A Zoo of Query Languages: Datalog, UCQ, CQ...

Jef Wijsen

March 1, 2024

Outline

Database

A relational database instance I will be represented by a set I of *facts*.

Knows	1	2
	Jeb	Don
	Don	Jeb
	An	Don
	Ed	An
	Jo	Ed

Owns	1	2
	Don	iPad
	Don	iPod
	Jeb	iPod

$\rightsquigarrow$

$$I = \{\mathit{Knows}(\mathrm{Jeb}, \mathrm{Don}), \mathit{Knows}(\mathrm{Don}, \mathrm{Jeb}), \ldots, \mathit{Owns}(\mathrm{Jeb}, \mathrm{iPod})\}$$

Knows and *Owns* are extensional database (edb) predicates.

Deductive Databases

The following rule defines the *Happy* view.

$$Happy(x) \quad \leftarrow \quad Owns(x, \mathrm{iPad}), Owns(x, \mathrm{iPod})$$

Happy is an intentional database (idb) predicate.

With this rule, the intentional database contains $Happy(\mathrm{Don})$, but not $Happy(\mathrm{Jeb})$.

Multiple Rules

$$\begin{array}{lcl} \mathit{Happy}(x) & \leftarrow & \mathit{Owns}(x, \mathrm{iPad}), \mathit{Owns}(x, \mathrm{iPod}) \\ \mathit{Happy}(x) & \leftarrow & \mathit{Owns}(x, \mathrm{iPad}) \\ \mathit{Happy}(x) & \leftarrow & \mathit{Owns}(x, \mathrm{iPod}) \end{array}$$

With these rules, the intentional database contains $\mathit{Happy}(\mathrm{Don})$ and $\mathit{Happy}(\mathrm{Jeb})$.

The first rule is redundant.

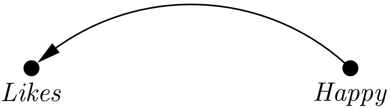
Composition

$$
\begin{aligned}
\mathit{Likes}(x, y) &\leftarrow \mathit{Knows}(x, y), \mathit{Owns}(y, \mathrm{iPad}) \\
\mathit{Likes}(x, y) &\leftarrow \mathit{Knows}(x, y), \mathit{Owns}(y, \mathrm{iPod}) \\
\mathit{Happy}(y) &\leftarrow \mathit{Likes}(x, y)
\end{aligned}
$$

The first two rules state that people like every person they know who has an iPad or an iPod. The third rule states that you are happy if someone likes you.

With these rules, the intentional database contains, among others:

- $\mathit{Likes}(\mathrm{Jeb}, \mathrm{Don})$ because Jeb knows Don, and Don owns an iPad;
- $\mathit{Happy}(\mathrm{Don})$ because Jeb likes Don.



Recursion

$$\begin{aligned} \mathit{Happy}(x) &\leftarrow \mathit{Owns}(x, \mathrm{iPad}) \\ \mathit{Happy}(x) &\leftarrow \mathit{Knows}(x, y), \mathit{Happy}(y) \end{aligned}$$

The second rule says that knowing happy people makes you happy.

With these rules, the intentional database contains, among others:

- *Happy*(Don) because Don owns an iPad;
- *Happy*(An) because An knows Don, and Don is happy;
- *Happy*(Ed) because Ed knows An, and An is happy;
- *Happy*(Jo) because Jo knows Ed, and Ed is happy.

Safe negation of edb predicates

$$Unhappy(x) \leftarrow Knows(x, y), Owns(y, z), \neg Owns(x, z)$$

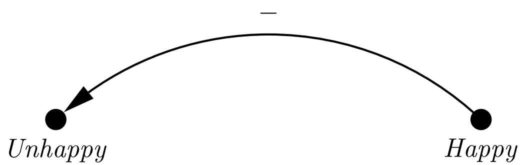
The safety requirement states that every variable that occurs in a rule, must occur positively in the body of the rule (i.e., the part of the rule that occurs at the right of $\leftarrow$).

The following two rules are **not** safe (so they are syntactically **incorrect**), because z occurs in the rule but z does not occur positively in the body:

$$\begin{aligned} R(x, z) &\leftarrow Knows(x, y) \\ S(x) &\leftarrow Knows(x, y), \neg Owns(x, z) \end{aligned}$$

Safe negation of idb predicates

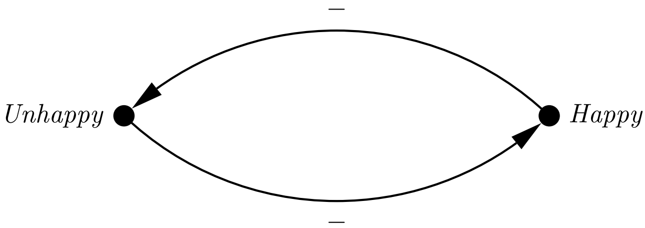
$$\begin{aligned} Unhappy(x) &\leftarrow Knows(x,y), Owns(y,z), \neg Owns(x,z) \\ Happy(x) &\leftarrow Owns(x,y), \neg Unhappy(x) \end{aligned}$$



Safe negation of idb predicates

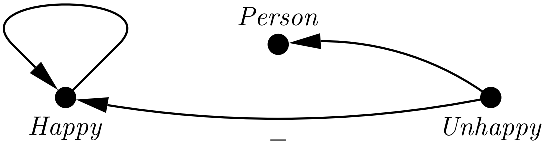
The following program is syntactically correct but meaningless:

$$\begin{aligned} \mathit{Unhappy}(x) &\leftarrow \mathit{Owns}(x,y), \neg \mathit{Happy}(x) \\ \mathit{Happy}(x) &\leftarrow \mathit{Owns}(x,y), \neg \mathit{Unhappy}(x) \end{aligned}$$



Recursion and Negation

$$
\begin{aligned}
\mathit{Person}(x) &\leftarrow \mathit{Knows}(x, y) \\
\mathit{Person}(y) &\leftarrow \mathit{Knows}(x, y) \\
\mathit{Person}(x) &\leftarrow \mathit{Owns}(x, y) \\
\mathit{Happy}(x) &\leftarrow \mathit{Owns}(x, \mathrm{iPad}) \\
\mathit{Happy}(x) &\leftarrow \mathit{Knows}(x, y), \mathit{Happy}(y) \\
\mathit{Unhappy}(x) &\leftarrow \mathit{Person}(x), \neg \mathit{Happy}(x)
\end{aligned}
$$



Exercise

Get owners who own everything that can be owned.

$$\begin{array}{rcl} Owner(x) & \leftarrow & Owns(x,y) \\ MissingSomething(x) & \leftarrow & Owner(x), Owns(u,z), \neg Owns(x,z) \\ Answer(x) & \leftarrow & Owner(x), \neg MissingSomething(x) \end{array}$$

In relational calculus:

$$\{x \mid \exists y\,(Owns(x,y)) \wedge \forall u \forall z\,(Owns(u,z) \rightarrow Owns(x,z))\}$$

In relational algebra (if the schema is $Owns[A,B]$):

$$\pi_A(Owns) - \pi_A((\pi_A(Owns) \bowtie \pi_B(Owns)) - Owns)$$

Translating Relational Calculus into Rules

- Variables present in the body of a rule, yet absent in its head, are existentially quantified.
- We can first rewrite $\forall \vec{v}\,(\varphi(\vec{v}))$ as $\neg\exists \vec{v}\,(\neg\varphi(\vec{v}))$.

For example,

$$\{x \mid \exists y\,(Owns(x,y)) \wedge \forall u \forall z\,(Owns(u,z) \rightarrow Owns(x,z))\}$$

is equivalent to:

$$\{x \mid \overbrace{\exists y\,(Owns(x,y))}^{Owner(x)} \wedge \neg \overbrace{\exists u \exists z\,(Owns(u,z) \wedge \neg Owns(x,z))}^{MissingSomething(x)}\}.$$

Then, the atom $Owner(x)$ is needed in the third rule below to ensure the rule's safety:

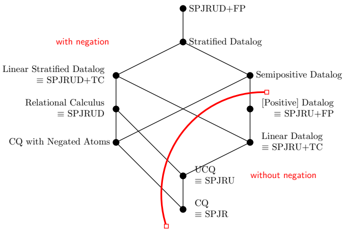
$$\begin{aligned}
Answer(x) &\leftarrow Owner(x), \neg MissingSomething(x)\\
Owner(x) &\leftarrow Owns(x,y)\\
MissingSomething(x) &\leftarrow Owner(x), Owns(u,z), \neg Owns(x,z)
\end{aligned}$$

Outline

Language Design and Reasoning About Programs

Language design:

- restrict how rules, negation, and recursion can be used and combined; and
- define a precise semantics.



Reasoning about programs (a.k.a. queries):

- Is there an algorithm for simplifying a given program P (i.e., for constructing a "shorter" program that is equivalent to P)?

Containment ($\sqsubseteq$) and equivalence ($\equiv$) of queries

Let q_1, q_2 be two queries in some query language $\mathcal{L}$ (e.g., $\mathcal{L} =$ relational calculus or $\mathcal{L} =$ SPJR algebra).

We write $q_1 \equiv q_2$ if for every database I,

$$q_1(I) = q_2(I).$$

We write $q_1 \sqsubseteq q_2$ if for every database I,

$$q_1(I) \subseteq q_2(I).$$

Note:

- $q_1(I)$ denotes the answer of q_1 on database I; and
- $q_1(\vec{x})$ denotes that $\vec{x}$ is the sequence of free variables of q_1.

Problems

Let $\mathcal{L}$ be a query language.

- The containment problem for $\mathcal{L}$ is the following: Given two queries $q_1, q_2 \in \mathcal{L}$, decide whether $q_1 \sqsubseteq q_2$.
- The equivalence problem for $\mathcal{L}$ is the following: Given two queries $q_1, q_2 \in \mathcal{L}$, decide whether $q_1 \equiv q_2$.
- The satisfiability problem for $\mathcal{L}$ is the following: Given $q \in \mathcal{L}$, is there a database I such that $q(I) \neq \emptyset$?

These problems are related:

$$
\begin{aligned}
q_1(\vec{x}) \sqsubseteq q_2(\vec{x}) \quad &\iff \quad q_1 \equiv q_1 \wedge q_2 \\
q_1(\vec{x}) \equiv q_2(\vec{x}) \quad &\iff \quad (q_1 \wedge \neg q_2) \vee (q_2 \wedge \neg q_1) \text{ is not satisfiable}
\end{aligned}
$$

That is, the containment problem can be "reduced" to the equivalence problem, provided that $\mathcal{L}$ is closed under $\wedge$.
The equivalence problem can be "reduced" to the complement of the satisfiability problem, provided that $\mathcal{L}$ is closed under $\wedge$, $\vee$, and $\neg$.

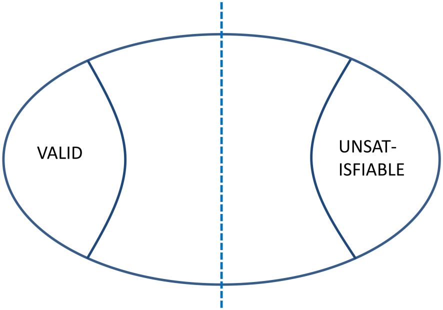
Undecidability

Theorem

1. *The containment problem for relational calculus is undecidable.*
2. *The equivalence problem for relational calculus is undecidable.*
3. *The satisfiability problem for relational calculus is undecidable.*

This is different from the undecidability of the Entscheidungsproblem [Tur36] because database instances are finite, whereas in conventional predicate calculus, both finite and infinite structures are considered.

The Geography of First-Order Sentences (inspired by [Pap94])



- $\exists x \, (P(x) \wedge \neg P(x))$ is unsatisfiable
- $\neg \exists x \, (P(x) \wedge \neg P(x))$ is valid
- $\exists x (P(x))$ is satisfiable and not valid

Negation can be thought of as "flipping" of the figure around its vertical axis of symmetry.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, Trakhtenbrot, Boris (1950). *The Impossibility of an Algorithm for the Decidability Problem on Finite Classes*. Proceedings of the USSR Academy of Sciences (in Russian). 70 (4): 569–572.

Theorem (Trakhtenbrot's theorem)

The following problem is undecidable: Given a first-order logic sentence φ, is there a finite model (i.e., a database) that satisfies φ?

Finite and Unrestricted Satisfiability

$$\begin{aligned}
\sigma_1 &= \forall x \forall y \forall z \, ((R(x,y) \wedge R(x,z)) \rightarrow y = z) \\
\sigma_2 &= \neg \exists w \exists z \, (R(w,z) \wedge \neg \exists z' \, (R(z',w))) \\
\sigma_3 &= \exists z \exists w \, (R(z,w) \wedge \neg \exists z' \, (R(w,z')))
\end{aligned}$$

1. No constant occurs more than once in the first column.
2. Every constant in the first column also occurs in the second column.
3. Some constant c in the second column does not occur in the first column.

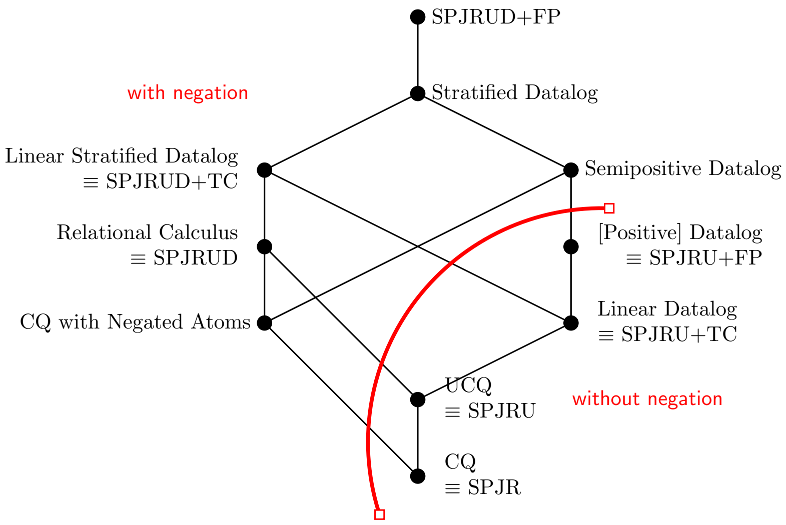
The formula $\sigma_1 \wedge \sigma_2 \wedge \sigma_3$ cannot be satisfied by a finite database, but is satisfied by the infinite structure shown next.

$$\begin{array}{c|cc}
R & 1 & 2 \\
\hline
 & 0 & c \\
 & 1 & 0 \\
 & 2 & 1 \\
 & \multicolumn{2}{c}{\vdots}
\end{array}$$

Languages

- Conjunctive queries (single nonrecursive rule)
- Unions of conjunctive queries (a family of conjunctive queries with the same head predicate)
- Conjunctive queries with atomic negation
- Nonrecursive queries with negation $=$ relational calculus
- Recursive queries without negation $=$ datalog
- Datalog with stratified negation

Overview



SPJRUD+FP
with negation
Stratified Datalog
Linear Stratified Datalog
≡ SPJRUD+TC
Semipositive Datalog
Relational Calculus
≡ SPJRUD
[Positive] Datalog
≡ SPJRU+FP
CQ with Negated Atoms
Linear Datalog
≡ SPJRU+TC
UCQ
≡ SPJRU
without negation
CQ
≡ SPJR

Outline

Datalog Syntax

Read A Datalog Primer.

A set of rules without negation.

Immediate Consequence Operator T_P

$$P : \left\{ \begin{array}{lcl} Path(x,y) & \leftarrow & R(x,y) \\ Path(x,z) & \leftarrow & Path(x,y), R(y,z) \end{array} \right.$$

$$J = \overbrace{\{\underbrace{R(a,b), R(b,c), R(c,d), R(d,e)}_{\text{edb}}, \underbrace{Path(a,c), Path(c,e)}_{\text{idb}}\}}^{\text{deductive database}}$$

$$T_P(J) = \left\{ \begin{array}{l} \overbrace{R(a,b), R(b,c), R(c,d), R(d,e),}^{\text{copy of edb}} \\ \overbrace{Path(a,b), Path(b,c), Path(c,d), Path(d,e),}^{\text{1st rule}} \\ \overbrace{Path(a,d)}^{\text{2nd rule}} \end{array} \right\}$$

Fixpoint of the Immediate Consequence Operator T_P

$$P : \left\{ \begin{array}{lcl} Path(x,y) & \leftarrow & R(x,y) \\ Path(x,z) & \leftarrow & Path(x,y), R(y,z) \end{array} \right.$$

$$J = \left\{ \begin{array}{l} R(a,b), R(b,c), R(c,d), R(d,e), \\ Path(a,b), Path(a,c), Path(a,d), Path(a,e), \\ Path(b,c), Path(b,d), Path(b,e), \\ Path(c,d), Path(c,e), Path(d,e) \end{array} \right\}$$

$$T_P(J) = J$$

We define datalog semantics in a non-procedural way, as follows:

Definition

Given a database instance I (i.e., a set of edb facts), the answer to a datalog program P is the smallest fixpoint of T_P that includes I.

Datalog Semantics

Let P be a datalog program.

- We use the term deductive database for a set of facts that can use both edb and idb predicates.
- The immediate consequence operator T_P maps each deductive database J to the deductive database $T_P(J)$ satisfying
  1. $T_P(J)$ "copies" all edb facts of J;
  2. $T_P(J)$ contains all idb facts that can be derived from J by executing once every rule of P; and
  3. no other facts belong to $T_P(J)$.
- Given an edb database I, the answer $P(I)$ is defined as the (unique) smallest (w.r.t. $\subseteq$) deductive database J such that $I \subseteq J$ and $T_P(J) = J$. *That is, the answer is the smallest fixed point of T_P that includes I.*

  Intuition: Accept all and only those idb facts that are supported by the rules.

"Couldn't there be two smallest fixed points, J_1 and J_2, both including I, such that $J_1 \not\subseteq J_2$ and $J_2 \not\subseteq J_1$?"

Properties of the Immediate Consequence Operator T_P

Lemma (Monotonicity)

Let P be a datalog program (without negation). Let J_1 and J_2 be deductive databases. If $J_1 \subseteq J_2$, then $T_P(J_1) \subseteq T_P(J_2)$.

Lemma (Smallest fixed point)

Let P be a datalog program (without negation). Let I be a set of edb facts. There is a fixed point of T_P that (i) includes I, and (ii) is a subset of any other fixed point of T_P that includes I.

Proof.

See next slide. $\square$

Proof that there is a unique smallest fixed point

Proof. Let J be any other fixed point of T_P that includes I. That is $I \subseteq T_P(J) = J$.

$$\begin{array}{rcll} I & \subseteq & J & (\textit{Given.}) \\ T_P(I) & \subseteq & T_P(J) = J & (\textit{Monotonicity.}) \\ T_P^2(I) := T_P(T_P(I)) & \subseteq & T_P(T_P(J)) = J & (\textit{Monotonicity.}) \\ T_P^3(I) := T_P(T_P(T_P(I))) & \subseteq & T_P(T_P(T_P(J))) = J & (\textit{Monotonicity.}) \\ & \text{etc.} & & \end{array}$$

$$\begin{array}{rcll} I & \subseteq & T_P(I) & (T_P \text{ copies all edb facts.}) \\ T_P(I) & \subseteq & T_P(T_P(I)) & (\text{Monotonicity.}) \\ T_P(T_P(I)) & \subseteq & T_P(T_P(T_P(I))) & (\text{Monotonicity.}) \\ & \text{etc.} & & \end{array}$$

We must reach n such that $T_P^n(I) = T_P^{n+1}(I)$, because there are only finitely many facts that can be added. Consequently,

1. $T_P^n(I)$ is a fixed point;
2. $T_P^n(I)$ includes I; and
3. $T_P^n(I) \subseteq J$. □

Note: the proof tells us how to construct the smallest fixed point!

Immediate Consequence Operator: Example

Let P contain two rules:

$$\begin{array}{rcl} A(x,y) & \leftarrow & R(x,y) \\ A(x,y) & \leftarrow & R(x,z), A(z,y) \end{array}$$

Let

$$J = \{R(1,2), R(2,3), A(2,5)\}.$$

Then

$$\begin{array}{rcl} T_P(J) & = & \{R(1,2), R(2,3), A(1,2), A(2,3), A(1,5)\} \\ T_P(T_P(J)) & = & \{R(1,2), R(2,3), A(1,2), A(2,3), A(1,3)\} \\ T_P(T_P(T_P(J))) & = & T_P(T_P(J)) \end{array}$$

Note that monotonicity does **not** imply $J \subseteq T_P(J)$. Indeed, in the preceding example, $J \nsubseteq T_P(J)$ and $T_P(J) \nsubseteq T_P(T_P(J))$.

Multiple Fixpoints

Let P contain one rule:

$$A(x) \leftarrow R(x), A(x)$$

Let

$$\begin{aligned} I &= \{R(a)\}; \\ J_1 &= \{R(a)\}; \\ J_2 &= \{R(a), A(a)\}. \end{aligned}$$

Then,

$$\begin{aligned} T_P(J_1) &= J_1; \\ T_P(J_2) &= J_2. \end{aligned}$$

Undecidability

Theorem

- *The containment problem for datalog is undecidable.*
- *The equivalence problem for datalog is undecidable.*

Outline

Syntax of Datalog with Stratified Negation

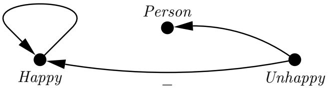
- a set of safe rules such that
- the program dependence graph (PDG) contains no cycle with a negated edge

Semantics of Datalog with Stratified Negation

- The stratum of an idb predicate S is the greatest number of negated edges on any path in the PDG that starts from S.
- Since the PDG contains no cycle with a negated edge, the stratum of an idb predicate cannot be $+\infty$.
- Evaluate the idb predicates "lowest-stratum-first." Once an idb predicate has been evaluated, it is treated as an edb predicate for higher strata.

Recursion and Negation

$$
\begin{array}{rcl}
\mathit{Person}(x) & \leftarrow & \mathit{Knows}(x, y) \\
\mathit{Person}(y) & \leftarrow & \mathit{Knows}(x, y) \\
\mathit{Person}(x) & \leftarrow & \mathit{Owns}(x, y) \\
\mathit{Unhappy}(x) & \leftarrow & \mathit{Person}(x), \neg \mathit{Happy}(x) \\
\mathit{Happy}(x) & \leftarrow & \mathit{Owns}(x, \mathrm{iPad}) \\
\mathit{Happy}(x) & \leftarrow & \mathit{Knows}(x, y), \mathit{Happy}(y)
\end{array}
$$



- *Person* and *Happy* have stratum 0; *Unhappy* has stratum 1.
- First, we evaluate the rules for *Person* and *Happy*. Then, we evaluate the rule for *Unhappy*, treating *Person* and *Happy* as edb predicates.

The Barber Paradox

There is a male village barber who shaves **all and only those men** in the village who do not shave themselves.
Does the barber shave himself?

$$\mathit{Shaves}(\mathrm{Barber}, x) \quad \leftarrow \quad \mathit{Male}(x), \neg \mathit{Shaves}(x, x)$$

The negation in this program is not stratified.

Let $I = \{\mathit{Male}(\mathrm{Barber})\}$.

What happens if we try fixpoint semantics?

$$\begin{aligned} T_P(I) &= \{\mathit{Male}(\mathrm{Barber}), \mathit{Shaves}(\mathrm{Barber}, \mathrm{Barber})\} \\ T_P(T_P(I)) &= \{\mathit{Male}(\mathrm{Barber})\} \end{aligned}$$

There is no fixpoint.

Semipositive Datalog

- Semipositive Datalog = Datalog + negation that applies only on edb predicates
- Thus, the PDG contains no negative edges.
- Semipositive Datalog can express some queries that are neither in the relational calculus nor in Datalog.
- Semipositive Datalog cannot express universal quantification.

Stratified Datalog (defined without using PDG)

A stratified Datalog program is a sequence $P = (P_0, \ldots, P_r)$ of basic Datalog programs, which are called the strata of P, such that each of the IDB predicates of P is an IDB predicate of precisely one stratum P_i and can be used as an EDB predicate (but not as an IDB predicate) in higher strata P_j where $j > i$. In particular, this means that

1. if an IDB predicate of stratum P_j occurs positively in the body of a rule of stratum P_i, then $j \leq i$, and
2. if an IDB predicate of stratum P_j occurs negatively in the body of a rule of stratum P_i, then $j < i$.

Stratified Datalog programs are given natural semantics using semantics for Datalog programs for each P_i, where the IDB predicates of a lower stratum are viewed as EDB predicates for a higher stratum.

In other words, each program slice P_i is a semipositive Datalog program relative to IDB predicates of lower strata P_j, $j < i$.
A rule is recursive if its body contains an IDB predicate of the same stratum.

Multiple Stratifications

$$\left\{\begin{array}{lcl} R(x) & \leftarrow & A(x), \neg S(x) \\ S(x) & \leftarrow & B(x) \\ T(x) & \leftarrow & S(x) \end{array}\right.$$

The stratification found by the PDG is

$$\left(P_0 : \left\{\begin{array}{lcl} S(x) & \leftarrow & B(x) \\ T(x) & \leftarrow & S(x) \end{array}\right., P_1 : \left\{\begin{array}{lcl} R(x) & \leftarrow & A(x), \neg S(x) \end{array}\right. \right).$$

Another stratification is:

$$\left(P_0 : \left\{\begin{array}{lcl} S(x) & \leftarrow & B(x) \end{array}\right., P_1 : \left\{\begin{array}{lcl} R(x) & \leftarrow & A(x), \neg S(x) \\ T(x) & \leftarrow & S(x) \end{array}\right. \right).$$

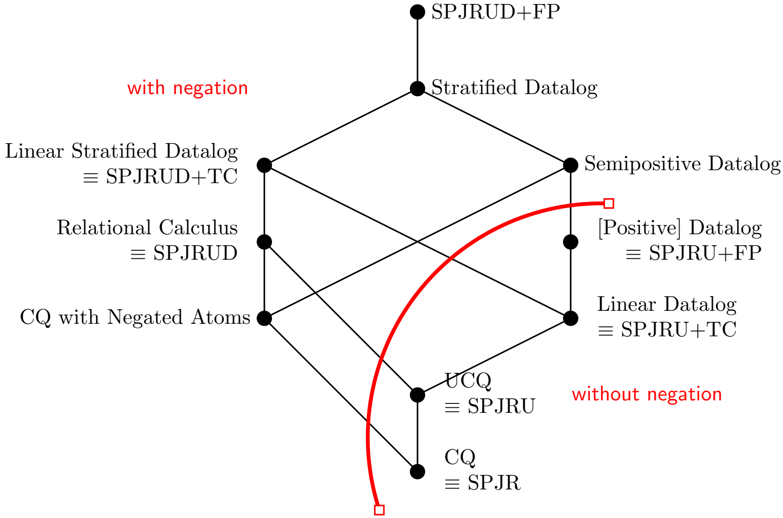
It is known that all stratifications are equivalent.

Datalog and Prolog

- Datalog semantics does *not*, repeat *not*, depend on the order in which the rules are stated.
- Cite from [BBS06, p. 47]

  > *"But Prolog is* not, *repeat* not, *a full logic programming language. If you only think about the declarative meaning of a Prolog program, you are in for a very tough time."*

Overview


SPJRUD+FP
with negation
Stratified Datalog
Linear Stratified Datalog
≡ SPJRUD+TC
Semipositive Datalog
Relational Calculus
≡ SPJRUD
[Positive] Datalog
≡ SPJRU+FP
CQ with Negated Atoms
Linear Datalog
≡ SPJRU+TC
UCQ
≡ SPJRU
without negation
CQ
≡ SPJR

Outline

Linear Stratified Datalog

[Following up on a question by a student in 2017.]
A linear program for transitive closure:

$$\begin{array}{rcl} Trans(x,y) & \leftarrow & Knows(x,y) \\ Trans(x,y) & \leftarrow & Knows(x,z), Trans(z,y) \end{array}$$

A nonlinear program for transitive closure:

$$\begin{array}{rcl} Trans(x,y) & \leftarrow & Knows(x,y) \\ Trans(x,y) & \leftarrow & Trans(x,z), Trans(z,y) \end{array}$$

- Two predicates R and R' are mutually recursive if $R = R'$ or R and R' participate in the same cycle of the PDG.
- A rule with head predicate R is linear if there is at most one atom in the body of the rule whose predicate is mutually recursive with R.
  Note: this allows more than one idb predicate in the body.
- A program is linear if each rule in it is linear.

Transitive Closure in SQL

$$\begin{aligned} \mathit{Trans}(x, y) &\leftarrow \mathit{Knows}(x, y) \\ \mathit{Trans}(x, y) &\leftarrow \mathit{Trans}(x, z), \mathit{Knows}(z, y) \\ \mathit{Ans}(y) &\leftarrow \mathit{Trans}(\mathrm{Jo}, y) \end{aligned}$$

Assume that the schema of *Knows* is $[A, B]$

```
WITH RECURSIVE Trans(A',B') AS
          ( (SELECT A as A', B as B' FROM Knows)
              UNION
            (SELECT Trans.A', Knows.B as B'
              FROM Trans, Knows
              WHERE Trans.B' = Knows.A) )
SELECT B' FROM Trans WHERE A'= "Jo"
```

Extending Relational Calculus with Transitive Closure

1. Every formula in relational calculus is a formula in Transitive Closure Logic (TC).
2. If $\varphi(x, y, z)$ is a formula in TC with free variables x, y, z, then

   $$[\mathbf{tcl}_{x,y}\varphi(x, y, z)](x', y')$$

   is a formula in TC with free variables x', y', z.

The semantics is as follows. For any fixed value c for z,

$$[\mathbf{tcl}_{x,y}\varphi(x, y, c)](a, b)$$

evaluates to true on a database if (a, b) is in the transitive closure of the answer to the query $\{x, y \mid \varphi(x, y, c)\}$.
[In general, x, y, z can be sequences of variables.]

Mimicking $[\mathbf{tcl}_{x,y}\varphi(x, y, z)](x', y')$ in Datalog

$$
\begin{array}{rcl}
\mathit{Trans}(x, y, z) & \leftarrow & \varphi(x, y, z) \\
\mathit{Trans}(x, y, z) & \leftarrow & \mathit{Trans}(x, u, z), \mathit{Trans}(u, y, z) \\
\mathit{Answer}(x', y') & \leftarrow & \mathit{Trans}(x', y', z)
\end{array}
$$

In Other Words...

`[slide added for completeness]`

$$[\mathbf{tcl}_{x,y}\varphi(x,y,z)](x',y')$$

is the same as

$$[\mathbf{fp}_{\Delta:x,y,z}\,(\varphi(x,y,z) \vee \exists w\,(\varphi(x,w,z) \wedge \Delta(w,y,z)))](x',y',z)$$

See Sections 6 and 7 of “Adding Recursion to SPJRUD”

Example: Graph Connectivity

Let the binary relation E encode the directed edges of a graph, i.e., $E(a, b)$ holds true if there is a directed edge from a to b.
Is the undirected graph associated with E (obtained by ignoring the directions of the edges) connected?

$$\forall u \forall v \, (\nu(u) \wedge \nu(v) \rightarrow [\mathbf{tcl}_{x,y} E(x, y) \vee E(y, x)](u, v))$$

where $\nu(z)$ is a syntactic shorthand for "z is a vertex":

$$\nu(z) := \exists w \, (E(z, w) \vee E(w, z))$$

In linear stratified Datalog:

$$\begin{array}{rcl}
Adjacent(x, y) & \leftarrow & E(x, y) \\
Adjacent(x, y) & \leftarrow & E(y, x) \\
Trans(u, v) & \leftarrow & Adjacent(u, v) \\
Trans(u, v) & \leftarrow & Adjacent(u, w), Trans(w, v) \\
V(x) & \leftarrow & Adjacent(x, y) \\
Disconnected() & \leftarrow & V(u), V(v), \neg Trans(u, v) \\
Connected() & \leftarrow & \neg Disconnected()
\end{array}$$

Expressiveness and Complexity

Fact

Linear stratified Datalog is equivalent to Transitive Closure Logic.

Intuitively,

$$\text{Linear stratified Datalog} = \text{relational algebra} + \text{transitive closure}$$

Transitive closure is not as expressive as general recursion.

Fact

*The data complexity of linear stratified Datalog is lower than for Datalog (**NL** versus **P**-complete).*

Recursion that is Not Linear

Here is a program that is not linear (and you will not be able to find an equivalent linear program).

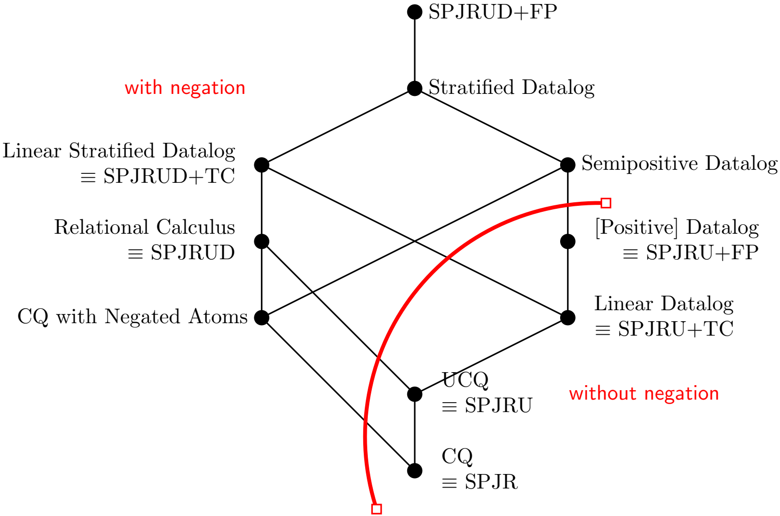
$$\begin{array}{lcl} T(x) & \leftarrow & A(x) \\ T(x) & \leftarrow & R(x,y,z), T(y), T(z) \end{array}$$

To give a meaning to this program, think of the variables as placeholders for Boolean propositions:

- $R(p,q,r)$ says that "p IF (q AND r)"
- $A(p)$ says that "p is TRUE"

So this is an interpreter for [a subset of] propositional logic.

Overview


SPJRUD+FP
with negation
Stratified Datalog
Linear Stratified Datalog
≡ SPJRUD+TC
Semipositive Datalog
Relational Calculus
≡ SPJRUD
[Positive] Datalog
≡ SPJRU+FP
CQ with Negated Atoms
Linear Datalog
≡ SPJRU+TC
UCQ
≡ SPJRU
without negation
CQ
≡ SPJR

Exercise

Let the binary relation E encode the directed edges of a graph. Express the following questions in Datalog (or explain why this is impossible).

- Is the undirected graph associated with E two-colorable?
- Is the undirected graph associated with E three-colorable?

Hint: An undirected graph is two-colorable iff it contains no undirected cycle of odd length.

Exercise

Let the binary relation E encode the directed edges of a graph, without self-loops. Let C be a binary relation such that $C(v, c)$ means that the vertex v has color c. Every vertex has exactly one color. Say that a directed path from vertex v_1 to vertex v_2 is well-colored if no three successive vertices on the path have the same color (but two successive vertices can have the same color). Express the following questions in Datalog (or explain why this is impossible); the disequality predicate $\neq$ can be used.

- Find pairs (v_1, v_2) of vertices such that there exists no well-colored directed path from v_1 to v_2.
- Find pairs (v_1, v_2) of vertices such that (i) there exists a directed path from v_1 to v_2 and (ii) all directed paths from v_1 to v_2 are well-colored.

Executing Datalog Programs in DLV

You can download dlv.exe from http://www.dlvsystem.com/

Three useful commands:

```
dlv -help
dlv input.txt
dlv -filter=Answer input.txt
```

```
%%% This is input.txt %%%
%%% The database facts %%%
C(1,blue). C(2,red). C(3,red). C(4,red).
E(1,2). E(2,3). E(3,4). E(4,1).
%%% The Datalog program %%%
V(X) :- E(X,Y).
V(Y) :- E(X,Y).
WCP(X,Y,R) :- E(X,Y), C(X,R).
WCP(X,Y,R) :- WCP(X,Z,S), E(Z,Y), C(Z,R), R != S.
WCP(X,Y,R) :- WCP(X,Z,S), E(Z,Y), C(Z,R), C(Y,T), R != T.
ExistsWCP(X,Y) :- WCP(X,Y,R).
ExistsWCP(X,X) :- V(X).
% Every vertex has a well-colored path to itself...
Answer(X,Y) :- V(X), V(Y), not ExistsWCP(X,Y).
```

Exercice

Soient *Rouge Cy* et *Bleu Cy* deux compagnies d'autobus qui utilisent des prédicats *Rouge*/2 et *Bleu*/2 pour stocker leurs connexions directes. Écrivez un programme en Datalog stratifié avec $\neq$ pour le prédicat IDB *Critique*/2 tel que *Critique*($\mathtt{v1}, \mathtt{v2}$) est vrai si les conditions suivantes sont toutes les deux satisfaites:

1. Une des deux compagnies, mais pas les deux, assure une connexion de v1 à v2. Donc, soit *Rouge*($\mathtt{v1}, \mathtt{v2}$) est vrai et *Bleu*($\mathtt{v1}, \mathtt{v2}$) est faux, soit *Rouge*($\mathtt{v1}, \mathtt{v2}$) est faux et *Bleu*($\mathtt{v1}, \mathtt{v2}$) est vrai; et
2. si la connexion de v1 à v2 est supprimée, il ne sera plus possible d'atteindre v2 à partir de v1.

Par exemple, pour la base de données ci-dessous, *Critique*($\mathtt{mons}, \mathtt{dour}$) est vrai. *Critique*($\mathtt{huy}, \mathtt{ath}$) n'est pas vrai : si la connexion *Rouge*($\mathtt{huy}, \mathtt{ath}$) est supprimée, il est encore possible d'aller de huy à ath en utilisant, par exemple, les connexions *Bleu*($\mathtt{huy}, \mathtt{dour}$) et *Rouge*($\mathtt{dour}, \mathtt{ath}$).

Rouge 1	2
ath	mons
mons	dour
dour	ath
dour	huy
huy	ath

Bleu 1	2
mons	ath
ath	mons
huy	dour
dour	huy

Connexions Critiques

```
%%%
Con(X,Y) :- Bleu(X,Y).
Con(X,Y) :- Rouge(X,Y).
%%%
% TConWO(X,Y,U,V) est vrai s'il est possible d'aller de X à Y sans
% utiliser la connexion existante de U à V.
% (TConWO = Transitive Connection Without)
%%%
TConWO(X,Y,U,V) :- Con(X,Y), Con(U,V), X!=U.
TConWO(X,Y,U,V) :- Con(X,Y), Con(U,V), Y!=V.
TConWO(X,Y,U,V) :- Con(X,Z), TConWO(Z,Y,U,V), X!=U.
TConWO(X,Y,U,V) :- Con(X,Z), TConWO(Z,Y,U,V), Z!=V.
%%%
Critique(X,Y) :- Bleu(X,Y), not Rouge(X,Y), not TConWO(X,Y,X,Y).
Critique(X,Y) :- not Bleu(X,Y), Rouge(X,Y), not TConWO(X,Y,X,Y).
```

Outline

Conjunctive Queries

Read A Primer on the Containment Problem for Conjunctive Queries.

A conjunctive query is an expression of the form

$$Answer(\vec{x}) \quad \leftarrow \quad R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$$

where every variable that occurs in $\vec{x}$ also occurs in some $\vec{x}_i$.

$Answer(\vec{x})$ is called the head, and each $R_i(\vec{x}_i)$ is called a subgoal. The set of all subgoals is called the body.

Given a database instance, the answer to this query is defined as follows:

for every valuation θ,
if the facts $R_1(\theta(\vec{x}_1))$, ..., $R_n(\theta(\vec{x}_n))$ all belong to the database, then $Answer(\theta(\vec{x}))$ belongs to the answer.

Boolean Conjunctive Query

A conjunctive query is Boolean if its head contains no variables. For example,

$$\mathit{Answer}(\mathrm{yes}) \quad \leftarrow \quad \mathit{Knows}(\mathrm{An}, y), \mathit{Owns}(y, \mathrm{iPad})$$

One can use a predicate of arity 0 instead:

$$\mathit{AnswerProposition}() \quad \leftarrow \quad \mathit{Knows}(\mathrm{An}, y), \mathit{Owns}(y, \mathrm{iPad})$$

Given a database instance I, the answer to the latter query is either $\{\mathit{AnswerProposition}()\}$ or $\{\}$, interpreted as true and false respectively.

Containment of Conjunctive queries (intuition by example)

$$q_1 : \mathit{Answer}(y) \quad \leftarrow \quad \{\mathit{Knows}(\mathrm{An}, y), \mathit{Owns}(y, \mathrm{iPad}), \mathit{Owns}(y, \mathrm{iPod})\}$$

$$\begin{aligned} q_2 : \mathit{Answer}(z) \quad \leftarrow \quad & \{\mathit{Knows}(\mathrm{An}, z), \mathit{Owns}(z, \mathrm{iPod}), \\ & \mathit{Knows}(\mathrm{An}, u), \mathit{Owns}(u, \mathrm{iPad})\ \} \end{aligned}$$

First, argue, semantically, that $q_1 \sqsubseteq q_2$.
Then, can you think of a syntactic characterization of $\sqsubseteq$?

Containment of Conjunctive queries [Ull00]

Let q_1 and q_2 be conjunctive queries. To test whether $q_1 \sqsubseteq q_2$:

1. Freeze the body of q_1 by turning each of its subgoals into facts in the database. That is, replace each variable in the body by a distinct constant, and treat the resulting subgoals as the only tuples in the database.
2. Apply q_2 to this canonical database.
3. If the frozen head of q_1 is derived by q_2, then $q_1 \sqsubseteq q_2$. Otherwise, not; in fact, the canonical database is a counterexample to the containment, since surely q_1 derives its own frozen head from this database.

Example [Ull00]

$$\begin{array}{lcl} q_1 & : & Answer(x,z) \leftarrow Knows(x,y), Knows(y,z); \\ q_2 & : & Answer(x,z) \leftarrow Knows(x,u), Knows(v,z). \end{array}$$

The canonical database I constructed from q_1 is

$$I = \{Knows(0,1), Knows(1,2)\}.$$

The frozen head is $Answer(0,2)$.

Let $\theta = \{x \mapsto 0, u \mapsto 1, v \mapsto 1, z \mapsto 2\}$. Since θ maps the body of q_2 into I, we have $Answer(0,2) \in q_2(I)$.
From this, it is correct to conclude $q_1 \sqsubseteq q_2$.

Another Example

$$q_1 \quad : \quad \mathit{Answer}(x) \leftarrow \mathit{Owns}(x, \mathrm{iPad}), \mathit{Owns}(x, \mathrm{iPod});$$
$$q_2 \quad : \quad \mathit{Answer}(y) \leftarrow \mathit{Owns}(y, \mathrm{iPad}).$$

1. Freezing q_1 gives us $I_1 := \{\mathit{Owns}(0, \mathrm{iPad}), \mathit{Owns}(0, \mathrm{iPod})\}$ and $\mathit{Answer}(0) \in q_1(I_1)$.
   Since also $\mathit{Answer}(0) \in q_2(I_1)$, it is correct to conclude $q_1 \sqsubseteq q_2$.
2. Freezing q_2 gives us $I_2 := \{\mathit{Owns}(42, \mathrm{iPad})\}$ and $\mathit{Answer}(42) \in q_2(I_2)$.
   Since $\mathit{Answer}(42) \notin q_1(I_2)$, it is correct to conclude $q_2 \not\sqsubseteq q_1$.
3. $\{y \mapsto x\}$ is a homomorphism from q_2 to q_1.
4. There is no homomorphism of q_1 to q_2.

Homomorphism Theorem

Let q_1 and q_2 be conjunctive queries.
A homomorphism from q_2 to q_1 is a substitution μ such that

- μ maps the head of q_2 to the head of q_1; and
- μ maps every subgoal of q_2 to a subgoal of q_1.

For example,

$$\begin{aligned} q_1 &: \quad \mathit{Answer}(x,z) \leftarrow \mathit{Knows}(x,y), \mathit{Knows}(y,z); \\ q_2 &: \quad \mathit{Answer}(x,z) \leftarrow \mathit{Knows}(x,u), \mathit{Knows}(v,z). \end{aligned}$$

A homomorphism μ from q_2 to q_1 is $\mu = \{x \mapsto x,\ z \mapsto z,\ u \mapsto y,\ v \mapsto y\}$.

Theorem (Homomorphism Theorem)

$q_1 \sqsubseteq q_2 \iff$ *there exists a homomorphism from q_2 to q_1*

Valuation and Substitution

- A valuation maps variables to constants.
- A substitution maps variables to variables or constants.
- A renaming is a substitution that is injective (i.e., no two distinct variables are substituted with the same variable) and maps no variable to a constant.
- It is understood that any constant is mapped to itself.

Homomorphism Theorem: Example

$$
\begin{aligned}
q_1 &: \quad \mathit{Answer}(x) \leftarrow \mathit{Knows}(x, y), \mathit{Knows}(y, x), \mathit{Knows}(y, \mathrm{Don});\\
q_2 &: \quad \mathit{Answer}(v) \leftarrow \mathit{Knows}(u, v), \mathit{Knows}(v, z).
\end{aligned}
$$

- A homomorphism μ from q_2 to q_1 is $\mu = \{v \mapsto x,\ u \mapsto y,\ z \mapsto y\}$. Hence, $q_1 \sqsubseteq q_2$.
- There exists no homomorphism from q_1 to q_2, hence $q_2 \not\sqsubseteq q_1$.

Homomorphism Theorem: Sketch of Proof

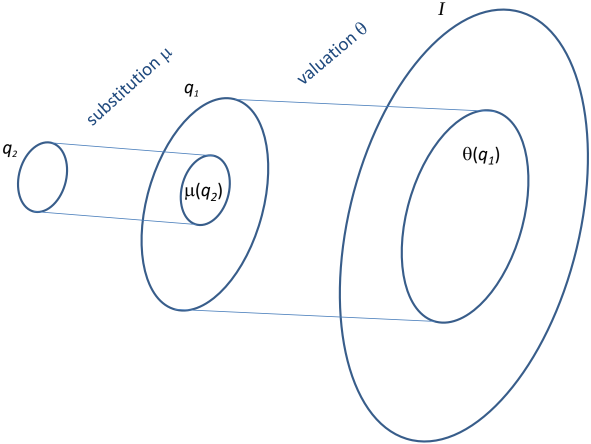
Theorem (Homomorphism Theorem)

$q_1 \sqsubseteq q_2 \iff$ *there exists a homomorphism from q_2 to q_1*

$\boxed{\Longrightarrow}$ Take the canonical database for q_1. Since the frozen head of q_1 is in the answer to q_1, it must be in the answer to q_2. This implies a homomorphism from q_2 to q_1 (because the constants in the frozen database map one-to-one to the variables in q_1).

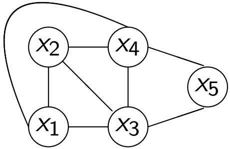
$\boxed{\Longleftarrow}$ Let μ be the homomorphism from q_2 to q_1. Assume that the fact h belongs to the answer to $q_1 : H \leftarrow B$ on some database I. Then, there exists a valuation θ such that $\theta(H) = h$ and $\theta(B) \subseteq I$. The composition $\theta \circ \mu$ shows that $h \in q_2(I)$.

Rough Visualization of the $\boxed{\Longleftarrow}$ Proof



Side Remark: 3-Colorability

Containment of conjunctive queries is fundamental in computer science, beyond database courses.



b, g, r are three distinct constants, representing three colors.

$$q_1 \quad : \quad Answer() \leftarrow R(b, g), R(g, b), R(b, r), R(r, b), R(g, r), R(r, g)$$
$$q_2 \quad : \quad Answer() \leftarrow R(x_1, x_2), R(x_2, x_1), \ldots, R(x_4, x_5), R(x_5, x_4)$$

Whenever there is an edge between x_i and x_j in the graph, the body of q_2 contains $R(x_i, x_j)$ and $R(x_j, x_i)$.

Check: $q_1 \sqsubseteq q_2 \iff$ the graph encoded by q_2 is 3-colorable

Data Complexity and Query Complexity

For a database I and a query q, what is the time complexity of computing $q(I)$?

One can distinguish between three complexities:

Data complexity
: Time complexity in terms of the size of the database, for a *fixed* query. This is the complexity that matters in most practical applications.

Query complexity
: Time complexity in terms of the size of the query, for a *fixed* database. E.g., one could fix a canonical database $\{R(b, g), R(g, b), R(b, r), R(r, b), R(g, r), R(r, g)\}$.

Combined complexity
: Time complexity in terms of both the size of the database and the size of the query.

The data complexity of datalog is polynomial-time, but the query complexity is already exponential-time for conjunctive queries (unless $\mathbf{P} = \mathbf{NP}$).

Side Remark: Satisfiability

$$\varphi = (p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p) \wedge (\neg q \vee \neg r)$$

$$\begin{array}{lcll} q_1 & : & \mathit{Answer}() & \leftarrow & PP(0,1), PP(1,0), PP(1,1), \\ & & & & NP(0,0), NP(1,1), NP(0,1), \\ & & & & NN(0,1), NN(1,0), NN(0,0) \\ q_2 & : & \mathit{Answer}() & \leftarrow & PP(p,q), NP(q,r), NP(r,p), NN(q,r) \end{array}$$

A homomorphism from q_2 to q_1 is $\mu = \{p \mapsto 1, q \mapsto 0, r \mapsto 0\}$.
μ is also a satisfying truth assignment for φ.

Check: $q_1 \sqsubseteq q_2 \iff$ the 2-CNF formula encoded by q_2 is satisfiable

Query Optimization for Conjunctive Queries

A conjunctive query is minimal if it is not equivalent to any conjunctive query with a strictly smaller number of subgoals.

Theorem

For every conjunctive query $q_1 : H \leftarrow B_1$, there exists a subset $B_2 \subseteq B_1$ such that $q_2 : H \leftarrow B_2$ is minimal and equivalent to q_1.

Theorem

If two minimal conjunctive queries are equivalent, then they are identical up to a renaming of variables.

For the proofs, see A Primer on the Containment Problem for Conjunctive Queries.

Outline

Unions of Conjunctive Queries

A union of conjunctive queries is a finite set $Q = \{q_1, \ldots, q_\ell\}$ of conjunctive queries, all with the same head predicate.
The semantics is natural: $Q(I) = \bigcup_{i=1}^{\ell} q_i(I)$.

Theorem

For Q_1 and Q_2 unions of conjunctive queries,

$$Q_1 \sqsubseteq Q_2 \iff \forall q \in Q_1 \exists p \in Q_2 : q \sqsubseteq p$$

The proof of $\Longleftarrow$ is straightforward. For the $\Longrightarrow$ direction, see what happens if we take the canonical database for any $q \in Q_1$.

Query Optimization for Unions of Conjunctive Queries

Check:

$$\begin{array}{rcl} \mathit{Answer}(y) & \leftarrow & \mathit{Knows}(y,x), \mathit{Knows}(x,y), \mathit{Knows}(y, \mathrm{Don}) \\ \mathit{Answer}(y) & \leftarrow & \mathit{Knows}(x,y), \mathit{Knows}(y,x), \mathit{Knows}(y,z) \end{array}$$

is equivalent to

$$\mathit{Answer}(y) \quad \leftarrow \quad \mathit{Knows}(x,y), \mathit{Knows}(y,x)$$

UCQ $\equiv$ SPJRU

$$
\begin{aligned}
\sigma_{A=c}(E \cup F) &\equiv \sigma_{A=c}(E) \cup \sigma_{A=c}(F) \\
\sigma_{A=B}(E \cup F) &\equiv \sigma_{A=B}(E) \cup \sigma_{A=B}(F) \\
\pi_X(E \cup F) &\equiv \pi_X(E) \cup \pi_X(F) \\
\rho_{A\mapsto B}(E \cup F) &\equiv \rho_{A\mapsto B}(E) \cup \rho_{A\mapsto B}(F) \\
E \bowtie (F \cup G) &\equiv (E \bowtie F) \cup (E \bowtie G) \\
(E \cup F) \bowtie G &\equiv (E \bowtie G) \cup (F \bowtie G)
\end{aligned}
$$

$\Longrightarrow$ every expression E in SPJRU can be equivalently rewritten in the form $E_1 \cup E_2 \cup \cdots \cup E_\ell$ where each E_i is union-free (i.e., each E_i is a conjunctive query).

Note: the last two rules result in an exponential blowup in the size of the query (but that does not matter if we are only concerned about data complexity).

Containment of Conjunctive queries in Datalog Queries

`[slide added for completeness]`

Let q_1 be a conjunctive query, and q_2 a datalog query. To test whether $q_1 \sqsubseteq q_2$:

1. Freeze the body of q_1 by turning each of its subgoals into facts in the database.
2. Apply q_2 to the canonical database.
3. If the frozen head of q_1 is derived by q_2, then $q_1 \sqsubseteq q_2$. Otherwise, not.

Outline

Containment of Conjunctive Queries with Safe Atomic Negation

Failure of the "canonical database" approach.

$$\begin{array}{lcl} q_1 & : & Answer() \leftarrow R(x,y,z) \\ q_2 & : & Answer() \leftarrow R(x,y,z), \neg R(z,x,y) \end{array}$$

- Clearly, $q_1 \not\equiv q_2$, but
  - q_1 and q_2 agree on $\{R(a,b,c)\}$; and
  - q_1 and q_2 even agree on $I = \{R(a,b,c), R(c,a,b)\}$ (because $R(c,a,b) \in I$ and $R(b,c,a) \notin I$).
- q_1 and q_2 disagree on $I = \{R(a,b,c), R(c,a,b), R(b,c,a)\}$.

Containment of Conjunctive Queries with Safe Atomic Negation

The Levy-Sagiv test [LS93] for testing $q_1 \sqsubseteq q_2$, where q_1, q_2 are queries without constants.

- We use an alphabet A of k constants, where k is the number of variables in q_1.
- We consider all databases I whose active domain is contained in A. If $q_1(I) \subseteq q_2(I)$ for each of these canonical databases, then $q_1 \sqsubseteq q_2$, and if not, then not.

That is, if $q_1 \not\sqsubseteq q_2$, then there exists a database I whose active domain contains no more than k constants such that $q_1(I) \not\subseteq q_2(I)$.

Choice of constants

The choice of constants in A is not important, because database queries q are generic:

for each permutation σ of constants, $q(\sigma(I)) = \sigma(q(I))$.

Correctness Proof (Sketch)

1. Assume that for every database I in the Levy-Sagiv test, $q_1(I) \subseteq q_2(I)$.
2. Let E be an arbitrary database, and let t be a fact such that $t \in q_1(E)$. It suffices to show $t \in q_2(E)$.
3. Let $\{c_1, \ldots, c_n\}$ be the (necessarily finite) set of constants that variables of q_1 are mapped to when showing $t \in q_1(E)$.
4. Let D be the database containing all (and only) the facts of E all of whose components are in $\{c_1, \ldots, c_n\}$. From 1 and genericity, it follows $q_1(D) \subseteq q_2(D)$. From $t \in q_1(D)$ (because $t \in q_1(E)$), it follows $t \in q_2(D)$.
5. The valuation that shows $t \in q_2(D)$ maps positive subgoals of q_2 to facts in E, and maps negative subgoals of q_2 to facts not in E (Why?). Hence, $t \in q_2(E)$. Recall that every variable that occurs in q_2, occurs in a nonnegated subgoal of q_2 (safety).

Correctness Proof (in More Detail)

1. Assume $q_1(I) \subseteq q_2(I)$ for every database I in the Levy-Sagiv test.
2. Let E be an arbitrary database, and let t be a fact such that $t \in q_1(E)$. It suffices to show $t \in q_2(E)$.
3. Let B_1^+ and B_2^+ be the sets of positive subgoals in, respectively, q_1 and q_2. Let H_1 and H_2 be the heads of, respectively, q_1 and q_2.
4. Since $t \in q_1(E)$, there is a valuation θ such that $\theta(H_1) = t$, $\theta(B_1^+) \subseteq E$, and θ maps negative subgoals of q_1 to facts not in E.
5. Let D be the database that contains all (and only) facts of E that use only constants occurring in $\theta(B_1^+)$. Clearly, $\theta(B_1^+) \subseteq D \subseteq E$. From item 1 and genericity, it follows $q_1(D) \subseteq q_2(D)$. From $t \in q_1(D)$ (because $t \in q_1(E)$), it follows $t \in q_2(D)$. Hence, there is a valuation μ such that $\mu(H_2) = t$, $\mu(B_2^+) \subseteq D$, and μ maps negative subgoals of q_2 to facts not in D.
6. Let $\neg R(\vec{x})$ be a subgoal of q_2, and let $\vec{b} = \mu(\vec{x})$. Since every variable of $\vec{x}$ occurs in B_2^+ (safety) and since $\mu(B_2^+) \subseteq D$, it follows that $R(\vec{b})$ uses only constants occurring in D. Since $R(\vec{b}) \notin D$ (by item 5), we have $R(\vec{b}) \notin E$ (by our construction of D). Thus, μ maps negative subgoals of q_2 to facts not in E. Hence, $t \in q_2(E)$.

Example

$$q_1 \quad : \quad Ans(x,z) \leftarrow Knows(x,y), Knows(y,z), \neg Knows(x,z)$$
$$q_2 \quad : \quad Ans(x,z) \leftarrow Knows(x,y), Knows(y,z), Knows(y,u), \neg Knows(x,u)$$

Is $q_1 \sqsubseteq q_2$?

The query q_1 contains three variables. Let $A = \{0, 1, 2\}$.

- Let $I = \emptyset$. Since $q_1(I) \subseteq q_2(I) = \emptyset$, this is not a counterexample for $q_1 \sqsubseteq q_2$.
- ...
- Let $I = \{Knows(0,1), Knows(1,0)\}$, a database whose active domain is contained in A. We have $q_1(I) = \{Ans(0,0), Ans(1,1)\}$. Since $q_1(I) \subseteq q_2(I)$, this is not a counterexample for $q_1 \sqsubseteq q_2$.
- ...

After a lot (but finite amount) of work, we will have found no counterexample for $q_1 \sqsubseteq q_2$. It is correct to conclude $q_1 \sqsubseteq q_2$.

Example

$$
\begin{aligned}
q_1 &: \quad Ans(x,z) \leftarrow Knows(x,y), Knows(y,z), \neg Knows(x,z) \\
q_2 &: \quad Ans(x,z) \leftarrow Knows(x,y), Knows(y,z), Knows(y,u), \neg Knows(x,u)
\end{aligned}
$$

Is $q_2 \sqsubseteq q_1$?

Let $I = \{Knows(0,1), Knows(1,2), Knows(0,2), Knows(1,3)\}$.
We have $Ans(0,2) \in q_2(I)$ and $Ans(0,2) \notin q_1(I)$, hence $q_2 \not\sqsubseteq q_1$.

References

Patrick Blackburn, Johan Bos, and Kristina Striegnitz.
Learn Prolog Now!, volume 7 of *Texts in Computing*.
College Publications, 2006.

Alon Y. Levy and Yehoshua Sagiv.
Queries independent of updates.
In *VLDB*, pages 171–181, 1993.

Christos H. Papadimitriou.
Computational complexity.
Addison-Wesley, 1994.

Alan M. Turing.
On computable numbers, with an application to the Entscheidungsproblem.
Proceedings of the London Mathematical Society, 2(42):230–265, 1936.

Jeffrey D. Ullman.
Information integration using logical views.
Theor. Comput. Sci., 239(2):189–210, 2000.