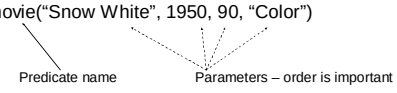
Deductive Database

Relation Database

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

A different view: a set of atoms

movie("Harry Porter", 2001, 180, "Color")
movie("Gone with the wind", 1938, 125, "Black and White")
movie("Snow White", 1950, 90, "Color")



Predicates & Atoms

- Predicate – relation (associated with a fixed number of arguments)
- Atoms – predicates followed by a list of arguments (arguments can be variables or constants)
  - grounded if all arguments are constants, e.g., *movie("Harry Porter", 2001, 180, "Color")*
  - non-grounded if some arguments are variables, e.g., *movie(X, Y, 180, "Color")*

Atoms & Rows

movie("Harry Porter", 2001, 180, "Color")
movie("Gone with the wind", 1938, 125, "Black and White")
movie("Snow White", 1950, 90, "Color")

Each *relation instance* can be represented by a set of grounded atoms
Each *row* corresponds to an *atom*.

An atom is *true* if it corresponds to a row in the table; it is *false* if it does not correspond to any row in the table.

Why Deductive Database?

- Declarative: clear understanding of what a database represents
- Expressiveness: there are queries that cannot be formulated using relational algebra can be easily expressed by datalog rules

Datalog

- Rule
  a :- *a*1, …, *an, not b*1, …, *not bm*

where a, a1,..,an,b1,…,bm are atoms and *not* is the *negation-as-failure* operator (n, m can be 0).

- A program (or query, knowledge base) is a set of rules

Rule - Meaning

- Rule
  a :- *a*1, …, *an, not b*1, …, *not bm*

says the following: "If a1,…,an are true and b1,…,bm can be assumed to be false then a must be true."

If n=0 and m=0, then a must be true (*fact*).
If m=0, the rule is a definite rule.

Rule – Examples

colorMovie(M) :- movie(M,X,Y,Z), Z = "*Color".*

longMovie(M):- movie(M,X,Y,Z), Y>120.

oldMovie(M):- movie(M,X,Y,Z), X<1940.

movieWithSequel(M):- movie(M,X1,Y1,Z1),
movie(M,X2,Y2,Z2),
X1 <> X2.

Rule – Do-not-care variables

colorMovie(M) :- movie(M,_,_,Z), Z = "*Color".*

longMovie(M):- movie(M,_,Y,_), Y>120.

oldMovie(M):- movie(M,X,_,_), X<1940.

movieWithSequel(M):- movie(M,X1,_,_),
movie(M,X2,_,_),
X1 <> X2.

Rule – Negation as failure

noColorMovie(M) :- movie(M,_,_,Z), not Z = "*Color".*

notYetProducedMovie(M) :- not movie(M,X,Y,Z).

NOTE: should be careful when using 'not'; the second rule is not a good one (unsafe). It can say that any movie is not yet produced !

Deductive Database & Relational Algebra

- Projection
  colorMovie(M) :- movie(M,_,_,Z), Z = "*Color".*
- Selection
  oldMovie(M,X,Y,Z):- movie(M,X,Y,Z),X<1940.
- Join (two relations p(A,B), q(B,C))
  r(X,Y,Z):- p(X,Y), q(Y,Z)
- Union (two relations p(A,B), q(A,B))
  r(X,Y):- p(X,Y).
  r(X,Y):- q(X,Y).
- Set difference (two relations p(A,B), q(A,B))
  r(X,Y):- p(X,Y), not q(X,Y).
- Cartesian product (two relations p(A,B), q(C,D))
  c(X,Y,Z,W):- p(X,Y), q(Z,W).

Programs – Examples

Given the following program

Stored in Tables:
q(1,2).
q(1,3).
r(2,3).
r(3,1).

q, r are "extensional predicates"
p is "intensional predicate"

p(*X,Y*):- q(*X,Z*), *r*(*Z,Y*), *not q*(*X,Y*).

Question: what are the atoms of the predicate p which are defined by the program?

Computed from the programs

Programs – Examples

q(1,2). q(1,3). r(2,3). r(3,1).

p(*X,Y*):- q(*X,Z*), *r*(*Z,Y*), *not q*(*X,Y*).

X, Y can be 1, 2, or 3

p(1,3):- q(1,2),r(2,3), not q(1,3). NO
p(1,3):- q(1,3),r(3,1), not q(1,1). YES
….

The meaning of a program

- For a program P, what are the *extensional atoms* defined by P?
- It is simple for the case where rules defining the extensional predicate is non-recursive (previous example!)

Recursive Datalog

- Given: Mother-Child relation
  mother(Ana, Deborah)
  mother(Deborah, Nina)
  mother(Nina, Anita)
- Define: grand-mother relation?
  grandma(X,Y):- mother(X,Z),mother(Z,Y).

Recursive Datalog

- Given: Mother-Child relation
  mother(Ana, Deborah)
  mother(Deborah, Nina)
  mother(Nina, Anita)
- Define: grand-mother relation?
  grandma(X,Y):- mother(X,Z),mother(Z,Y).
- Answer: grandma(Deborah, Anita) and grandma(Ana,Nina)
- Question: relational algebra expression for *grandma*? POSSIBLE?

Recursive Datalog

- Given: Mother-Child relation
  mother(Ana, Deborah)
  mother(Deborah, Nina)
  mother(Nina, Anita)
- Define: ancestor relation?
  ancestor(X,Y):- mother(X,Y).
  ancestor(X,Y):- mother(X,Z), ancestor(Z,Y).
- Answer: ? (mother, grandma, and great-grandma(Ana, Anita))
- Question: relational algebra expression for *ancestor*? POSSIBLE?

Computing the intensional predicates in recursive Datalog

- ancestor = ∅
- Check if any rule satisfies: three rules – get
  - ancestor(Ana, Deborah)
  - ancestor(Deborah, Nina)
  - ancestor(Nina, Anita)
- Repeat the second steps until ancestor does not change:
  - ancestor(Ana, Nina)
  - ancestor(Deborah, Anita)
  - ancestor(Ana, Anita)

Program with Negation-as-failure

- Different situations:
  - There might be only one solution
  - There might be several solutions
  - There might be no solution

Recursive Program – Another Example

- Given a table of flight schedule of UA and AA

Airline	From	To	Departure	Arrival
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

reachable(X,Y) – Y can be reached from X

flight(Airline,From,To,Departure,Arrival)

reachable(X,Y) :- *flight*(A,X,Y,D,R).
reachable(X,Z) :- *flight*(A,X,Y,D,R), *reachable*(Y,Z).

Computing *reachable* – similar to *ancestor*

reachableUA(X,Y) – Y can be reached by UA from X

reachableUA(X,Y) :- *flight*(ua,X,Y,D,R).
reachableUA(X,Z) :- *flight*(ua,X,Y,D,R), *reachableUA*(Y,Z).

Easy: only considers UA flights

reachableUA(X,Y) – Y can be reached by UA's flight from X

reachableUA(X,Y) :- *flight*(ua,X,Y,D,R).
reachableUA(X,Z) :- *flight*(ua,X,Y,D,R), *reachableUA*(Y,Z).

Easy: only considers UA flights

reachableUA(X,Y) & reachableAA(X,Y)

reachableUA(X,Y) :- *flight*(ua,X,Y,D,R).
reachableUA(X,Z) :- *flight*(ua,X,Y,D,R), *reachableUA*(Y,Z).

reachableAA(X,Y) :- *flight*(aa,X,Y,D,R).
reachableAA(X,Z) :- *flight*(aa,X,Y,D,R), *reachableAA*(Y,Z).

reachableOnlyAA(X,Y)

reachableUA(X,Y) :- *flight*(ua,X,Y,D,R).
reachableUA(X,Z) :- *flight*(ua,X,Y,D,R), *reachableUA*(Y,Z).

reachableAA(X,Y) :- *flight*(aa,X,Y,D,R).
reachableAA(X,Z) :- *flight*(aa,X,Y,D,R), *reachableAA*(Y,Z).

reachableOnlyAA(X,Y):- *reachableAA*(X,Y),
not *reachableUA*(X,Y).

reachableOnlyAA(X,Y) – Unique set of atoms satisfying this properties

A different situation

- Suppose that the extensional predicate is *r* and *r*(0) is the only atom which is true
- Consider the program
  p(X) :- r(X), not q(X)
  q(X) :- r(X), not p(X)
- {p(0)} or {q(0)} could be used as 'the answer' for this program

Yet another situation

- Suppose that the extensional predicate is *r* and *r*(0) is the only atom which is true
- Consider the program
  r(X) :- not r(X)
- Is r(0) true or false? Is r(1) true or false?

Minimal Model of Positive Program

- Given a program P without negation-as-failure

$T_P(X) = \{a \mid a :- a1,\ldots,an \in P, \{a1,\ldots,an\} \subseteq X\}$

- If P is a positive program, the sequence
  $T_P(\emptyset)$, $T_P(T_P(\emptyset))$, …
  converges again a set of atoms, which is called the *minimal model* of P

mother(a, d). mother(d, n). mother(n, t)
ancestor(X,Y):- mother(X,Y).
ancestor(X,Y):- mother(X,Z), ancestor(Z,Y).

$T0 = T_P(\emptyset) = \{mother(a, d), mother(d, n), mother(n, t)\}$
$T1 = T_P(T_P(\emptyset)) = T0 \cup \{ancestor(a, d), ancestor(d, n), ancestor(n, t)\}$
$T2 = T_P(T_P(T_P(\emptyset))) = T1 \cup \{ancestor(a, n), ancestor(d, n)\}$
$T3 = T_P(T_P(T_P(T_P(\emptyset)))) = T2 \cup \{ancestor(a, t)\}$

Stable Models

- Given a program P and a set of atoms S
- P^S is a program obtained from S by
  - Removing all rules in P which contain some *not* a in the body and a $\in$ S
  - Removing all *not a* from the remaining rules
- S is a *stable model* of P if S is the minimal model of the program P^S

Example

Given the program P:
p :- not q.
q :- not p.

S = {q}
P^S consists of a single rule
q.
The minimal model of P^S is {q}. So, {q} is a stable model of P

S = {p}
P^S consists of a single rule
p.
The minimal model of P^S is {p}. So, {p} is a stable model of P

Example

Given the program P:
p :- not q.
q.

S = {q}
P^S consists of a single rule
q.
The minimal model of P^S is {q}. So, {q} is a stable model of P

S = {q, p}
P^S consists of a single rule
q.
The minimal model of P^S is {q}. So, {q, p} is not a stable model of P

Example

Given the program P:
p :- not p.

S = {}
P^S consists of a single rule
p.
The minimal model of P^S is {p}. So, {} is not a stable model of P

S = {p}
P^S is empty.
The minimal model of P^S is {}. So, {p} is not a stable model of P

Entailment

Given a program P and an atom p.
Question: Is p true given P (or: P $\models$ p)?
Answer
- YES if p appears in every stable model of P
- NO if p does not appear in at least on stable model of P

There are systems for computing the stable models.