

Logic as a database language, Information Integration, CSCI 6967-01

1 Logic Basics

Assume the existence of constants over a (possibly infinite) set D , a set of variables and predicate symbols. Variables will be denoted with upper case letters (such as X, Y, Z) and constants and predicates will be denoted with lower case letters. In addition, we assume the existence of function symbols (f, g).

A **term** is defined recursively as follows:

- A variable or a constant is a term.
- If t_1, \dots, t_n are terms and f is an n -ary function, then $f(t_1, \dots, t_n)$ is also a term.

Intuitively, a term is an expression denoting some value. Predicates denote m -ary relations and take terms as arguments. If t_1, \dots, t_m are terms and p is an m -ary predicate symbol, then $p(t_1, \dots, t_m)$ is an **atom**. An atom is said to be **ground** if it contains no variables, in this case it is also called a **fact**. Atoms represent true facts about the world they model. For example:

$parent(sibel, lalé)$
 $parent(sibel, turgay)$

may be used to represent that $lalé$ and $turgay$ are two parents of $sibel$. Note that, we say that $parent(sibel, lalé)$ is true in the database and is a fact.

Conjunctive rules of the form:

$H : \neg G_1, G_2, \dots, G_n$

makes it possible to define new facts based on existing facts. We read this rule as H is true (or H is a fact) if G_1, G_2, \dots, G_n are all true. H is called the **head** of the rule and G_1, G_2, \dots, G_n is called the **body** of the rule and each G_i is called a **subgoal** of the rule. Each subgoal is either a predicate symbol or an expression involving a built-in predicate such as $=, >, \dots$. Other notations for conjunctive rules are equivalent:

$H \leftarrow G_1, G_2, \dots, G_n$

$H \leftarrow G_1 \wedge G_2 \wedge G_n$

Note that facts are simply rules with an empty body, i.e. $parent(sibel, lala)$ is equivalent to $parent(sibel, lala) : -true$, which means that it is true without a condition.

A substitution θ is a set of equality relations of the form $A = B$ where A is a variable and B is a term that does not contain A . A substitution is said to be grounding if all B in the substitution is a term with no constants. A unifier for two predicates $p(\dots)$ and $q(\dots)$ is a substitution such that $p(\dots)\theta = q(\dots)\theta$.

2 Databases

A **data model** in databases consists of a set of predicates where each attribute is assigned a specific domain of values (hence is a subset of D). A **database instance** is a set of facts that are true for the given data model. We will also call the database instance the **model** of a database. We store the facts for the database in one of two ways:

1. The **extensional** database (**EDB**) stores a set of explicit facts. This is similar to the data that is stored in a database management system.
2. The **intensional** database (**IDB**) stores a set of rules that can be used to derive new facts based on EDB.

If none of the rules in the database contain function symbols, then the database is considered to be in **Datalog** format. Otherwise, we refer to it as **First order logic**.

An example database is given below:

$$\begin{aligned} &p(1, 2). \\ &p(2, 3). \\ &a(X, Y, 1) : -p(X, Y). \\ &a(X, Z, d(T)) : -a(X, Y, T), p(Y, Z). \end{aligned}$$

To find the model of a database, we can simply find a set of ground atoms I (also called an interpretations) such that all facts in the database are true in I and for any possible ground version r' of a rule in the database, if the body of r' is true with respect to I , then the head of r' is also in I . The model that is obtained by intersecting all possible models for the database is called the **least model**. The least model is the most compact database that can be obtained from this definition and hence is considered to be the correct model for the database.

Note that a Datalog program with a finite set of facts in EDB has a finite least model. A database in general may have an infinite set of facts in its least model. The least model for the above database is:

$$\{p(1, 2), p(2, 3), q(1, 2, 1), q(2, 3, 1), q(1, 3, d(1))\}$$

2.1 Fixpoint computation

The least model of a database can be computed using a **bottom up** computation as follows. Let F be the fixpoint operator. Each iteration i of the operator is denoted by F^i .

$$\begin{aligned}
 F^0 &= EDB \\
 F^i &= \{H\theta \mid (H : -G_1, G_2, \dots, G_n) \in IDB, \theta \text{ is a grounding substitution, } G_1\theta, G_2\theta, \dots, G_n\theta \in F^{i-1}\}, i > 0
 \end{aligned}$$

The fixpoint operator is said to have a fixpoint if for some i , $F^{i+1} = F^i$. The smallest such i is considered to be the fixpoint for F . The fixpoint computation for the above program is given below:

$$\begin{aligned}
 F^0 &= \{p(1, 2), p(2, 3)\} \\
 F^1 &= \{p(1, 2), p(2, 3), q(1, 2, 1), q(2, 3, 1)\} \\
 F^2 &= \{p(1, 2), p(2, 3), q(1, 2, 1), q(2, 3, 1), q(1, 3, d(1))\} \\
 F^3 &= F^2
 \end{aligned}$$

Note that, the following program has a fixpoint that is reached only after applying F an infinite number of times:

$$\begin{aligned}
 &p(0). \\
 &p(f(X)) : -p(X). \\
 F^\infty &= \{p(0), p(f(0)), p(f(f(0))), \dots\}
 \end{aligned}$$

The contents of the fixpoint operator after it reaches its fixpoint is equivalent to the least model for a database.

2.2 Proof computation

A query is given by a headless rule, i.e. a set of subgoals G_1, \dots, G_n that needs to be proven. To prove a set of goals, each subgoal needs to be proven. Suppose we are given a query

$$: -G_1, \dots, G_n$$

and a rule $H : -G'_1, \dots, G'_m$ where all the variables in the rule are renamed so that there are no common variable names between the query and the rule. Suppose θ is a unifier for G_i and H (i.e. $G_i\theta = H\theta$). Then, the result of resolution of the query with this rule is given by:

$$: -(G_1, \dots, G_{i-1}, G'_1, \dots, G'_m, G_{i+1}, \dots, G_n)\theta$$

where the subgoal G_i is replaced by the body of the rule and the substitution θ is applied to all the predicates in the subgoal.

Recall that a fact is a rule with an empty body. Hence, applying resolution with a fact will reduce the number of subgoals to be proven by one.

A proof is then a sequence of resolution steps (sometimes called SLD-resolution) such that at the end of the last step, the query has empty body. If $\theta_1, \dots, \theta_x$ are the substitutions applied to find this proof, then this implies that

$$: -(G_1, \dots, G_n)\theta_1, \dots, \theta_x$$

is implied by the database ($\theta_1, \dots, \theta_x$ means that we are applying all the substitutions one after another). If the query contains a single predicate $: -G$, then the proof implies that $G\theta_1, \dots, \theta_x$ is a fact in the database.

The set of all facts that can be proven by is identical to the least model, which is identical to the set of facts in the fixpoint of the database. The proof method is called a top down method where a different proof is needed for each fact. If there exists no proof for a fact, then we conclude that the fact is not in the database.