulty comes from the fact that the termination of recursive rules depend on the factual data stored in an EDB and the termination of recursion cannot be determined by rules in an IDB alone. That is, queries are resolved with rules in an IDB and stop resolution only when a resolvent consists of non-intensional literals or when it cannot be further resolved. Since the data contained in an EDB are not accessed during resolution, the termination of recursion is actually impossible in our approach. Note that accessing an EDB, in IQP, was limited just to check whether intensional answers derived are contradictory or tautological against the EDB. Since we want to derive answers as a set of non-ground first-order logic formulas, we do not instantiate any variables in intensional answers.

We illustrate the difficulty of IQP with recursive rules using an example taken from [Chan78].

Example 8.2.1:

EDB Schema:

emp(Name, Salary, Mgr, Dept)
sales(Dept, Item)

IDB:
(r1):command(Sup, Sub) ← emp(Sub, Sal, Sup, Dept)
(r2): command(Sup1, Sub1) ← emp(Sub1, Sal1, Name1, Dept1), command(Sup1, Sub1)

The EDB contains two relations; the relation *emp* stores employee's *name, salary, manager,* and *department.* The relation *sales* stores *department name* and *item* that the department is selling. The IDB contains two rules which describe the command

hierarchy in the company. Suppose we have a query *find all employees who can be commanded by John and who sell gun.* The query can be represented by the formula **Q(X) = command(john, X), emp(X, Sal, Mgr, Dept), sales(Dept, gun).** Then the SLD-RC tree for this query is:

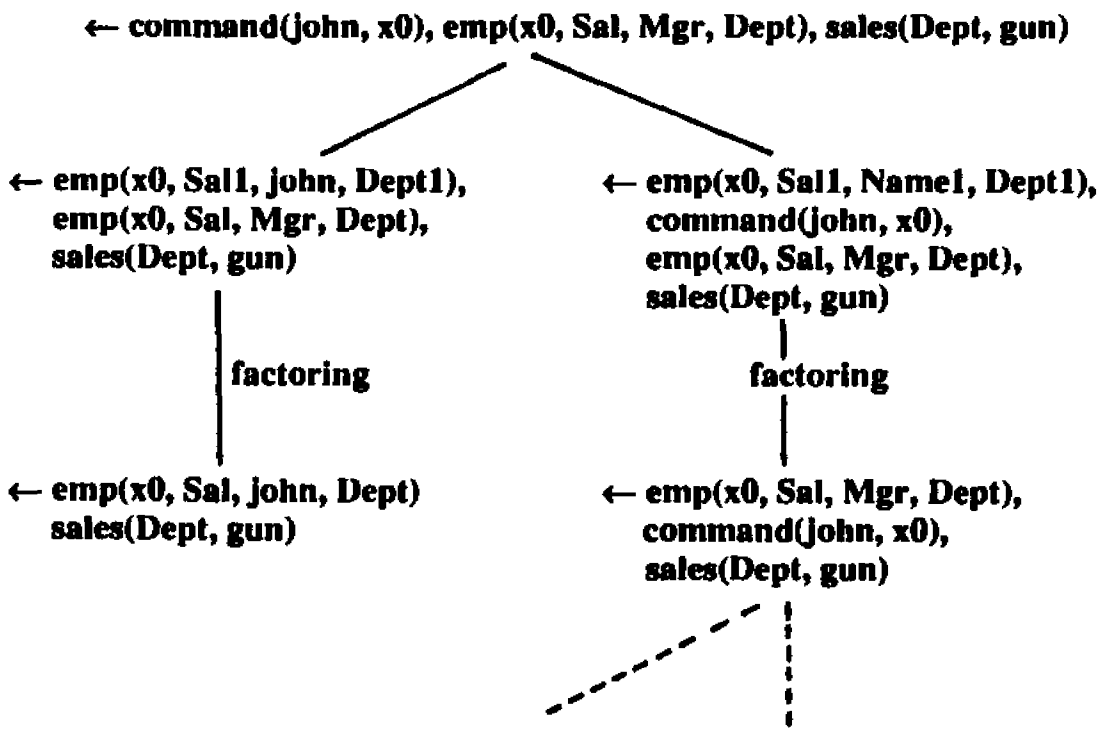


Figure 8.2.1 An SLD-RC tree with factoring for recursive rules

Note that, in Figure 8.2.1, the last node in the second branch has the same number of literals as the root node and looks similar to the root node except their different variable names. Whenever we perform resolution one time, we will have **emp** one more time in a resolvent. However, as long as we perform factoring, we will continuously have a resolvent which is similar to the previous resolvent. Note that the last node in the second branch still contains an intensional literal **command**. Since the sequence of resolution can only be terminated by accessing an EDB, resolution with

rules in an IDB can never be terminated. Thus, we can never have a resolvent which consists only of non-intensional literals.

Note that the problem of repeated extensional literals, in conventional query processing systems, is solved by accessing the data in an EDB. If an extensional literal in a resolvent is false under the EDB, then the branch becomes a failure branch and the branch together with the resolvent is discarded. If the literal is true under the EDB, then the literal is removed from the resolvent after applying the unification to the resolvent.

8.3. Using Closure Literals

In previous section, we have shown that performing factoring to the resolvent makes a new resolvent look similar to previous resolvents. However, without factoring, the resolution will generate resolvents that the same extensional literal is added to the previous resolvent as shown in Figure 8.2.2.

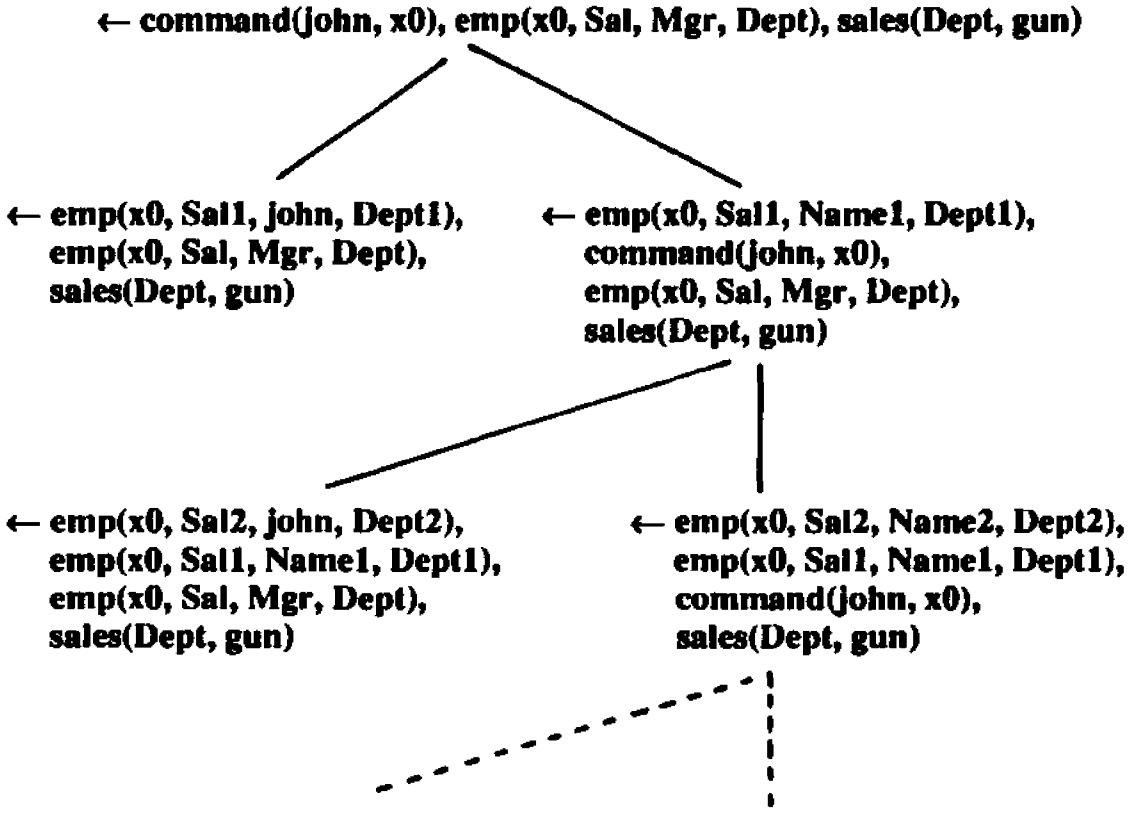


Figure 8.2.2 An SLD-RC tree without factoring for recursive rules

One way to summarize the series of the same literal is using a *closure literal* which is similar to the notion of Kleene closure [Hopc79] in automata theory. For example, the last node in the second branch in Figure 8.2.2 can be written as

← emp^{+}(x0, Sal, Name2, Dept1), command(john, x0), sales(Dept, gun)

However, this notation actually does not provide any additional information than the query formula itself. The new notation still contains an intensional literal and looks similar to the query formula. Also in order to evaluate this formula against the EDB, we still need recursive query processing strategies. Thus, recursion is expected to be a major source of difficulty for IQP.

CHAPTER 9
CONCLUSIONS

9.1. Summary

We have addressed the problem of deriving a set of non-ground first-order logic formulas (intensional answers), as an answer set to a given query, rather than a set of facts (extensional answers), in deductive database (DDB) systems based on non-recursive Horn clauses. The intensional queries have been defined as a type of queries whose answers can be represented by a set of intensional answers.

A strategy in previous work in this area was to use resolution to derive intensional answers. It left, however, several important problems. Some of them are: no specific resolution strategy was given; no specific methodologies to formalize meaningful intensional answers were given; no solution was given to handle a large number of facts in an extensional database (EDB); and no strategy was given to avoid deriving meaningless intensional answers. Furthermore, no relationships between intensional answers and extensional answers were studied.

As a solution, we have proposed a three-stage formalization process (pre-resolution, resolution, and post-resolution), which can solve all the problems mentioned above, for the derivation of meaningful intensional answers.

In the pre-resolution stage, we take two main actions - rule transformations and the identification of relevant literals and relevant clauses. Rule transformations include the transformation of rules into the unique intensional literals and the

extended term-restricted rules. The notions of relevant literals and relevant clauses to a given query have been introduced to reduce a set of clauses to those that are necessary only to derive a set of meaningful intensional answers. These notions are necessary when an IDB contains rules whose heads are comparison literals.

During resolution, factoring is performed to the resolvent and only relevant clauses are chosen as introduced clauses for resolution. Any evaluable comparison literals in a resolvent are evaluated to simplify the resolvent.

In the post-resolution stage, only the last resolvents are taken as candidate formulas for intensional answers and the candidate formulas are checked against an EDB to remove any contradictory and tautological formulas.

We have introduced SLD-RC resolution which derives a set of meaningful intensional answers by incorporating the result of three-stage formalization process. We have proved several important properties of SLD-RC resolution including the finiteness of SLD-RC resolution, the soundness and the completeness of SLD-RC resolution for intensional query processing (IQP). We have also presented an algorithm which implements SLD-RC resolution with the three-stage formalization process and proved the correctness of the algorithm.

The logical relationship between intensional answers and a given query should be defined in such a way that intensional answers are necessary and sufficient conditions for the query formula. However, we have showed that, when we use resolution, intensional answers can be defined as only sufficient conditions for a given query.

Incorporating IQP systems into conventional database systems requires also the identification of the relationships between intensional answers and extensional answers. We have identified two relationships between them. One relationship, called a syntactic relationship, is based on our conventional definitions of answers which are based on a particular computational methodology, resolution. The syntactic relationship implies that, when we use resolution to compute answers, intensional answers are only sufficient conditions to derive all extensional answers. The important implication of this theorem is that, for some databases with a given query, some extensional answers might not be generated by evaluating all the intensional answers against an EDB.

Another relationship, called a semantic relationship, is independent of any particular computational methodology and based on the logical definitions of answers. The semantic relationship implies that we can derive all extensional answers by evaluating all intensional answers against an EDB and those extensional answers are only correct answers that satisfy the query.

Based on these two relationships and on the assumption that the facts in an EDB are complete and correct, we have discussed the global and local completeness of an IDB to a given query. We have also proved that, theoretically, every incomplete IDB can be transformed into a globally complete IDB, in which all the extensional answers can be generated by evaluating all intensional answers against an EDB.

9.2. Advantages of IQP

We think the functionality of DDBs can be greatly enhanced by incorporating the

notion of intensional answers to a query. We also believe that there are many practical situations which intensional answers could be useful. Some advantages of deriving intensional answers are :

(1) Intensional answers are represented in terms of the general rules of the database which are independent of any particular database state, even though they are query-constant dependent.

(2) Intensional answers tell us what conditions or formulas constitute extensional answers. Thus, intensional answers can help interpret extensional answers, providing intelligent user interface.

(3) When the number of extensional answers is large and they can be represented by first-order logic formulas, the corresponding intensional answers are compact and clear enough, unless users are interested in all the extensional answers. If users are interested in specific factual answers (extensional answers), they can request the system to evaluate the intensional answers to generate the extensional answers.

(4) When intensional answers can be derived without accessing an EDB or when users are satisfied with intensional answers, IQP has a computational advantage.

(5) IQP is a new methodology for query processing in DDBs. It treats rules as data and it can change the way we view the answer. Thus, IQP extends the category of queries. This advantage is discussed in more detail in next section.

9.3. Contributions

We claim that IQP provides a new methodology for query processing in DDBs and will greatly increase our insight into the nature of DDBs. We summarize the contributions we have made in this work.

(1) We have provided a new methodology for IQP in DDBs. It treats rules as data and it can change the way we view the answer. That is, answers can be a set of formulas, rather than a set of facts as in conventional database systems. In addition, IQP extends the category of queries. For example, IQP allows the queries, that are not available in conventional query processing systems, whose answers need to be conditions, definitions, or formulas that abstract factual answers.

(2) We have proposed a formal framework for IQP based on SLD-RC resolution. We have introduced a three-stage formalization process to derive a set of meaningful intensional answers and built those procedures into SLD-RC resolution. We have also introduced the notions of relevant literals and relevant clauses which are necessary to avoid deriving meaningless intensional answers when we have rules whose heads are comparison literals. In addition, we have shown $L_p \subseteq L_{nr}$ meaning that the set of non-relevant literals to a query is a subset of the set of pure literals to the query, where removing a set of pure literals is a conventional method of reducing a clause set for resolution into a smaller set. Our method can also handle a large number of facts in an EDB without converting all the facts in an EDB into rules. In addition, we have proved several important properties of SLD-RC resolution including the finiteness of

SLD-RC resolution, the soundness and the completeness of SLD-RC resolution for IQP. We have justified using only the leaves of the success branches of an SLD-RC tree as candidates for intensional answers based on the soundness of SLD resolution.

(3) We have shown that, when we use resolution to derive intensional answers, we can define intensional answers as only sufficient conditions, not sufficient and necessary conditions, to a given query. The intensional answers which are logically equivalent to the query cannot be computed using resolution. Similarly, we have also shown that the definition of extensional answers using Green's literal, $\forall X\ (Q(X) \rightarrow ans_E(X))$, is the only definition we can define when we use resolution, even though the formula should be $\forall X\ (Q(X) \leftrightarrow ans_E(X))$.

(4) We have identified two new relationships between intensional answers and extensional answers - a syntactic relationship and a semantic relationship. The syntactic relationship, which is the result of the discussion in (3), implies that, in general, intensional answers are sufficient conditions to derive all extensional answers. That is, when the syntactic relationship holds, we may or may not derive all extensional answers, for a given query, by evaluating all intensional answers against the EDB. However, for a complete IDB where the semantic relationship between two types of answers holds, intensional answers are sufficient and necessary conditions to derive all extensional answers. That is, all the extensional answers can be derived by evaluating all the intensional answers against the EDB.

(5) We have introduced the notions of the global and local completeness of an IDB based on two relationships described in (4), showing that intensional answers can be used to check the completeness of intensional databases (IDBs) for a given query. We have also proved that all incomplete IDBs can be transformed into globally complete IDBs. We claim that the notion of the completeness of an IDB is important for the design of an IDB.

(6) We have identified the requirement of rules for IQP. IQP requires rules to be in extended term-restricted form. Rules also must satisfy the assumption of unique intensional literals under SLD-like resolution. We have proved the theorem that the transformation of non-extended term-restricted rules into extended term-restricted rules is logically equivalent.

9.4. Limitations and Future Research

In this section, we discuss a wide variety of research issues related to this work, in addition to several limitations of our research result.

An Integrated System

One of the immediate practical concern related to IQP is to construct an integrated query processing system which incorporates an IQP system with a conventional database system. Issues involved are avoiding redundant access to an EDB, a compiler for intensional query processor, a high-level user interface, a natural language interpreter for intensional answers, efficient evaluation of intensional answers against an EDB, an efficient interface with conventional query processor, etc. We believe that a great deal could be learned by building an IQP system on a real-

world application.

Application Domain

It is also important to identify the application domains which intensional query processing is useful. Some interesting fields for the application of intensional answers are database consistency checking, expert systems, planning, rule learning, etc.

Recursive Rules

One of the inherent difficulty for IQP is in handling recursive rules. The real power of deductive database systems over conventional database systems is the ability of deductive databases to define recursive rules. However, since the termination of recursion depends on the factual data stored in an EDB, recursion is expected to be a major source of difficulty for IQP.

IQP in Non-Horn Clauses

We have assumed that database consists of non-recursive Horn clauses. Using non-Horn clauses in deductive databases is not well-developed yet [Mink82, Hens86, Przy86J. It will be nontheless interesting to study the IQP in the context of non-Horn clauses. For example, are the definitions of extensional answers and intensional answers, given by (3.2.4) and (3.3.1), still valid ? Is it possible to compute the logical definitions of extensional answers and intensional answers, given by (7.1.1) and (7.1.5), in the non-Horn clause systems ? What are the relationships between two types of answers in non-Horn clause systems ?

Negation and IQP

Even though we have used the negation symbol in the body of a rule, we have not considered the effect of negation in IQP. We think that the effect of negation to IQP must be investigated in detail. For example, in the proofs of Theorem 7.2.1 and 7.2.3, if we consider the assumption of negation as failure, we need to use the completed forms of the axioms [Clar78, Gene88].

Inference Rules

It will be also worth trying other inference rules, instead of SLD-like resolution, such as term-rewriting rules [Chan78, Chan79] or model elimination [Love78, Stic84, Stic86]. In addition, it will be worth investigating what computational methodology allows us to use the logical definition of intensional answers and extensional answers given in Definition (7.1.1) and (7.1.5).

Completeness of an IDB

We have addressed the global completeness of an IDB and the local completeness of an IDB for a given query. We think that we need more detailed studies on the completeness of IDBs such as the classification of incomplete cases, checking the incompleteness, and the transformation of incomplete IDBs into complete IDBs.

IQP and Semantic Data Models

Another interesting research is to study the relationships between IQP and semantic data models such as Entity-Relationship model [Chen76]. For example, what are the roles of semantic data models for IQP ? What are the roles of IQP for

semantic data models ? How can the knowledge representation embedded in semantic data models help IQP ?

Hybrid answers that consist of both formulas and facts

Further generalizing Imielinski's [Imie87] idea is also considered a significant work. That is, how to, in general, efficiently derive answers that consist of both intensional answers and extensional answers ? By incorporating this idea, users can have three types of answers to a given query - a set of facts, a set of formulas, or facts and formulas. In order to derive answers according to Imielinski's approach, not only must the database contain such rules, which can be embedded in answers, but a query processor must also be able to recognize such rules. Thus, a more general approach which can derive any form of answers, among three types, is highly desirable.

Long Intensional Answers

One of the problems in our research result is to represent intensional answers solely in terms of non-intensional literals, which are extensional literals or comparison literals. However, if rules are defined in terms of many extensional literals, then intensional answers will consist of many extensional literals, which are not easily interpretable. In this case, we need some other methods to represent our intensional answers in a more abstract form. One way to solve this problem is to allow intensional literals to appear in intensional answers. However, this approach involves the problems such as what intensional literals should be allowed and what intensional literals should not, in addition to the problem of including intermediate resolvents during resolution. Furthermore, evaluating this type of intensional answers against an EDB takes another inference procedure to transform the intensional

answers into those that consist only of non-intensional literals. A promising approach is to allow the user to define a new rule based on the answers in this case. We think that this idea can also be applied to the rule learning of an IDB.

BIBLIOGRAPHY

[Asir85] P. Asirelli, M. D. Santis, and M. Martelli, "Integrity Constraints in Logic Databases," *Journal of Logic Programming*, vol. 3, pp. 221-232, 1985.

[Balb87] I. Balbin and K. Ramaohanarao, "A Generalization of the Differential Approach to Recursive Query Evaluation," *J. of Logic Programming*, pp. 259-262, 1987.

[Banc86a] F. Bancilhon and R. Ramakrishnan, "An Amateur's Introduction to Recursive Query Processing Strategies," in *Proceedings of ACM SIGMOD* , ed. Carlo Zaniolo, pp. 16-52, Washington, DC, May 28-30, 1986.

[Banc86b] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman, "Magic Sets and Other Strange Ways to Implement Logic Program," *Proc. 5th ACM SIGMOD-SIGACT Symp. on PODS*, 1986.

[Bocc86] J. Bocca, "On the Evaluation of EDUCE," in *Proc. of ACM SIGMOD*, ed. C. Zaniolo, vol. 15, pp. 368-378, June 1986.

[Brod84] M. Brodie and M. Jarke, "On Integrating Logic Programming and Databases," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 40-62, Kiawah Island, South Carolina, October 24-27 1984.

[Brod86] M. L. Brodie, R. Balzer, and G. Wiederhold etc., "Knowledge Base Management Systems: Discussions from the Working Group," in *Expert*

Database Systems, ed. L. Kerschberg, pp. 19-32, 1986.

[Ceri86] S. Ceri, G. Gottlob, and G. Wiederhold, "Interfacing Relational Databases and Prolog Efficiently," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. L. Kerschberg, pp. 141-154, Charleston, South Carolina, April 1-4, 1986.

[Chak84] U. S. Chakravarthy, D. Fishman, and J. Minker, "Semantic Query Optimization in Expert Systems and Database Systems," in *Proceedings of Workshop on Expert Database Systems*, ed. L. Kerschberg, pp. 326-341, Kiawah Island, South Carolina, October 24-27, 1984.

[Chak86a] U. S. Chakravarthy, J. Minker, and J. Grant, "Semantic Query Optimization: Additional Constraints and Control Strategies," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. Larry Kerschberg, pp. 259-270, Charleston, South Carolina, April 1-4, 1986.

[Chak86b] U. S. Chakravarthy, J. Grant, and J. Minker, "Foundations of Semantic Query Optimization for Deductive Databases," in *Foundations of Deductive Databases and Logic Programming*, ed. Jack Minker, pp. 67-101, August, 1986.

[Chan73] C. L. Chang and R. C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[Chan78] C. L. Chang, "DEDUCE 2: Further Investigations of Deduction in Relational Databases," in *Logic and Databases*, ed. J. Minker, pp. 201-

236, Plenum Press, New York, 1978.

[Chan79] C. L. Chang and J. R. Slagle, "Using Rewriting Rules for Connection Graphs to Prove Theorems," *Artificial Intelligence*, pp. 159-180, North-Holland, 12, 1979.

[Chan84] C.L. Chang and A. Walker, "PROSQL: A Prolog Programming Interface with SQL/DS," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 378-396, Kiawah Island, South Carolina, October 24-27, 1984.

[Chen76] P. P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM TODS*, vol. 1, No.1, pp. 9-36, March, 1976.

[Chol86] L. Cholvy and R. Demolombe, "Querying a Rule Base," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. L. Kerschberg, pp. 365-371, Charleston, South Carolina, April 1-4, 1986.

[Chom83] J. Chomicki and W. Grudzinski, "A Database Support System for Prolog," *Proc. of Logic Programming Workshop*, University of Lisbon, Lisbon, Albufeira, Portugal, 1983.

[Clar78] K. L. Clark, "Negation as Failure," in *Logic and Databases*, ed. Gallier, Minker, and Nicolas, pp. 293-322, Plenum Press, New York, 1978.

[Deck86] H. Decker, "Integrity Enforcement on Deductive Database Systems," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. L. Kerschberg, pp. 271-285, Charleston, South Carolina, April 1-4, 1986.

[Ende72] H. B. Enderton, *A Mathematical Introduction to Logic*, Academic Press,

1972.

[Frey87] J.C. Freytag, "A Rule-Based View of Query Optimization," in *Proceedings of ACM SIGMOD* , ed. I. Traiger, vol. 16, pp. 173-180, December 1987.

[Gabb84] D. M. Gabbay and U. Reyle, "N-Prolog: An Extension of Prolog with Hypothetical Implications. I.," *J. of Logic Programming*, vol. 1, no. 4, pp. 319-355, December 1984.

[Gall78a] H. Gallaire, J. Minker, and J. Nicolas, (Eds)., *Logic and Databases*, Plenum Press, New York, 1978.

[Gall78b] H. Gallaire, J. Minker, and J. Nicolas, "An Overview and introduction to Logic and Databases," in *Logic and Databases*, pp. 3-30, 1978.

[Gall84] H. Gallaire, J. Minker, and J. Nicolas, "Logic and Databases: A Deductive Approach," *Computing Survey*, vol. 16, no. 2, pp. 153-185, 1984.

[Gall87] J. H. Gallier and S. Raatz, "HORNLOG: A Graph-Based Interpreter for General Horn Clauses," *Journal of Logic Programming*, vol. 4, no. 2, pp. 119-155, June, 1987.

[Gene87] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[Gett84] J. Getta and H. Rybinski, "A deduction augmented Database Management System," *Information Systems*, vol. 9, no. 2, pp. 167-179,

1984.

[Grae87] G. Graefe and D. J. DeWitt, "The EXODUS Optimizer Generator," in *Proceedings of ACM SIGMOD* , ed. I. Traiger, vol. 16, pp. 160-172, December 1987.

[Gree69] C. Green, "Application of Theorem Proving to Problem Solving," *Proceedings of First IJCAI*, pp. 219-239, Morgan Kaufmann, Washington, DC, 1969.

[Hamm80] M. M. Hammer and S. B. Zdonik, "Knowledge Based Query Processing," *Proc. of Sixth Int'l Conf. on VLDB*, pp. 137-147, Sept. 1980.

[Han86] J. Han and H. Lu, "Some Performance Results on Recursive Query Processing in Relational Database Systems," *Proceedings of Int'l Conference on Data Engineering*, pp. 533-539, Los Angeles, January 1986.

[Hens86] L. Henschen and H. Park, "Compiling the GCWA in indefinite databases," in *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, pp. 193-241, Washington, DC, August 1986.

[Hens84] L. J. Henschen and S. A. Naqvi, "On Compiling Queries in Recursive First-Order Databases," *JACM*, vol. 31, No. 1, pp. 47-85, January 1984.

[Hopc79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language, and Computation*, Addison-Wesley, Reading, MA, 1979.

[Imie87] T. Imielinski, "Intelligent Query Answering in Rule Based Systems," *J.*

of Logic Programming, vol. 4, no. 3, pp. 229-258, September 1987.

[Ioan84] Y. Ioannidis, L D. Shinkle, and E. Wong, "Enhancing INGRES with Deductive Power," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 847-850, Kiawah Island, South Carolina, October 24-27, 1984.

[Kell78] C. Kellogg, P. Klahr, and L. Travis, "Deductive Planning and Pathfinding for Relational Databases," in *Logic and Databases*, ed. J. Minker, pp. 179-200, Plenum Press, New York, 1978.

[Kell84] C. Kellogg, "The Transition from Data Management to Knowledge Management," *Data Engineering*, pp. 467-472, IEEE, 1984.

[Kell86] C. Kellogg, A. O'llare, and L. Travis, "Optimizing the Rule-Data Interface in a KMS," in *12th VLDB*, ed. S. Ohsuga, pp. 42-51, Kyoto,Japan, August 25-28, 1986.

[Kers84] L. Kerschberg, *Proceedings of the First Int'l Workshop on Expert Database Systems*, Kiawah Island, South Carolina, October 24-27, 1984.

[Kers85] L. Kerschberg, "Expert Database Systems (Workshop Review)," *Proceedings of ACM SIGMOD*, pp. 414-417, 1985.

[Kers86] L. Kerschberg, *Proceedings of the First Int'l Conference on Expert Database Systems*, Charleston, South Carolina, April 1-4, 1986.

[King81] J. J. King, "Query Optimization by Semantic Reasoning," *PhD Thesis*, Dept. of Computer Science, Stanford University, May 1981.

[Kowa71] R. Kowalski and D. Kuehner, "Linear Resolution with Selection Function," *Artificial Intelligence*, vol. 2, no. 3/4, pp. 227-260, 1971.

[Kowa79] R. Kowalski, *Logic for Problem Solving*, Elsevier North-Holland, New York, 1979.

[Lifs86] V. Lifschitz, "On the Declarative Semantics of Logic Programs with Negation," in *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, pp. 420-432, Washington, DC, August 1986.

[Lloy84a] J. W. Lloyd and R. W. Topor, "Making Prolog More Expressive," *Journal of Logic Programming*, vol. 1, no. 3, pp. 225-240, 1984.

[Lloy84b] J. W. Lloyd, *Foundations of Logic Programming*, Springer Verlag, 1984.

[Lloy85] J. W. Lloyd and R. W. Topor, "A Basis for Deductive Database Systems," *Journal of Logic Programming*, vol. 2, no. 2, pp. 93-109, 1985.

[Lloy86] J. W. Lloyd and R. W. Topor, "A Basis for Deductive Database Systems II," *Journal of Logic Programming*, vol. 3, no. 1, pp. 55-67, 1986.

[Love78] D. W. Loveland, *Automated Theorem Proving: A Logical Basis*, North-Holland Publishing Company, 1978.

[Lozi84] E. L. Lozinskii, "Deduction in Relational Databases Directed by Problem-Specific Data," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 694-, Kiawah Island, South Carolina, October 24-27, 1984.

[Lozi85] E. L. Lozinskii, "Evaluating Queries in Deductive Databases by Generating," *11th Proc. of IJCAI*, pp. 173-177, August 1985.

[Lozi86] E. L. Lozinskii, "A Problem-Oriented Inferential Database System," *ACM, TODS*, vol. 11, no. 3, pp. 323-356, Sept., 1986.

[Luca85] R. Lucas, "Expert System Modeling using a Logic Language and a Relational Database," *4th Int'l Conf. on System Eng.*, pp. 291-298, Coventry, England, September 10-12, 1985.

[Luck71] D. Luckham and N.J. Nilsson, "Extracting Information from Resolution Proof Trees," *Artificial Intelligence*, vol. 2, no. 1, pp. 25-54, 1971.

[Maie88] D. Maier and D. S. Warren, *Computing with Logic: Logic Programming with Prolog*, The Benjamin/Cummings Publishing Company, 1988.

[Mink78] J. Minker, "An Experimental Relational Database Management Systems based on Logic," in *Logic and Databases*, ed. Gallaire and Minker, pp. 107-147, 1978.

[Mink82] J. Minker, "An Indefinite Databases and the Closed World Assumptions," *Proceedings of the 6th Conf. on Automated Deduction*, vol. 138, pp. 292-308, Springer-Verlag, 1982.

[Mink86] J. Minker, *Proceedings of Foundations of Deductive Database Systems and Logic Programming*, Washington, DC, August 18-22, 1986.

[Miss84] M. Missikoff and G. Wiederhold, "Towards a Unified Approach for Expert and database Systems," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 186-206, 1984.

[Naqv86] S. A. Naqvi, "Negative Queries in Horn Databases," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. L. Kerschberg, pp. 157-166, Charleston, South Carolina, April 1-4, 1986.

[Nico82] J. Nicolas, "Logic for Improving Integrity Checking in Relational Databases," *Acta Informatica*, vol. 18, pp. 227-253, 1982.

[Nico78] J. M. Nicolas and H. Gallaire, "Data Base: Theory vs. Interpretation," in *Logic and Databases*, ed. J. Minker, pp. 33-54, Plenum Press, New York, 1978.

[Nils80] N.J. Nilsson, *Principles of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1980.

[Park86] D. Stott Parker and M. Carey, etc, "Logic Programming and Database Systems," in *Expert Database Systems*, ed. L. Kerschberg, pp. 35-48, Benjamin/Cummings Publ. Company, 1986.

[Przy86] Teodor C. Przymusiski, "On the Semantics of Stratified Deductive Databases," in *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, pp. 433-443, Washington, DC, August 1986.

[Raji86] M. R. Rajinikanth and P. K. Bose, "A semantic and Logical Front-end to a Database system," *ISMIS*, pp. 103-111, 1986.

[Reit78a] R. Reiter, "Deductive Question Answering on Relational Data Bases," in *Logic and Databases*, ed. Gallier, Minker, and Nicolas, pp. 149-177, Plenum Press, New York, 1978.

[Reit78b] R. Reiter, "On Closed World Databases," in *Logic and Databases*, ed. Gallier, Minker, and Nicolas, pp. 55-76, Plenum Press, New York, 1978.

[Reit83] R. Reiter, "Introduction and Overview," *IJCAI*, pp. 1119-1200, 1983.

[Reit84] R. Reiter, "Towards a Logical Reconstruction of Relational Database Theory," in *On Conceptual Modeling*, pp. 191-234, Springer-Verlag, New York, 1984.

[Robi69] G. Robinson and L. Wos, "Paramodulation and Theorem-Proving in First-Order Theories with Equality," in *Machine Intelligence 4*, ed. Meltzer and Michie, pp. 135-150, Edinburgh University Press, 1969.

[Robi65] J. A. Robinson, "A Machine Oriented Logic Based on the Resolution Principle," *JACM*, vol. 12, pp. 23-41, 1965.

[Sacc86] D. Sacca and C. Zaniolo, "On the Implementation of a Simple Class of Logic Queries for Databases," *PODS*, pp. 16-23, 1986.

[Sacc87] D. Sacca and C. Zaniolo, "Magic Counting Methods," in *Proceedings of ACM SIGMOD* , ed. I. Traiger, vol. 16, pp. 49-59, December 1987.

[Shen87] S. T. Shenoy and Z. M. Ozsoyoglu, "A System for Semantic Query Optimization," in *Proceedings of ACM SIGMOD* , ed. I. Traiger, vol. 16, pp. 181-195, December 1987.

[Smit84] J. Smith, "Expert Database Systems: A Database Perspective," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 1-22, Kiawah Island, South Carolina, October 24-27, 1984.

[Song87] I. Song, "Classification of Query Types in Deductive Database Systems," *Technical Report #87-037*, Computer Science Department, Louisiana State University, 1987.

[Spyr87] N. Spyratos, "The partition Model: a Deductive Database Model," *ACM Trans. Database Syst.*, vol. 12, no. 1, pp. 1-37, March 1987.

[Stic84] M. E. Stickel, "A Prolog Technology Theorem Prover," *New Generation Computing*, vol. 2, no. 4, pp. 371-383, 1984.

[Stic86] M. E. Stickel, "A Prolog Technology Theorem Prover: Implementation By an Extended Prolog Compiler," *CADE-8 8th Int'l Conf. on Automated Deduction*, pp. 573-587, 1986.

[Topo86] R. Topor and E. A. Sonenberg, "On Domain Independent Databases," in *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, pp. 403-419, Washington, DC, August 1986.

[Ullm85] J. D. Ullman, "Implementation of Logical Query Language for Databases," *TODS*, vol. 10, no. 3, pp. 289-321, September 1985.

[Ullm86] J. D. Ullman, "An Approach to Processing Queries in a Logic-Based Query Language," in *Knowledge Base Management System*, ed. John Mylopoulos, pp. 147-164, Springer-Verlag, New York, 1986.

[Viei86] L. Vieille, "Recursive Axioms in Deductive Databases: The Query/Subquery Approach," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. L. Kerschberg, pp. 179-194, Charleston, South Carolina, April 1-4, 1986.

[Warr86] D. S. Warren and S. Manchanda, "Towards a Logical Theory of Database View Updates," in *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, pp. 27-52, Washington, DC, August 1986.

[Yoko86] H. Yokota, K. Sakai, and H. Itoh , "Deductive Database Systems based on Unit Resolution," *2nd Int'l Conf. on Data Engineering*, pp. 228-235., Feb 5-7, 1986.

[Zani84] C. Zaniolo, "Prolog: A Database Query Language For All Seasons," *Proceedings of the First Int'l Workshop on Expert Database Systems*, pp. 63-73, Kiawah Island, South Carolina, October 24-27, 1984.

[Zani86] C. Zaniolo, "Safety and Compilation of Non-Recursive Horn Clauses," in *Proceedings of the First Int'l Conf. on Expert Database Systems*, ed. L. Kerschberg, pp. 167-178 , Charleston, South Carolina, April 1-4, 1986.

APPENDICES

APPENDIX A

CLASSIFICATION OF QUERY TYPES IN DDB SYSTEMS

In this appendix, we elaborate on the types of queries discussed in Section 2.4. More detailed discussion on query types can be found in [Song87].

A.1 Types of Answers

In this category, queries are classified by the form of desired answers - a set of facts or a set of formulas satisfying the query. They can be further classified into five types as follows:

(a) Answers that are facts in an EDB

(b) Answers that are facts not existing in an EDB

(c) Answers that are formulas

(d) Answers that consist of facts and formulas

(e) An answer that is either *yes* or *no*

A.1.1 Answers that are facts in an EDB

Answers to most queries in conventional database or deductive database systems belong to this category. In this category, queries in deductive database systems are often transformed into an easy-to-process, but semantically equivalent, form by ICs or rules in an IDB. However, the answers are eventually retrieved from an EDB. The work by Hammer and Zdonik [Hamm80], King [King81], and Chakravarthy, et. al. [Chak84, Chak86a] are some examples in this category.

A.1.2 Answers that are facts not existing in an EDB

Answers to some queries are not explicitly stored in an EDB, but they are computed, generated or deduced. This category can further be classified into two cases.

The first case is when the answers are computed from aggregate functions such as COUNT, SUM, or AVERAGE.

The second case is those instances when the answers to the queries exist in the rules in IDB. For example, suppose we have a rule as follows:

manager(X, john) ← emp(X, Salary, sales-dept).

The rule states that *the manager of the sales department is John*. A query such as *who is the manager of the sales department* can be simply answered by accessing this rule.

A.1.3 Answers that are formulas

We have discussed this type of query in this dissertation.

A.1.4 Answers that consists of facts and formulas

For this type of query and answer, refer to Section 2.5.2 of this dissertation.

A.1.5 An answer that is either yes or no

If the query does not contain any variables, the answer is simply either *yes* or *no*.

A.2 Free/Bound Information Given in a Query

One of the central problems in query processing in deductive databases is to reduce the search space by searching only for relevant facts [Lozi86]. In this sense, free or bound information given in the arguments of a query predicate can play a very

important role in reducing the search space so that, from the very beginning, the inference could be directed by this bounded information.

Sacca and Zaniolo [Sacc86, Zani86, Sacc87] and Ullman [Ullm85] show major efforts in this direction using the rule/goal graph which is adorned by the free/bound information of arguments of predicate.

Let us take a familiar example: **ancestor(X, Y)** which states that *Y is a ancestor of X*. In this case four query types are possible as follows:

(a) ancestor(mary, john).
(b) ancestor(mary, Y).
(c) ancestor(X, john).
(d) ancestor(X, Y).

In the first three cases, we can certainly take advantage of the bounded information in the predicate to reduce the search space in order to process the queries.

A.3 Preferred Reasoning Schemes

This criterion is concerned with the reasoning schemes that depend on the types of queries. Some queries can be more efficiently processed by forward reasoning, while others may be done by either backward or bi-directional reasoning schemes.

Kellogg [Kell84] briefly mentions this idea in his paper as follows:

> A "what-if" query requires forward reasoning from the "assumed" or "given" relationships through the rule base and the database to the generation of answers while a "find" query uses backward reasoning from "goal" relationships toward rules and data support for that goal. Bi-directional reasoning is used when both "given" and "goal" relationships occur in a query (i.e., "a "given-find" query).

We believe that this idea can further be explored by considering the relationships with other criteria.

A.4 Types of Predicates Used in a Query

This criterion distinguishes queries by the types of predicates used. The predicates can be classified as follows:

(a) Predicates that are defined only in an EDB

(b) Predicates that are defined only in an IDB

(c) Predicates that are defined both in an EDB and in an IDB

(d) Predicates that are system commands such as WHY, HOW, WHAT-IF, and PROVE.

(e) Queries mixed by predicates (a) through (d).

The types *(a)*, *(b)* and *(c)* are usual queries in DDBs. We want to distinguish type *(d)* from those former types. Types *(a)*, *(b)*, and *(c)* are concerned with a specific database schema, while type (d) can be independently defined regardless of the schema in a DDB.

A.5 Types of Rules

This criterion is based on the types of rules defined in an IDB. Many researchers have focused on the efficient realization of recursive rules in an IDB [Hens84, Ullm85, Lozi85, Viei86, Yoko86, Banc86, Han85] Among them, Han [Han85] provides an extensive classification of recursive rules and algorithms to process recursive queries. We refer to [Han85] for the classification of recursive rules.

Bancilhon and Ramakrishnan [Banc86a] survey and compare some known strategies for the processing of recursive queries.

Even though many different strategies in processing various recursive queries in DDBs have been proposed, it is still believed that there is no complete general scheme, yet.

According to our survey on recursive query processing, we believe the recursive query processing scheme must have the characteristics summarized below. These features are significant since these are the wish-list of a practical and ideal DDB, and could become the criteria for comparison of each approach.

Algorithms for recursive query processing in database systems

- must terminate.
- must provide complete answers for a given query.
- must provide all answers at one time.
- may need to consider different algorithms for different types of rules.
- must work for all types of recursive rules and queries.
- must take advantage of problem specific data given in the query.
- use relational algebra operations, if possible.
- need to clarify the effect of including function symbols and comparison predicates.
- need to find the largest subset of first order logic in which all these computations are efficient.

APPENDIX B

CAR-DEALERSHIP DATABASE

This appendic explains a CAR-DEALERSHIP database which is widely used in this dissertation.

EDB has three relations:

EDB Schema:

(d1) emp(Name, Salary, Job-type)
(d2) car(Cno, Model, Year, Price)
(d3) sold(Name, Cno)

The relation *emp* stores employees' *name, salary* and *job type*. The relation *car* stores *car number, model, year*, and *the price of car*. The relation *sold* stores which salesmen sold out which cars.

IDB:

IDB consists of 4 rules. Each rule is commented right above each rule with their rule numbers.

(r1): **/* Car C1 is an expensive car if its price is over 20k */**

expensive-car(C1) ← car(C1, M1, Y1, P1), gt(P1, 20)

(r2): **/* Car C2 is an economic car if its price is not greater than 5K */**

economic-car(C2) ← car(C2, M2, Y2, P2), ¬ gt(P2, 5)

(r3): **/* Manager's salary is greater than 20K */**

gt(S3, 20) ← emp(N3, S3, manager)

(r4): **/* The price of Benz is over 20K */**

gt(P3, 20) ← car(C3, benz, Y3, P3)

APPENDIX C

DEPARTMENT DATABASE

We define and explain DEPARTMENT database which is widely used in this dissertation. The example is similar to [Reit78, Chak84].

EDB has 3 relations. They are:

EDB Schema:

teaches(Tname, Dname, Cno)
enrolled(Sname, Dname, Cno)
dept(Dname, Cno, CHrs)

The relation *teaches* stores teachers' *name, department name,* and *the course number* he/she is teaching. The relation *enrolled* stores students' *name, department name,* and *the course number* the student is taking. The relation *dept* describes departments that have courses with specific credit hours. It stores *department name, course number,* and *number of credit hours.*

EDB facts:

teaches	Tname	Dname	Cno
	allen	phil	100
	smith	phil	200
	cook	phil	300
	davis	hist	100
	davis	hist	200

enrolled	Sname	Dname	Cno
	gray	math	100
	gray	phil	300
	gray	csc	100
	haas	math	200
	haas	csc	200
	haas	csc	300
	james	hist	100
	kelly	hist	200

dept	Dname	Cno	CHrs
	math	100	3
	math	200	4
	csc	100	3
	csc	200	3
	csc	300	3
	hist	100	2
	hist	200	3
	phil	100	2
	phil	200	3
	phil	300	3

IDB:

The IDB consists of 3 rules. Each rule is commented right above each rule with their rule numbers.

(r1) /* Professor *Allen* teaches all math courses.*/
teach(allen, math, Cno) ← dept(math, Cno, CHrs)

(r2) /*Professor *Baker* teaches all computer science courses. */
teach(baker, csc, Cno) ← dept(csc, Cno, CHrs)

(r3) /*If a teacher T is teaching a course Cno in department D and a student S is enrolled in a course Cno, then T is a teacher of S. */
teacher_of(S, T) ← enrolled(S, D, Cno), teach(T, D, Cno)

(r4) /*All other teaching information is in the EDB relation *teaches* */
teach(X, Y, Z) ← teaches(X, Y, Z)

For the detailed discussion of this database, see *Discussion of Examples 7.3.1 and 7.3.2* in Section 7.3.2.

VITA

Il Yeol Song was born in Andong, Korea, on March 10, 1953, the son of Hee-Sik Song and Seung-Hyang Lee. He studied at Andong High School and graduated from Han Yang University with a B.E. degree in Nuclear Engineering in March 1975. He then worked for Agency for Defense Development for Korea, one of the most prestigious organization for college graduates in Korea at that time, from March 1975 to July 1982 as a researcher and a senior researcher. In March 1979, he married Ms. Young-Choon Yoo in Seoul, Korea and they now have two daughters, Jae-Won and Sharon. He entered Louisiana State University at Baton Rouge in August 1982 and received a M.S. degree in Systems Sciences in December 1984. He is now a Ph.D candidate in the Department of Computer Science at LSU in August 1988. His non-academic interests involve martial arts, traveling, reading, and music. He accepted an Assistant Professorship in the College of Information Studies at Drexel University, Philadelphia, Pennsylvania.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Il Yeol Song

Major Field: Computer Science

Title of Dissertation: Intensional Query Processing in Deductive Database Systems

Approved:

Peter P. Chen
Major Professor and Chairman

F. Glen Kemery
Dean of the Graduate School

EXAMINING COMMITTEE:

Mark M. Pat

Ronald Hall

Sukhamay Kundu

Leslie P Jones

Dan B. Rinker

Date of Examination:

July 8, 1988