Data Integration: Logic Query Languages

Jan Chomicki

University at Buffalo
Datalog

A logic language

- Datalog programs consist of logical facts and rules
- Datalog is a subset of Prolog (no data structures)

Basic concepts

- term: constant, variable
- predicate (relation)
- atom
- clause, rule, fact
- groundness: no variables
- substitution: mapping from variables to terms
- ground substitution: mapping from variables to constants
## Logic programs

### Atom

- **syntax:** \( P(T_1, \ldots, T_n) \)
- **semantics:** predicate \( P \) is **true** of terms \( T_1 \) and \( \ldots \) and \( T_n \)
- **variables present:** truth under a ground substitution

### Implication (clause)

- **syntax:** \( A_0 : - A_1, \ldots, A_k \)
- **semantics:** atom \( A_0 \) is **true** if atoms \( A_1 \) and \( \ldots \) and \( A_k \) are **true**
- **all the variables universally quantified:**
  - implication **true** under **all** ground substitutions
- **all the variables in the head occur also in the body**
- **some predicates in the body may be built-in:** \( >, \geq, \ldots \)

### Kinds of clauses

- \( k = 0 \): **fact**
- \( k > 0 \): **rule** consisting of head \( A_0 \) and body \( A_1, \ldots, A_k \)
Logic query languages

Datalog program $P$

- $EDB(P)$: a set of true ground facts encoding a database instance
- $IDB(P)$: a set of rules encoding a query, with a special predicate `query` to return the result

Predicate dependence

- direct: the predicate in the head depends on each predicate in the body
- indirect: through multiple rules
- recursion: predicate depends on itself

Logic query languages

- conjunctive queries: single rule, no recursion
- unions of conjunctive queries: multiple rules, no recursion
- Datalog: multiple rules, recursion
Example logic program

Friends

%% Facts
friend(joe,sue).
friend(ann,sue).
friend(sue,max).
friend(max,ann).

%% Rules
fof(X,Y) :- friend(X,Y).
fof(X,Z) :- friend(X,Y), fof(Y,Z).

%% Query 1
query(X) :- fof(X,ann).

%% Query 2
query(X) :- fof(X,Y), fof(Y,X).
Deriving facts of $IDB(P)$ predicates bottom-up

1. $EDB(P)$ facts are true
2. Single step: for all the ground substitutions that map the body of a rule in $IDB(P)$ to true facts, make the (substituted) head true
3. Repeat until no new true facts are derived

Derivation properties

- the derivation terminates: why?
- soundness: the derived true facts are logical consequences of $P$
- completeness: all the logical consequences of $P$ are derived

Derived facts: all friend facts and

- % Direct friends
  - fof(joe,sue).
  - fof(ann,sue).
  - fof(sue,max).
  - fof(max,ann).

- % Second-level friends
  - fof(joe,max).
  - fof(sue,ann).
  - fof(ann,max).
  - fof(max,sue).

- % Third-level friends
  - fof(joe,ann).
  - fof(sue,sue).
  - for(ann,ann).
  - for(max,max).
### Open vs. Closed World Assumption

**Closed World Assumption (CWA)**

What is not implied by a logic program is **false**.

**Open World Assumption (OWA)**

What is not implied by a logic program is **unknown**.

### Scope

- traditional database applications: CWA
- information integration: OWA or CWA

Can negation be allowed **inside** Datalog rules?
### Syntax

Rules with negative atoms in the body:

\[ A_0 : -A_1, \ldots, A_k, not\ B_1, \ldots, not\ B_m. \]

### Example

\[ \text{asymmetric}(X,Y):- \ fof(X,Y), \ not\ fof(Y,X). \]

### Semantics

Cannot be adequately given in terms of implication.
Coding relational algebra

Coding operators

- selection, Cartesian product: single clause
- projection: single clause with projected out attributes only in the body
- union: multiple clauses
- set difference: negation

But what about recursion?
Stratification

Dependency graph $pdg(P)$

- vertices: predicates of a Datalog\textsuperscript{not} program $P$
- edges:
  - a positive edge $(p, q)$ if there is a clause in $P$ in which $q$ appears in a positive atom in the body and $p$ appears in the head
  - a negative edge $(p, q)$ if there is a clause in $P$ in which $q$ appears in a negative atom in the body and $p$ appears in the head

Stratified $P$

No cycle in $pdg(P)$ contains a negative edge.

Stratification

Mapping $s$ from the set of predicates in $P$ to nonnegative integers such that:

1. if a positive edge $(p, q)$ is in $pdg(P)$, then $s(p) \geq s(q)$
2. if a negative edge $(p, q)$ is in $pdg(P)$, then $s(p) > s(q)$

There is a polynomial-time algorithm to determine whether a program is stratified, and if it is, to find a stratification for it.
Stratified Datalog\textsuperscript{not}: query evaluation

Bottom-up evaluation

1. compute a stratification of a program $P$
2. partition $P$ into $P_1, \ldots, P_n$ such that
   - each $P_i$ consisting of all and only rules whose head belongs to a single stratum
   - $P_1$ is the lowest stratum
3. evaluate bottom-up $P_1, \ldots, P_n$ (in that order).

Result

- does not depend on the stratification
- can be semantically characterized in various ways
- is used to compute query results
Universal quantification

Coding universal quantification through double negation.

Example

everybodysFriend(X):- person(X), not isNotFriend(X).

isNotFriend(X):- person(X), person(Y), not friend(Y,X).
**Expressiveness**

**Query result**

The query result \( Q(D) \) is defined for every input database \( D \).

**Query containment**

\[ Q_1 \subseteq Q_2 \text{ if and only if } Q_1(D) \subseteq Q_2(D) \text{ for every input database } D. \]

**Query equivalence**

\[ Q_1 \equiv Q_2 \text{ if and only if } Q_1 \subseteq Q_2 \text{ and } Q_2 \subseteq Q_1. \]

**Query language containment**

\( L_1 \subseteq L_2 \text{ if for every query } Q_1 \in L_1, \text{ there is an equivalent query } Q_2 \text{ in } L_2. \)

- \( Q_2 \) may be in a different **syntax** than \( Q_1 \).
Comparing query languages

Expressiveness

- Datalog $\subseteq$ Stratified Datalog\(^{not}\)
- Relational Algebra $\subseteq$ Stratified Datalog\(^{not}\)
- Datalog $\not\subseteq$ Relational Algebra
  - transitive closure
- Relational Algebra $\not\subseteq$ Datalog
  - set difference

How to prove expressiveness results?

- considering the syntax not enough
- semantic properties
A query language $L$ is **monotonic** if for every query $Q \in L$, adding facts to the database cannot remove any tuples from the result of $Q$. Formally: for all databases $D_1$ and $D_2$

$$D_1 \subseteq D_2 \text{ implies } Q(D_1) \subseteq Q(D_2).$$

**Query languages**

- **monotonic**: Datalog
- **nonmonotonic**: Datalog$^{\text{not}}$, relational algebra, SQL