The notion of deductive database

The Datalog query language

Maurizio Lenzerini

Sapienza Università di Roma

A.Y. 2021/22

Expressive power of relational algebra and calculus

We have seen that they have the same expressive power, but which is the expressive power?
We remind the reader that a binary relation on X is transitive if for all elements a, b, c in X, whenever R relates a to b and b to c, then R also relates a to c, i.e., $R(a, b)$ and $R(b, c)$ imply $R(a, c)$.

Definition

The transitive closure of a finite binary relation R on a set X is the smallest binary relation on X that contains R and is transitive.
Here, "smallest" means having the fewest pairs.

Note that if the binary relation R models a graph, then the transitive closure R^* of such binary relation has an edge from a to b if and only if R has a path (of any length greater than 0) from a to b.

Expressive power of relational algebra and calculus

Let us consider the database schema S with a binary relation R. For every database D, the extension of R in D is denoted as $R(D)$.

Theorem

There is no query Q in the relational calculus (and therefore, in relational algebra too) such that, for every database D on the schema S, computes $R(D)^*$, i.e., is such that $Q(D) = R(D)^*$.

This means that the transitive closure is *not* expressible in the relational calculus.

- Note that for every database D, there is a relational calculus query Q such that $Q(D) = R(D)^*$. This is because, given D, we know a bound on the size of the longest path in any relation of D.
- However, the theorem asserts a different property: there is no query Q in relational calculus such that, for every database D we have $Q(D) = R(D)^*$.

Database as a theory

Until now, we have considered a database as an intepretation for a first-order language. In particular,

- the schema of the database determines the predicates in the alphabet, and the active domain of the database determines both the constants in the alphabet and the domain of the interpretation,
- the content of the relations determines the interpretation function.

Now we investigate the view of a database as a theory, i.e., a set of sentences in a first-order language, where

- the database schema S still determines the predicates of the alphabet and the possible values that can appear in the databases coherent with S determines both the constants in the alphabet and the domain of the possible interpretations,
- the content of the relations in a certain database determines the axioms (atomic formulas) specifying the theory formalizing the database.

Deductive databases

By considering a database as a theory, we will be able to define languages in which we can express queries not expressible in the relational calculus.

In particular, the querying mechanisms changes: processing a query does not correspond to evaluating a formula in an interpretation, but rather to performing logical implication.

We are in the world of deductive databases. The basic language for deductive databases that realizes the above mentioned idea is Datalog.

Datalog: introduzione

- Datalog è basato sulla logica del primo ordine senza simboli di funzione, come il calcolo relazionale.
- Il linguaggio include un insieme infinito numerabile *COST* di costanti.
- I termini in Datalog sono o variabli o costanti. Per convezionae, i simboli che iniziano con una maiuscola sono variabili, quelli che iniziano con la minuscola sono costanti.
- Vedremo che l'insieme dei simboli di predicato è partizionato in due insiemi: EPred (extensional predicates) e IPred (intensional predicates).
- In Datalog, l'unità fondamentale è la clausola di Horn, nel senso che una programma Datalog è un insieme di formule, ciascuna essendo una clausola di Horn.

Clausole di Horn al primo ordine

Abbiamo già visto le clausole di Horn nel contesto della logica proposizionale: una clausola è una disgiunzione di literal ed una clausola di Horn è una clausola con al più un literal positivo.

Nel contesto della logica del primo ordine

- un literal in una clausola o è una formula atomica (non banalmente una proposizione come nella logica proposizionale) oppure è la negazione di una formula atomica;
- tutte le variabili di una clausola si intendono quantificate universalmente.

Clausole di Horn al primo ordine

Ad esempio

$$C = (\neg p(X, a) \vee q(Y, b))$$

corrisponde a

$$\forall X \forall Y \ (\neg p(X, a) \vee q(Y, b))$$

Una clausola si scrive anche come l'insieme dei suoi literal. Ad esempio, La clausola C scritta sopra si può scrivere come $\{\neg p(X, a), q(Y, b)\}$.

Definizione

Una clausola si dice

- positiva se non ha literal negativi,
- unit se ha esattamente un literal,
- ground se non vi compaiono variabili.

Clausole di Horn al primo ordine

Ci sono tre tipi di clausole di Horn:

- Fatto: clausole ground con un literal positivo e senza literal negativi. Ad esempio, { padre(mario,aldo) } è un fatto, che si scrive anche semplicemente come
  padre(mario,aldo).
- Regola Horn (o semplicemente regola): clausola con un literal positivo ed almeno uno negativo. Ad esempio: {¬genitore(X), padre(X), ¬uomo(X) } è una regola, che si scrive anche come
  padre(X) :- genitore(X), uomo(x).
  in cui "padre(X)" si dice testa della regola, mentre "genitore(X), uomo(x)" si dice corpo della regola.
  Vale la safety condition: ogni variabile che compare nel literal positivo compare anche in almeno un literal negativo.
- Goal: clausola con un literal negativo e senza literal positivi. Ad esempio {¬padre(mario,X) } è un goal, che si scrive anche come
  ?- padre(mario,X).

Esempi di regole corrette

p(X) :– r(X,Y),s(Y,Z).
regola corretta

q() :– r(X,Y),s(Y,Z).
regola corretta

q(X) :– r(X,Y),q(Y).
regola corretta

q(X) :– r(X,Y),s(Y).
s(X) :- t(X,Y), q(Y).
regole corrette

Osservazione

I due ultimi esempi mostrano che le regole possono essere ricorsive, nel senso che lo stesso predicato può camparire sia nella testa sia nel corpo di una regola o di un insieme di regole.

Esempi di regole non corrette

p(W) :– r(X,Y),s(Y,Z).
regola non corretta: la variabile W non è “safe”

q() :– r(X,Y),s(Y,Z)
regola non corretta: manca il “.” alla fine

t(X,Y,r) :– r(X,Y),s(Y,Z).
regola non corretta: si usa un predicato come argomento di un predicato

Y(X,a,b) :– r(X,Y).
regola non corretta: si usa una variabile in posizione di predicato

I predicati di Datalog: condizioni

Sia S uno schema di basi di dati (simboli di relazioni, ciascuno con la relative arità).

- L'insieme dei predicati EPred corrisponde all'insieme dei simboli in S ed una base di dati coerente con S si può vedere come un insieme di fatti su tali predicati.
- I predicati dell'insieme EPred non possono apparire nelle teste delle regole.
- I predicati dell'insieme IPred non possono apparire nei fatti, mentre devono apparire nella testa di almeno una regola e possono apparire nel corpo delle regole. Siccome questi simboli devono apparire nella testa di almeno una regola, essi si possono considerare come predicati per le viste sulla base di dati coerenti con S.

Programma Datalog: sintassi

Sia S uno schema di basi di dati e $CONST$ un insieme di costanti per le basi di dati coerenti con S.

Definizione

Un programma Datalog su S è un insieme di regole i cui literal sono costruiti sulle costanti in $CONST$ e sui predicati EPred e IPred. Le regole soddisfano le condizioni illustrate nella slide precedente.

Esempio:

EPred (da S_1) = { father/2, mother/2, lives/2, university/1, enrolled/2 }
IPred = { anc/2, person/1, parent/2, student/1 }

Il seguente programma P_1 è un programma Datalog su S_1:

```
person(X) :- father(X,Y).
person(X) :- mother(X,Y).
person(X) :- lives(X,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
student(X) :- person(X), enrolled(X,Y), university(Y).
```

Programma Datalog: semantica del primo ordine

Definition

Se P è un programma Datalog su S, una interpretazione di P è una interpretazione del primo ordine tale che:

- il dominio è $CONST$;
- la funzione di interpretazione interpreta ogni constante come se stessa ed assegna ad ogni predicato di arità n un insieme finito di tuple di n elementi presi da $CONST$.

Come al solito considereremo spesso una interpretazione I come un insieme di fatti, cioè l'insieme delle formule atomiche ground $p(a_1, \ldots, a_n)$ che sono vere in I, (ossia tali che $(a_1, \ldots, a_n) \in p^I(a_1, \ldots, a_n)$).

Programma Datalog: semantica del primo ordine

Definition

Se P è un programma Datalog su S e D è una base di dati coerente con D, una interpretazione I di P è un modello di P e D se

- tutti i fatti di D sono veri in I,
- tutti le regole di P sono soddisfatte da I.

Esempio: base di dati $D_1 = \{$ father(aldo,gioia), mother(gioia,laura) $\}$

person(X) :- father(X,Y).
person(X) :- mother(X,Y).
person(X) :- lives(X,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

modello $I_1 = \{$ father(aldo,gioia), parent(aldo,gioia), mother(gioia,laura), parent(gioia, laura), person(aldo), person(gioia) $\}$

modello $I_2 = \{$ father(aldo,gioia), parent(aldo,gioia), mother(gioia,laura), parent(gioia, laura), person(aldo), person(gioia), lives(laura,roma), person(laura), person(roma) $\}$

non modello $I_3 = \{$ father(aldo,gioia), mother(gioia,laura), person(aldo) $\}$

Query Datalog: sintassi

Definition

Una query Datalog Q su S è una coppia $Q = \langle P, g \rangle$, dove

- P è un programma su S,
- g, detto il "predicato goal" di Q, è uno dei predicati in IPred.

Esempio:
EPred = { father/2, mother/2, lives/2, university/1, enrolled/2 }
D_1 = { father(aldo,gioia), mother(gioia,laura) }
IPred = { anc/2, person/1, parent/1, student/1 }

Una possibile query Q_1 per D_1 è $\langle P_1, \text{person} \rangle$, dove P_1 è il programma visto prima.

Query Datalog: semantica

Il significato intuitivo di una query $Q = \langle P, g \rangle$, dove l'arità di g è n, è il seguente: data una base di dati D, voglio tutte le tuple $\langle a_1, \ldots, a_n \rangle$ di costanti in $CONST$ tali che i fatti rappresentati da D ed il programma Datalog P implicano logicamente $g(a_1, \ldots, a_n)$, ossia tali che $g(a_1, \ldots, a_n)$ è vero in tutti i modelli di P e D.

Definition

La valutazione di una query Datalog $Q = \langle P, g \rangle$ rispetto ad una base di dati D, dove g ha arità n, è l'insieme Q^D:

$$\{(a_1, \ldots, a_n) \mid (P \cup D) \models g(a_1, \ldots, a_n)\}$$

Esempio:
Per la query Q_1 vista prima e la relativa base di dati D_1 si ha

$$P_1^{D_1} = \{aldo, gioia\}$$

Decidibilità del problema di valutare una query Datalog

Ma è decidibile il problema di calcolare l'insieme $\{(a_1, \ldots, a_n) \mid (P \cup D) \models g(a_1, \ldots, a_n)\}$?

Teorema

Se $Q = \langle P, g \rangle$ è una query Datalog sullo schema S e D è una base di dati coerente con S, allora $(P \cup D) \models g(a_1, \ldots, a_n)$ se e solo se $g(a_1, \ldots, a_n)$ è vero nella intersezione di tutti i modelli di $P \cup D$.

Teorema

Se P è un programma Datalog sullo schema S e D è una base di dati coerente con S, allora l'intersezione di tutti i modelli di $P \cup D$ è l'unico modello minimo di $P \cup D$, che denotiamo con $MM(P, D)$, cioè è un modello tale che non esiste alcun modello I di $P \cup D$ tale che $I \subset MM(P, D)$.

Corollario

Se $Q = \langle P, g \rangle$ è una query Datalog e D è una base di dati, allora $Q^D = \{(a_1, \ldots, a_n) \mid g(a_1, \ldots, a_n) \in MM(P, D)\}$

Grounding di un programma

Daremo un metodo per calcolare il modello minimo di un programma Datalog ed una base di dati.

Sia P un programma Datalog, D una base di dati e γ il sottoinsieme di $CONST$ che contiene tutte le costanti in D e P.

- Una (P, D)-istanziazione di variabili per una regola r di P rispetto a D è una funzione che assegna ad ogni variabile di r una costante in γ.
- Se μ è una (P, D)-istanziazione di variabili per la regola r in P rispetto a D, denotiamo con $\mu(r)$ la regola ground ottenuta sostituendo ogni occorrenza di variabile X in r con $\mu(X)$. La regola ground $\mu(r)$ si chiama (P, D)-istanza di r.
- Il (P, D)-grounding di r, che denotiamo con $ground(r, P, D)$ è l'insieme di tutte le (P, D)-istanze $\mu(r)$.
- Il D-grounding di P, che denotiamo con $ground(P, D)$ è il programma Datalog ground ottenuto come unione di tutti gli insiemi $ground(r, P, D)$ tali che r è una regola di P.

L'operatore di conseguenza immediata

Definizione

Dato un programma Datalog ground P e la base di dati D, l'operatore di conseguenza immediata per P e D, che denotiamo con $T_{P,D}$ è la funzione che assegna ad ogni interpretazione I di P tale che $D \in I$ un insieme di atomi ground secondo questa definizione:

$$T_{P,D}(I) = \{\alpha \mid \text{esiste } \alpha \text{ :- } \beta_1, \ldots, \beta_n \text{ in } P \text{ tale che } \{\beta_1, \ldots, \beta_n\} \subseteq I\}$$

Definizione

Il minimo punto fisso (least fixed point) della funzione $T_{P,D}$ è la minima interpretazione I tale che $D \in I$ e $T_{P,D}(I) = I$.

Teorema

Se P è un programma Datalog ground e D è una base di dati, allora l'operatore $T_{P,D}$ di conseguenza immediata per P, D ha un unico minimo punto fisso che coincide con il modello minimo $MM(P, D)$ di $P \cup D$.

Come calcolare il modello minimo di un programma Datalog

Resta a questo punto di vedere come si calcola il modello minimo di un programma Datalog P e di una base di dati D. Ci sono diversi algoritmi per questo calcolo, e noi illustreremo quello più semplice, chiamato "naive evaluation algorithm", che di fatto calcola il minimo punto fisso di $T_{P,D}$.

1. $P' \leftarrow ground(P, D)$
2. esegui le seguenti operazioni:
   1. $I \leftarrow D$
   2. repeat $J \leftarrow I$; $I \leftarrow T_{P',D}(J)$ until $I = J$
   3. return I

Teorema

Se P è un programma Datalog e D è una base di dati, allora il naive evaluation algorithm eseguito su P e D termina restituendo $MM(P, D)$.

Valutazione di query Datalog e complessità

L'algoritmo di valutazione di una query Datalog è a questo punto semplicissimo. Data una query $Q = \langle P, g \rangle$ ed una base di dati D,

- calcola $MM(P, D)$ mediante il naive evaluation algorithm su P e D
- restituisci $\{(a_1, \ldots, a_n) \mid g(a_1, \ldots, a_n) \in MM(P, D)\}$

Teorema

- La data complexity di Datalog è PTIME.
- La combined complexity di Datalog è EXPTIME.
- Se P è un programma Datalog per la base di dati D, il problema di verificare se per una tupla t sia ha $t \in Q^D$ è EXPTIME-completo.

Datalog: espressività

Teorema

Esiste una query Datalog che, per ogni base di dati D, calcola la chiusura transitiva di R, dove R è una relazione binaria in D. Quindi l'algebra relazionale (ed anche il calcolo relazionale) non è più espressiva di Datalog.

Teorema

Datalog è più espressivo dell'algebra relazionale senza l'operatore di differenza.

Teorema

Esiste una query che si può esprimere nell'algebra relazionale ma non in Datalog. Quindi Datalog non è più espressivo dell'algebra relazionale e non è più espressivo del calcolo relazionale.

Datalog: espressività

Sia R una relazione binaria in uno schema S. Sia P_t il seguente programma Datalog sullo schema S:

$ct(X, Y) :- R(X, Y).$
$ct(X, Y) :- R(X, Z), ct(Z, Y).$

Consideriamo ora la query $Q = \langle P_t, ct \rangle$. È immediato verificare che, qualunque sia la base di dati D sullo schema S, si ha che $Q^D = R(D)^*$, ovvero $\langle P, ct \rangle$ eseguita sulla base di dati D calcola la chiusura transitiva di $R(D)$.

Datalog: containment ed equivalence

Teorema

- Verificare se due query Datalog sono equivalenti è un problema indecidibile, anche per query Datalog che usano un solo predicato binario in IPred.
- Anche verificare se una query Datalog è contenuta in un'altra è un problema indecidibile, anche per query Datalog che usano un solo predicato binario in IPred.

Suggerimento per la dimostrazione: riduzione al problema di verificare se due grammatiche context-free sono equivalenti.