Design and Implementation of a Three-Step Intensional Query Processing Scheme

Il-Yeol Song  
Drexel University

Hyoung-Joo Kim  
Seoul National University

When processing a query in a conventional database, a set of facts (extensional answers) are usually returned for an answer set. This can be different in a deductive or expert database system. Users may, as before, be interested in a set of facts to answer their query, but in certain queries users are also able to get the answer of a query as a set of formulas (intensional answers). These “intensional answers” can greatly reduce the costs of processing a query, can be represented in a more compact way than a large set of facts, and are moreover independent of the current state of the database. In this paper, we introduce an intensional query processing technique composed of three steps and its implementation in Prolog.

The three steps are pre-resolution, resolution, and post-resolution. In pre-resolution step, deductive rules are transformed into certain forms. In resolution step, SLD-resolution is applied until the last resolvents consist of either extensional literals or comparison literals. In post-resolution step, intensional answers are generated and checked against integrity constraints to remove meaningless answers. Our work is an extension to Cholvy and Demolombe (1986), and Pascual and Cholvy (1988).

Introduction and Motivation

Relational database systems generate only a set of factual data (extensional answers) as an answer set for a given query. Deductive database systems, however, can generate a set of first order logic formulas as an answer set (intensional answers) for a given query. A query is called intensional if answers to the query can be represented by a set of formulas. When querying a deductive or expert database for intensional answers, we are not interested in the set of objects that satisfy a given query in a particular database state. Instead, we want to know the conditions that objects must satisfy, in any state, to belong to an answer.

But why are we dealing with intensional query processing at all? As we get intensional answers as a set of formulas, they are independent of the particular circumstances in the database. Displaying the output of a query in terms of a formula gives us the answer in a more compact form than a set of facts could ever do. Intensional query processing also has an advantage in computation compared with extensional answers. Intensional answers can be computed without accessing the database, which saves a lot of time. Also, intensional answers tell us exactly what conditions must be fulfilled to get a certain extensional answer. That means intensional answers can help us to interpret extensional answers.

This paper looks closely at some work dealing with the topic of intensional answers and suggests a technique composed of three steps.

Terminology

We assume the reader is familiar with the terminology of first-order logic which are normally defined in standard logic references (Chang & Lee, 1973), such as formulas,
atomic formulas, free variables, bound variables, ground formulas, etc.

**Deductive Databases**

A Deductive Database (DDB) is a database in which new facts may be derived from the facts that were explicitly stored by using an inference system (Gallaire et al., 1984). The database of a DDB can be divided into an extensional database (EDB), which consists of a set of facts explicitly stored in a physical database, and an intensional database (IDB) consisting of a set of deductive rules. These rules can be used to derive new facts from the facts in the EDB. Deductive databases deal with two different types of queries which can be distinguished by the kind of answer they are providing. These two are known as:

- **extensional answers**
- **intensional answers**

Whereas an extensional answer consists solely of a set of facts, an intensional answer is a set of non-ground first-order logic formulas. So an extensional query is a query which has only extensional answers and an intensional query has only intensional answers.

**Rules and Horn clauses**

Previous works in intensional query processing used resolution to generate answer formulas for a given query. In order to deal with resolution, we first must introduce the terms of rules and clauses

A **rule** has the form \( a \leftarrow b_1, b_2, \ldots, b_m \), \( m \geq 0 \) where \( a \) is an atomic formula and the \( b_i \)'s are literals. All the variables occurring in the the rule are assumed to be universally quantified. The literal \( a \) is called the **head** of the rule, the \( b_i \)'s are referred to as the **body** of the rule.

A **clause** is finite disjunction of zero or more literals. Every rule can be represented as a clause, so our rule can be represented as a clause, and would look like:

\[
\forall x_1 \forall x_2 \forall x_3 \ldots \forall x_n : \neg a \leftarrow \neg b_1, \neg b_2, \ldots, \neg b_m
\]

We note that the head of a rule becomes a positive literal, and predicates in the body of a rule becomes negative literals in the clause form. In this paper we are only dealing with Horn clauses, which are a subset of clauses that have at most one positive literal.

**Example**

The following example illustrates the notion of intensional answers. Suppose we have a database for students, consisting of 3 relations representing the extensional database and 4 rules constituting the intensional database.

**EDB:**
- (relation 1:) student(S_name,Year)
- (relation 2:) course(C_number,C_name,C_hrs)
- (relation 3:) has_taken(S_name,C_number)

The relation _student_ contains information about student names (S_name) and the years they have been studying (Year). The relation _course_ contains a course number (C_number), the course name (C_name) and the number of credit hours (C_hrs). The relation _has_taken_ describes who (S_name) has already taken what courses (C_number).

**IDB:**
- (rule 1:) pre_req(S_name,4402) \leftarrow has_taken(S_name,3102)
- (rule 2:) cannot_take(S_name,C_number) \leftarrow student(S_name,1), C_number = 3102
- (rule 3:) cannot_take(S_name,C_number) \leftarrow student(S_name,Year), \neg pre_req(S_name,C_number)
- (rule 4:) cannot_take(S_name,C_number) \leftarrow student(S_name,1), C_number > 4000

Rule 1 states that course number 3102 is a prerequisite for course 4402. Rule 2 says that a freshman cannot take course 3102. The third rule tells us that if the student does not satisfy the prerequisite of a course, then he cannot take that course. Finally rule 4 states that freshmen cannot take courses above the 4000 level.

Suppose we have a query _cannot_take(X,4402)?_, asking “Who cannot take course 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 course 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

\[
\text{ans}_1 = \text{student}(X,1) \quad \text{and} \quad \text{ans}_2 = \neg \text{has_taken}(X,3102)
\]

The first answer implies that if any student is a freshman, then he cannot take 4402. The second one implies that if any student has not taken 3102, then he cannot take 4402.

This example shows that whereas intensional answers distinguish between two cases in which a student cannot take 4402, extensional answers would only provide a set of facts. The user would not be able to conclude for which reason a student could not take that particular course. Thus, by telling us what conditions or formulas constitute the answers, intensional answers are more informative than extensional ones in some cases.

**Resolution and SLD Tree**

Resolution is an automatic theorem proving method that uses a refutation proof procedure. So, in order to prove that a theorem \( F \) follows from a set of axioms \( T \), we prove that the formula \( T \land \neg F \) is unsatisfiable using resolution.

For query processing, when resolution is used in the context of databases, a query formula is negated, converted