Datalog

Thursday, April 18, 2019 5:52 PM

- Datalog
  - Another query language for relational model
  - extends relational queries with recursion.

  e.g. Souffle

- Soufflé
  - "sqlite" of Datalog
  - set-based.
- Facts : tuples in database

  Rules : queries
  - data type { number, symbol }

.decl table(name:type) create tables

table(data). insert table.

e.g. Q1(y) :- Movies(x,y,z), z=1940.

order of variables matters

⇔ → name doesn't matter, unless you want to use later.

select

Q1(y) :- Movies(_, y, z), z=1940.

return y

underscore means it's unnecessary.

head (subgoal) atom "AND"

x, z, f : extensional vars

Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, _, 1940).

head variables

body

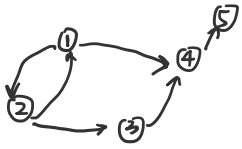
"," ⇔ "AND" (in some DBMS)

- Extensional Database Predicates (EDB) Predicates that are stored in databases.

  Intentional Database Predicates (IDB) Predicates that are expressed by rules.

- Multiple rules for same IDB means OR.

  ✡ e.g. Use graph to express friends. Find all connected friends.



e.g. Find all connected dots.

T(x,y) :- R(x,y). // copy once R

T(x,y) :- R(x,y), T(z,y). // join R & T.

Stop until the new T(x,y) has covered all paths.

- Datalog Semantics

  - Fixpoint Semantic

    Start

    IDB_0 = empty relations

    $t = 0$.

    Repeat

Repeat

IDB_{t+1} = Compute Rules (EDB, IDB_t)

$t = t+1$

Until $IDB_t = IDB_{t-1}$

- Notice that since rules are monotone, $\phi = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \cdots$
- Right linear `T(x,y) :- R(x,z), T(z,y).`
  Left linear `T(x,y) :- T(x,y), R(x,z).`
  Non linear `T(x,y) :- T(x,z), T(z,y).` // will terminate because set semantics.
  - Non linear have least iterations.

- Model of data program
  - a relation instance T is called a model if it satisfies these

$\forall x, y \ R(x,y) \rightarrow T(x,y)$. `T(x,y) :- R(x,y).`

$\forall x, y, z, (R(x,z) \wedge T(z,y)) \rightarrow T(x,y)$. `T(x,y) :- R(x,z), T(z,y).`

$\Leftrightarrow \forall x, y, \exists z$ s.t. $R(x,z) \wedge T(z,y) \rightarrow T(x,y)$

The datalog program computes minimum model.

- Aggregates aggregate_name variable :{ relation to compute aggregate on }

  count, min, max, sum

  e.g. Q(c) :- c = count : { Actor(_, y, _), y = 'John' }

  count will ignore empty group.

  e.g. Q(minId) :- minId = min x : { Actor(x, y, _), y = 'John' }

  assign variable to the value of the aggregate.

- Grouping

  e.g. Q(y, c) :- Movie(_, _, y), c = count : {Movie(_, _, y)}

- Negation "!"

  e.g., find all descendants of Bob that are not of Alice.

  D(x,y) :- ParentChild(x,y).

  D(x,y) :- D(x,y), ParentChild(y,z). // Calculate all descendants

  Q(x) :- D("Bob", x), !D("Alice", x).

e.g. Compute children in the same generation. (two people are in the same gene if they are descendants are at the same gen of common ancestor.

SG(x,y) :- ParentChild(p,x), ParentChild(p,y), x<y // common parent

SG(x,y) :- ParentChild(p,x), ParentChild(q,y), SG(p,q), x<y // parents at same generation

avoid output duplicate

- A datalog rule is **safe** if every variable appears in some **positive, non-aggregated relational atom.**

  e.g. U1(x,y) :- ParentChild("Alice", x), y != "Bob" // infinite ∵ did not specify y.

  U2(x,y) :- ParentChild("Alice", x), !ParentChild(x,y) // did not specify y

  because !
  so y can be any string ⇒ not safe!

- Stratified Datalog
  - Recursion does not cope well with aggregates or negation.

    e.g., A() :- !B();
    B() :- !A();

  - A datalog program is ~~stratified~~ if it can be partitioned into strata.
    syn. Stratum
    - Only IDB predicates defined in strata 1, 2, ..., n may appear under ! or agg in stratum 1.

    (Many datalog DBMS accepts only stratified data)

    e.g.,
    ```
    D(x,y) :- ParentChild(x,y).                    // Stratum I
    D(x,y) :- ParentChild(x,z), D(z,y).
    Q(x) :- D("Alice", x), !("Bob, x).             // Stratum II
    ```

    - If we don't use agg or negation, then the Datalog program is already stratified.

    If we used, it's natural to write a program in stratified way.