Deductive Databases

A Short Note

April 18, 2005

Datalog - a logical query language that allows us to express queries about relations.

1 Predicates and Atoms

A relation is represented in Datalog by a *predicate*. Each predicate has a fixed number of arguments.

An *atom* is a predicate followed by its arguemnts.

Example 1

Title	Year	Length	Type
Harry Porter	2001	180	Color
Gone with the wind	1938	125	Black and White
Snow White	1950	90	Color

The Movie Relation

in *Datalog* term, this relation is represented by the predicate named $movie$ that have four arguments.

$movie(\text{“}Harry\ Porter\text{”}, 2001, 180, \text{“}Color\text{”})$ is an atom consisting of the predicate symbol $movie$ and the values of the four arguments are ”Harry Porter”,2001,180, and ’Color”, respectively.

In Datalog, a predicate represents the membership function of tuples with respect to a relation. For example, if p is a relation with n attributes with some fixed order, we will use p as the name of the predicate representing this relation. Then, $p(v_1, \ldots, v_n)$ is true if $(v_1, \ldots, v_n)$ is a tuple of p; it is false otherwise. For instance, for the relation movie in Example 1, the three atoms:

$movie(\text{“}Harry\ Porter\text{”}, 2001, 180, \text{“}Color\text{”})$
$movie(\text{“}Gone\ with\ the\ wind\text{”}, 1938, 125, \text{“}Black\ and\ White\text{”})$
$movie(\text{“}Snow\ White\text{”}, 1950, 90, \text{”}Color\text{”})$

are true and all other atoms, say $movie(\text{“}Harry\ Porter\ -\ the\ next\text{”}, 2002, 134, \text{“}Color\text{”})$, are false.

NOTE: The types of argumenrs are not specified. They are often implicitely defined. Also, the name of an argument in a particular atom is not important, rather its order is important. E.g., $movie(X, Y, Z, U)$ and $movie(A, B, C, D)$ is identical.

An atom is a *ground atom* if every of its arguments is a constant. Otherwise, it is a *non-ground* atom. A non-ground atome has some variables (in this note, they are terms beginning with upper-case letter) could be viewed as a Boolean function that takes values for these variables and then returns true or false.

$movie(\text{“}Harry\ Porter\text{”}, 2001, 180, \text{“}Color\text{”})$ is a ground atom and $movie(X, 2001, Y, Z)$ is a non-ground atom with three variables X, Y, Z. $movie(X, 2001, Y, Z)$ is true if $X = \text{“}Harry\ Porter\text{”}$, $Y = 180$, and $Z = \text{“}Color\text{”}$; it is false otherwise.

2 Arithmetic Atoms

An *arithmetic atom* is a comparison between two arithmetic expressions, like $x + y < z$ or $(x - 3) * 7 \geq 5$ etc. Variables occurring in an arithmetic atom are arguments of predicates. An atom – defined as in the previous subsection – is called a *relational atom*.

3 Datalog Rules and Queries

A Datalog rule has the form

$$a \leftarrow a_1, \ldots, a_n, not\ a_{n+1}, \ldots, not\ a_{n+m} \tag{3.1}$$

where a is a relational atom (called the $head$ of the rule) and $a_1, \ldots, a_{n+m}$ are atoms (called the $body$ of the rule). not is the negation-as-failure operator. Each of the $a_1, \ldots, a_n$ is called a $subgoal$ and each of the $a_{n+1}, \ldots, a_{n+m}$ is called a negated $subgoal$.

A rule of the form (3.1) reads as if $a_1, \ldots, a_n$ are true and $a_{n+1}, \ldots, a_{n+m}$ can be assumed to be false then a must be true.

Example 2 We define the predicate $colorMovie$ as follows:

$colorMovie(M, X) \leftarrow movie(M, X, Y, Z), Z = \text{“}Color\text{”}$.

That is, we say that a movie is a color movie if the fourth argument of the predicate has the value “Color”. Let assume that we have the same relation instance as above, we would have $colorMovie(\text{“}Harry\ Porter\text{”}, 2001)$ and $colorMovie(\text{“}Snow\ White\text{”}, 1950)$ true.

Notice that this is exactly the result we get from the relational algebra expression

$$\pi_{title,year}(\sigma_{type=\text{“}Color\text{”}}(Movie)).$$

Query. A set of datalog rules is called a *query*. It is also called a *program*. A query defines one or more predicates (those occur in the head of the rules.)

To answer a query, we compute all the atoms that belong to the predicates in the head of rues of the query. This can be done by considering all possible values of the variables occurring in the rule, then checking if the head would be true following the definition of a rule. In this process, we will only need to consider *consistent assigments* which assign the same value for a variable occurring in different places of the rule.

Example 3 Given the rule

$$p(X, Y) \leftarrow q(X, Z), r(Y, 1), Y \leq 2, not\ X \geq 5$$

and assume that X, Y, Z range over the integer domain. One possible variable assignment for this rule is given by the following rule

$$p(2, 3) \leftarrow q(3, 4), r(2, 1), 3 \leq 2, not\ 3 \geq 5.$$

This assignment is not consistent because the variable Y (the second argument of p) receives different value than the variable Y (the first argument of r).

A consistent assignment is given by

$$p(2, 3) \leftarrow q(2, 4), r(3, 1), 3 \leq 2, not\ 2 \geq 5.$$

Safe rule. A rule of the form (3.1) is *safe* if every variable occurring in the rule occurs in some nonnegated, relational atom.

Example 4 The rule

$$p(X, Y) \leftarrow q(X, Z), r(Y, 1), Y \leq 2, not\ X \geq 5$$

is safe but the rule

$$p(X, Y) \leftarrow q(X, Z), X < Y, T > 4, not\ r(W, X, Z)$$

is unsafe because W occurs in $r(W, X, Z)$ but not in a nonnegated, relational atom. (find another variable) In this rule, we can say that the subgoal $T > 4$ will be always true if the domain of T contains a value greater than 4. In this case, $T > 4$ could be removed from the rule.

How to compute the atoms defined by a rule.

Example 5 Assume that q and r are two predicates and that the atoms of q and r are $q(1, 2)$, $q(1, 3)$ and $r(2, 3)$, $r(3, 1)$, respectively. Consider the rule

$$p(X, Y) \leftarrow q(X, Z), r(Z, Y), not\ q(X, Y).$$

This rule defines the predicate p. We compute the atoms of p as follows.

	Tuple for $q(X, Z)$	Tuple for $r(Z, Y)$	Consistent Assignments	$not\ q(X, Y)$ True?	$p(X, Y)$
1)	(1,2)	(2,3)	Yes (Why?)	No	false
2)	(1,2)	(3,1)	No (Why?)	-	-
3)	(1,3)	(2,3)	No	-	-
3)	(1,3)	(3,1)	Yes	Yes	true

Computing $p(X, Y)$

Extensional and Intentional Predicates.

- *Extensional predicates*: whose relations are stored in a database. (q and r in Example 5)
- *Intensional predicates*: whose relations are computed by applying one or more Datalog rules. (p in Example 5)

Example 6 (Writing queries in Datalog) Let us consider the relational database schema on products, pcs, laptops, and printers:

- $product(maker, model, type)$
- $pc(model, speed, ram, hd, cd, price)$
- $laptop(model, speed, ram, hd, screen, price)$
- $printer(model, color, type, price)$

Some queries:
What PC models have a speed of at least 150?

$$fastPC(M) \leftarrow pc(M, S, \_, \_, \_, \_), S \geq 150.$$

NOTE: The symbol '_' stands for don-care value (or every possible value)

Find the model number and price of all products (of any type), made by manufacture 'B'.

$productOfB(M, P) \leftarrow product(Maker, M, \_), pc(M, \_, \_, \_, \_, P), Maker = `B'.$
$productOfB(M, P) \leftarrow product(Maker, M, \_), laptop(M, \_, \_, \_, \_, P), Maker = `B'.$
$productOfB(M, P) \leftarrow product(Maker, M, \_), printer(M, \_, \_, , P), Maker = `B'.$

We learn how relational algebra operators can be represented using Datalog rules. Assume that we have r have n attributes (then so is s). For each operator, we develop a Datalog rule query that yields the same answer as the relational operator.

Union ($\cup$) $union(X_1, X_2, \ldots, X_n) \leftarrow r(X_1, X_2, \ldots, X_n).$
$union(X_1, X_2, \ldots, X_n) \leftarrow s(X_1, X_2, \ldots, X_n).$

Set difference (\) $difference(X_1, X_2, \ldots, X_n) \leftarrow r(X_1, X_2, \ldots, X_n), not\ s(X_1, X_2, \ldots, X_n).$

Intersection (∩) :

$intersection(X_1, X_2, \ldots, X_n) \leftarrow r(X_1, X_2, \ldots, X_n), s(X_1, X_2, \ldots, X_n).$

Projection (π) $pi(A_1, A_2, \ldots, A_k) \leftarrow r(..., A_1, ..., A_2, ..., A_k, ...)$. (Here, the $A_1, \ldots, A_k$ is the sequence of attributes we want.)

Selection (σ) Consider $\sigma_C(R)$. If C is a conjunction $C_1 \wedge \ldots \wedge C_k$ where each C_i is an atom or a neative atom $\neg D_i$ then

$select(A_1, \ldots, A_n) \leftarrow r(A_1, \ldots, A_n), C'_1, \ldots, C'_k$. where $C'_j = C_j$ if C_j is not of the form $\neg D_j$ and $C'_j = not\ D_i$ if $C_j = \neg D_j$.

Cartesian Product ($\times$) $cartesianProduct(X_1, \ldots, X_k, Y_1, \ldots, Y_m) \leftarrow r(X_1, \ldots, X_k), s(Y_1, \ldots, Y_m).$

Natural Joins ($\rhd\lhd$) Let assume that r has k attributes and s has m attributes, the set of shared attributes is $\{X_{i_1}, \ldots, X_{i_t}\}$ of r and $\{Y_{j_1}, \ldots, Y_{j_t}\}$ of s

$join(X_1, \ldots, X_k, Z_1, \ldots, Z_l) \leftarrow r(X_1, \ldots, X_k), s(Y_1, \ldots, Y_m), X_{i_1} = Y_{j_1}, \ldots, X_{i_t} = Y_{j_t}.$ (*)
where $\{Z_1, \ldots, Z_l\} = \{Y_1, \ldots, Y_m\} \setminus \{Y_{j_1}, \ldots, Y_{j_t}\}$ and (*) needs to be modified to take this condition into account.

Renaming (ρ): $\rho_{S(A_1,...,A_k)}(R)$

$s(A_1, \ldots, A_k) \leftarrow r(X_1, \ldots, X_k), X_1 = A_1, \ldots, X_k = A_k.$

Combining Operations to Form Queries. Any complex relational algebral expression can be expressed by a Datalog query.

4 Recursive Programming in Datalog

Consider the relation

Mother	Child
Ana	Deborah
Deborah	Nina
Nina	Anita

Question: Can we write a relational algebral expression for the $ancestor(X, Y)$ relation?

(1) $ancestor(X, Y) \leftarrow mother(X, Y).$
(2) $ancestor(X, Z) \leftarrow ancestor(X, Y), mother(Y, Z).$

See also the example in the book.

How to compute the IDB $ancestor$?

1. Assume that $ancestor = \emptyset$.
2. Compute the new atoms of the predicate $ancestor$.
3. Until no new atom can be derived, stop.

Example 7 1. $ancestor = \emptyset$.

2. Only the body of (1) can be true. So, we get:

   $ancestor(ana, deborah).$
   $ancestor(deborah, nina).$
   $ancestor(nina, anita).$

3. Continue with the new set of atoms for ancestor: rule (1) is no longer useful (does not produce new atoms), but rule (2) produces the following atoms.

   $ancestor(ana, nina).$
   $ancestor(deborah, anita).$

Example 8 Consider an airline time table

Airline	from	to	departs	arrives
UA	SF	DEN	930	1230
AA	SF	DEN	900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	NY	1500	1930
AA	DAL	CHI	1530	1730
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

• Defining a predicate named *reaches*(X, Y) that means "Y is reachable from X"

(1) $reaches(X, Y) \leftarrow flight(A, X, Y, D, R).$
(2) $reaches(X, Z) \leftarrow reaches(X, Y), reaches(Y, Z).$

Computing the $reaches$ relation:

1. $reaches = \emptyset$.

2. Only the body of (1) can be true. So, we get:

   (SF, DEN), (SF, DAL), (DEN, CHI), (DEN, DAL), (DAL, CHI), (DAL, NY), (CHI, NY)

3. Continue with the new set of atoms for ancestor: rule (1) is no longer useful (does not produce new atoms), but rule (2) produces the following atoms.

   (SF, CHI), (SF, NY), (DEN, NY)

• Defining a predicate named *connects*(X, Y, D, R) that means "starting from X at D, take one or more flights, arrive at Y at R"

(1) $connects(X, Y, D, R) \leftarrow flight(A, X, Y, D, R).$
(2) $connects(X, Y, D, R) \leftarrow connects(X, Z, D, T1), connects(Z, Y, T2, R), T1 <= T2 - 100.$

Computing the IDB we get the following table

Iteration	from	to	departs	arrives
1	SF	DEN	930	1230
1	SF	DEN	900	1430
1	DEN	CHI	1500	1800
1	DEN	DAL	1400	1700
1	DAL	NY	1500	1930
1	DAL	CHI	1530	1730
1	CHI	NY	1900	2200
1	CHI	NY	1830	2130
2	SF	CHI	900	1730
2	SF	CHI	930	1800
2	SF	DAL	930	1700
2	DEN	NY	1500	2200
2	DAL	NY	1530	2130
2	DAL	NY	1530	2200
3	SF	NY	900	2130
3	SF	NY	900	2200
3	SF	NY	930	2200

5 Negation in Recursive Rule

When negated atoms appear in the body of rules of a datalog program, the set of answers of a program might not be unique.

6 Only one answer

Find all the cities reachable by UA but not by AA.

(1) $reachesUA(X,Y) \leftarrow flight(ua,X,Y,D,R).$
(2) $reachesUA(X,Y) \leftarrow reachesUA(X,Z), reachesUA(Z,Y).$

(3) $reachesAA(X,Y) \leftarrow flight(aa,X,Y,D,R).$
(4) $reachesAA(X,Y) \leftarrow reachesAA(X,Z), reachesAA(Z,Y).$

(5) $reachesUAOnly(X,Y) \leftarrow reachesUA(X,Z), not\ reachesAA(X,Z).$
Computing $reachesUAOnly$ – compute $reachesUA$ and $reachesAA$. We still get a unique set of atoms of $reachesUAOnly$.

7 Multiple Answers

Let r be a relation that has only one attribute and one tuple $r(0)$. Consider

(1) $p(X) \leftarrow r(X), not\ q(X).$
(2) $q(X) \leftarrow r(X), not\ p(X).$
X belongs to r then X belongs to p or q but not both.

Question: $p(0)$? or $q(0)$?

Two possible solutions: a) $p(0)$ and $\neg q(0)$; and b) $\neg p(0)$ and $q(0)$

8 Stratified Datalog Program

To make sure that a Datalog program gives only one solution.

For a program P, G_P, the dependency graph of P is defined as follows:

1. each IDB predicae is a node of G_P
2. for each rule with the predicate A in the head and the body contains a negated subgoal in the body with the predicate B, there is a rule from A to B with the label '-'
3. for each rule with the predicate A in the head and the body contains a non-negated subgoal in the body with the predicate B, there is a rule from A to B with the label '+'

Definition 1 *A program P is stratified if its dependency graph G_P does not contain a cycle containing a negative arc.*

Example 9 The program for $reachesUAOnly$ is stratified whereas the program for p and q is not.

9 Models of A Datalog Program

We will define the notion of a *model* of a program. In this section, we will assume that programs do not contain variables. We begin with programs that do not contain negated atoms. They are called *positive programs* or *definite programs*.

10 Minimal Models of Positive Programs

Let P be a program without negated atoms. We define a fixpoint operator T_P that maps a set of atoms (of program P) to another set of atoms as follows.

$$T_P(X) = \{a \mid \text{ there exists a rule } a \leftarrow a_1, \ldots, a_n \text{in } P \text{ s. t. } a_i \in X\} \quad (10.2)$$

It is easy to see that $T_P(X) \subseteq T_P(Y)$ if $X \subseteq Y$. This implies that $T_P(X)$ is a monotonic function and hence it has a fixpoint that can be computed as follows.

1. Let $X_1 = T_P(\emptyset)$ and $k = 1$
2. Compute $X_{k+1} = T_P(X_k)$. If $X_{k+1} = X_k$ then stops and return X_k.
3. Otherwise, increase k and repeat the second step.

Note: The above algorithm will terminate for positive program. For each program P, let $M(P)$ denote the result of this algorithm.

Definition 2 *$M(P)$ is called a* minimal model *of P.*

Proposition 1 *$M(P)$ is the least set of atoms that satisfies the following conditions:*

1. *For every rule $a \leftarrow a_1, \ldots, a_n$, if a_i's belong to $M(P)$ then a belongs to $M(P)$.*
2. *No proper subset of $M(P)$ satisfies the first condition.*

11 Stable Models of Normal Programs

Let P now a normal datalog program (it might contain negated atoms). For a set of atoms X, by P^X we denote the program obtained from P by

1. Deleting any rule $a \leftarrow a_1, \ldots, a_n, not\ a_{n+1}, not\ a_{n+m}$ for that $\{a_{n+1}, \ldots, a_{n+m}\} \cap X \neq \emptyset$, i.e., the body of the rule contains a negated atom $not\ a_l$ and a_l belongs to X; and
2. Removing all of the negated atoms from the remaining rules.

Note: P^X is a positive program.

Definition 3 *A set of atoms X is called a* stable model *of a program P if X is the minimal model of the program P^X.*

12 Stratification and Uniqueness of Stable Model

Proposition 2 *For every stratified program P, P has a unique stable model.*

Checking for stratification. The following algorithm can be used to determine whether P is a stratified program or not.

Algorithm 1 *Let P be a program. Assume that P contains n IDB predicates. We compute a function $strata$ that maps each IDB predicate to an integer as follows.*

1. *Let $j = 0$. Assign $strata(p) = 0$ for every IDB predicate p.*
2. *$j = j + 1$. If $j > n$, then stops and concludes that the program is not stratified. Otherwise, compute a 'new strata' for the p's as follows. For every rule*

$$a \leftarrow a_1, \ldots, a_k, not\ a_{k+1}, \ldots, not\ a_{k+l}$$

   let $n_1 = \max\{strata(a_i) \mid i = 1, \ldots, k\}$ (the maximal strata of non-negated atoms) and $n_2 = \max\{strata(a_{k+i}) \mid i = 1, \ldots, l\} + 1$ (the maximal strata of negated atoms plus one). We define the new strata of a is the maximal of n_1 and n_2, i.e., $strata(a) = \max\{n_1, n_2\}$.
3. *If the new strata is not different than the old strata, stops and concludes that the program is stratified. Otherwise, goes to step 2.*