Deductive databases

Datalog

Guillaume Raschia — Université de Nantes

Last update: October 15, 2012

[Source : S. Abiteboul, INRIA]

[Source : E. Zimányi, Univ. libre de Bruxelles]

[Source : J. D. Ullman, Stanford]

[Source : W. Nutt, Free University of Bozen–Bolzano]

Introduction

Querying relational database

- Relational Calculus (RC) over FO formulas: highly expressive query language
- RC cannot express many important queries, e.g.,
  - transitive closures and
  - generalized aggregates
- Need of more powerful logic-based languages that subsume RC

Contents

Datalog

Two alternative viewpoints

database vs. logic programming

	R	
	A	B
	0	1
	1	2

	S	
	A	C
	1	1
	1	2

Columns are **named**

$$\mathtt{I} = \{R(0,1), R(1,2), S(1,1), S(1,2)\}$$

Set of facts

Columns are **numbered**

Another example

Student

Name	Major	Year
Alice	cs	senior
Bob	cs	junior
Carol	ee	junior

Enroll

Name	Course	Grade
Alice	cs123	2.7
Bob	cs101	3.0
Bob	cs143	3.3
Carol	cs143	3.3
Carol	cs101	2.7

Student(Alice, cs, senior)
Student(Bob, cs, junior)
Student(Carol, ee, junior)

Enroll(Alice, cs123, 2.7)
Enroll(Bob, cs101, 3.0)
Enroll(Bob, cs143, 3.3)
Enroll(Carol, cs143, 3.3)
Enroll(Carol, cs101, 2.7)

In the following: Student is "St" and Enroll is "En"

Datalog rules

"*A tool for deducing new facts*"

Example

$$\underbrace{Q_1(x)}_{\text{head}} \leftarrow \underbrace{\mathrm{St}(x, \_, \ \text{junior}), \overbrace{\mathrm{En}(x, \mathrm{cs101}, \_)}^{\text{goal}}, \mathrm{En}(x, \mathrm{cs143}, \_)}_{\text{body}}$$

- Commas are logical conjuncts, order of goals is immaterial
- Goals are **stored relations**, head is not
- $\mathbf{var}(r) = \{x, \_1, \_2, \_3\}$
- $\mathrm{adom}(r) = \{\mathrm{junior}, \mathrm{cs101}, \mathrm{cs143}\}$

Safe rule

$$R(\vec{u}) \leftarrow R_1(\vec{u_1}), \ldots, R_n(\vec{u_n}) \qquad \textit{for } n \geq 1$$

- Variables in **var**(r) may be:
  - **Distinguished**: variables that occur in the head $\vec{u}$ of r
  - **Anonymous**: unnamed existentially bound variables, all denoted '_' in rules and '$\__i$' in **var**(r)

Safety

Each **distinguished variable** must also occur in the body $\vec{u_1}, \ldots, \vec{u_n}$.

- Safe rules are then **range-restricted** queries

Rule evaluation

Let r be a rule; $\mathtt{I}$ a database instance over $\mathcal{R}$.

- We denote $\text{adom}(r, \mathtt{I}) = \text{adom}(r) \cup \text{adom}(\mathtt{I})$
- **Valuation** ν : mapping from **var**(r) to D
- **Instantiation**: $R(\nu(\vec{u})) \leftarrow R_1(\nu(\vec{u_1})), \ldots, R_n(\nu(\vec{u_n}))$
- $r(\mathtt{I})$, the **image** of $\mathtt{I}$ under r, is the set of all possible instantiations
- $\text{adom}(r(\mathtt{I})) \subseteq \text{adom}(r, \mathtt{I})$: finiteness of the answer

Straightforward algorithm

Systematic valuation of the set of variables in r within $\text{adom}(r, \mathtt{I})$

Rules as queries

- **Boolean** or closed queries:

$$Q_2(\mathrm{Carol}) \leftarrow . \quad \text{or} \quad Q2(\mathrm{Carol}).$$

$$Q_3() \leftarrow \cdots \quad \text{where } Q3 \in \left\{\{\} = \text{false}, \{\langle\rangle\} = \text{true}\right\}$$

- **Open** queries with free tuples (constants and variables):

$$Q_4(x, \mathrm{cs101}) \leftarrow \cdots$$

- Their answer is a (possibly empty) set of facts satisfying the query

$$\{Q_4(\mathrm{Bob}, \mathrm{cs101}), Q_4(\mathrm{Carol}, \mathrm{cs101})\}$$

Incorporating equality

Rule-based queries with equality predicate

- Mainly the same: $R(3.0)$ and $R(x), x = 3.0$

Problem #1

$$S(x, y) \leftarrow R(x), y = z$$

Safe query revisited

Same as before + each variable in the body is equal to some constant or some variable in an atom $R_i(\vec{u_i})$

Incorporating equality (cont'd)

Problem #2

$$S(x) \leftarrow R(x), x = 3.0, x = 6.0$$

- Unsatisafiable queries $Q^{\emptyset}$
- Satisfiable queries have the same expressive power than regular rule-based queries
- Extension to any built-in predicate ($<$ $>$ etc.) is obvious

Conjunctive queries

Theorem

Rule-based queries are equivalent to SPC

- Selection:

$$\begin{aligned}\sigma_{v=w}(R) \quad &= \quad Q(\vec{x}) \leftarrow R(\vec{x}), \vec{x}_{[v]} = \vec{x}_{[w]}\\ \text{where } \vec{x}_{[i]} = x_i, \vec{x}_{[a]} = a\end{aligned}$$

- Projection:

$$\pi_{\vec{\ell}}(R) \quad = \quad Q(x_{\ell_1}, \ldots, x_{\ell_k}) \leftarrow R(\vec{x})$$

- Cross product:

$$R \times S \quad = \quad Q(\vec{x}, \vec{y}) \leftarrow R(\vec{x}), S(\vec{y})$$

Datalog programs

- Example:

$$Q_5(x) \leftarrow \mathrm{St}(x, \_, \mathrm{junior}), \mathrm{En}(x, \mathrm{cs143}, y), y > 3.0$$
$$Q_6(x, y) \leftarrow \mathrm{St}(x, \_, \mathrm{senior}), \mathrm{En}(x, \mathrm{cs143}, y)$$

- Datalog program P: **finite set of datalog rules**
- Order of rules doesn't matter
- Scope of variables is local to rules

Datalog vs. relational model

Datalog	Relational model
Base predicate	Table or relation
Derived predicate	View or query
Fact	Row or tuple
Argument	Column or Attribute

- **Extensional database**: base predicate
- **Intensional database**: derived predicate (defined by rules)
- Assumptions:
  - edb occurs only in the body of the rules
  - idb occurs at least in one head of some rule

More about programs

Let P a Datalog program;

- **Schema** $\mathcal{P} = \text{edb}(P) \cup \text{idb}(P)$
- **Semantics** of P:
  maps instances over $\text{edb}(P)$ to instances over $\text{idb}(P)$
- The 3 ways:
  1. Multiple outputs within the same evaluation
  2. Composition and views for CQ
  3. **Union** operator: multiple rules with same head

CQ programs

Closure under composition

A CQ program P is equivalent to a rule-based CQ

- Example:

$$S_1(x) \leftarrow Q(x,y), R(y)$$
$$S(x,z) \leftarrow S_1(x), T(x,y,z)$$

  is equivalent to CQ: $S(x,z) \leftarrow Q(x,y_1), R(y_1), T(x,y_2,z)$
- Problem arises with hidden unsatisfiable equalities

$$T(a,x) \leftarrow R(x)$$
$$S(x) \leftarrow T(b,x)$$

Expressiveness

- A program for union:

$$Q_7(x) \leftarrow \mathrm{St}(x, \_, \mathrm{junior}), \mathrm{En}(x, \mathrm{cs143}, y), y > 3.0$$
$$Q_7(x) \leftarrow \mathrm{St}(x, \_, \mathrm{junior}), \mathrm{En}(x, \mathrm{cs101}, y), y > 3.0$$

- Keep in mind that the semantics is set-theoretic

Theorem

Single output nr-Datalog programs are equivalent to SPCU

Negation

Syntax

- Negation can only be applied to goals of rules

$$\text{Registered}(x, y) \leftarrow \text{En}(x, y, z)$$
$$\text{UnregCs143}(x) \leftarrow \text{St}(x, \_, \text{junior}), \neg\text{Registered}(x, \text{cs143})$$

Unsafe rules

$$S(x, y) \leftarrow R(x), \neg T(y)$$

Unsafe rules with negation

- To sum up:

$$S(x) \leftarrow R(y) \quad (1)$$

$$S(x) \leftarrow R(y), x < y \quad (2)$$

$$S(x, y) \leftarrow R(x), \neg T(y) \quad (3)$$

- Two problems arise:
  - Infinite output from finite input
  - Domain-dependent query
- Domain-independence and finiteness of answer are **undecidable**
- Sufficient conditions are required

Safety—ultimately—revisited

A rule is **safe** iff:

1. Each distinguished variable,
2. Each named variable in a built-in predicate, and
3. Each named variable in a negated goal,

also occurs in a **positive relational goal**

- Safe rules prevent from infinite and domain-dependent results
- Variables bound to constants ($x = 2$) are obviously safe-range

Safety relaxation

- Safety: sufficient but not necessary condition
- Example with existential variables in negated goals

$$R(x, y) \leftarrow \mathrm{St}(x, \mathrm{cs}, y), \neg\mathrm{En}(x, \mathrm{cs143}, z)$$

- The above rule can be viewed as a shorthand for

$$\mathrm{RegCs143}(x, \mathrm{cs143}) \leftarrow \mathrm{En}(x, \mathrm{cs143}, z)$$
$$R(x, y) \leftarrow \mathrm{St}(x, \mathrm{cs}, y), \neg\mathrm{RegCs143}(x, \mathrm{cs143})$$

- Anonymous variable to the rescue: $\ldots \neg\mathrm{En}(x, \mathrm{cs143}, \_)$

Universal quantification

A common use of negation

Example

Find senior students who completed all requirements for a cs major

- Equivalence: $\forall x.P(x) \Leftrightarrow \neg\exists x.\neg P(x)$
- Find senior students who are not missing any requirement for a cs major

Universal quantification (cont'd)

Howto build Datalog query

- It requires two steps
  1. Formulate complementary query: find students who did not register to some of the courses required for a cs major

$$\text{MissingReq}(x) \leftarrow \text{St}(x, \_, \text{senior}), \text{Req}(\text{cs}, y), \neg\text{Registered}(x, y)$$

  2. Original query reformulated as : find senior students who are not missing any requirement for a cs major

$$Q_8(x) \leftarrow \text{St}(x, \_, \text{senior}), \neg\text{MissingReq}(x)$$

Expressiveness

Theorem

Single output nr-Datalog$^\neg$ programs are equivalent to RA

- Generic translation of **monotonic** rule r:

$$Q(r) = \pi_{\text{head}}\Big(\sigma_{\text{body constraints}}(\times \text{body predicates})\Big)$$

- Mapping rules **with negated goals** into RA

$$r : \cdots \leftarrow R(x), \neg S(y)$$

- Define $r^+ : \cdots \leftarrow R(x)$ and $r^- : \cdots \leftarrow R(x), S(y)$
- Template of algebraic expression: $Q(r) = Q(r^+) - Q(r^-)$

Evaluation of *nr*-Datalog$^\neg$ programs

For any *nr*-Datalog$^\neg$ program $P = \{r_1, \ldots, r_m\}$, there exists an **ordering** on rules such that the relation name in the head of rule r_i does not occur in the body of any rule r_j if $r_j \leq r_i$

- Rules evaluation follows from $\leq$
- Since there is no recursion, there always exists a first rule within edb's only in the body
- Distinct rules with same head have same position in the ordering

More about expressiveness

- *nr*-Datalog$^\neg$ can even express min and max!
- Counting elements in a set modulo an integer requires recursion
- Recursion addresses **connectivity** and **transitive closure** issues
  - Corporate hierarchy
  - File system
  - Transport network
  - XML document
  - ...
- Other aggregates (count, sum, avg, etc.) require recursion and **arithmetic**

Exercises 1/2

1. Definitions

Rule, Head, Distinguished variable, Goal, Datalog program, *nr*-Datalog$^{\neg}$, Safety, Edb, Idb

2. True or False?

i) Union op. is mimicked with multiple rules having the same head.
ii) Every *nr*-Datalog$^{\neg}$ program is equivalent to RA.
iii) Safety condition is necessary.
iv) Negation can occur in the head.
v) $\text{adom}(r(\mathtt{I})) \subseteq \text{adom}(r) \cup \text{adom}(\mathtt{I})$.

Exercises 2/2

3. *nr*-Datalog$^\neg$ queries ✎

1. Find courses where none of the students in cs major enrolled.
2. Find courses where all of the students in cs major enrolled.
3. Provide *nr*-Datalog$^\neg$ program to compute the maximum value from $R:1$

4. Problem ✎

An *inequality atom* is an expression of the form $x \neq y$ or $x \neq a$ where x, y are variables and a is a constant. Show that the family of rule-based conjunctive queries with equality and inequality strictly dominates the family of rule-based conjunctive queries with equality.

Outline

Excerpt of Tramway database

Links	Line	Station	Next-Station
	1	Chantiers Navals	Médiathèque
	1	Médiathèque	Commerce
	1	Commerce	Duchesse Anne Château
	1	Duchesse Anne Château	Gare SNCF
	2	Hôtel Dieu	Commerce
	2	Commerce	Place du Cirque
	2	Place du Cirque	50 Otages
	3	Commerce	Bretagne
	3	Bretagne	Jean Jaurès
	⋮	⋮	⋮

Example: (recursive) program P_{tram}

$$
\begin{array}{lll}
r_1: & \text{St\_Reach}(x, x) & \leftarrow \text{Links}(\_, x, \_) \\
r_2: & \text{St\_Reach}(x, y) & \leftarrow \text{St\_Reach}(x, z), \text{Links}(\_, z, y) \\
r_3: & \text{Li\_Reach}(x, u) & \leftarrow \text{St\_Reach}(x, z), \text{Links}(u, z, \_) \\
r_4: & \text{Ans}_1(y) & \leftarrow \text{St\_Reach}(\text{Commerce}, y) \\
r_5: & \text{Ans}_2(u) & \leftarrow \text{Li\_Reach}(\text{Commerce}, u) \\
r_6: & \text{Ans}_3() & \leftarrow \text{St\_Reach}(\text{Commerce}, \text{Gare SNCF})
\end{array}
$$

Observe that St_Reach is defined using recursion

Example: program P_{tram} (con't)

- Schema:
  - $\mathcal{P}_{\text{tram}} = \{\text{Links}, \text{St\_Reach}, \text{Li\_Reach}, \text{Ans}_1, \text{Ans}_2, \text{Ans}_3\}$
  - $\text{edb}(P_{\text{tram}}) = \{\text{Links}\}$
  - $\text{idb}(P_{\text{tram}}) = \{\text{St\_Reach}, \text{Li\_Reach}, \text{Ans}_1, \text{Ans}_2, \text{Ans}_3\}$
- Valuation ν of $\textbf{var}(r_2)$:

$$\nu(x) = \text{“Chantiers Navals”}, \nu(y) = \text{“Place du Cirque”},$$
$$\nu(z) = \text{“Commerce”}, \nu(\_) = 2$$

- Instantiation of r_2:

$$\begin{aligned}&\text{St\_Reach}(\text{Chantiers Navals}, \text{Place du Cirque}) \leftarrow\\ &\qquad \text{St\_Reach}(\text{Chantiers Navals}, \text{Commerce}),\\ &\qquad\qquad \text{Links}(2, \text{Commerce}, \text{Place du Cirque})\end{aligned}$$

Deciding recursion

Dependency graph $\mathcal{G}(P)$

- Vertices are $\mathrm{idb}(P)$
- Edge $X \rightarrow Y$ iff there is a rule with X in the head and Y in the body

Cycle means **recursion**; no cycle means no recursion

Example

$$\mathcal{G}(P_{\text{tram}}) = \Big(\{\text{St\_Reach}, \text{Li\_Reach}, \text{Ans}_1, \text{Ans}_2, \text{Ans}_3\},$$

$$\Big\{\text{St\_Reach} \rightarrow \text{St\_Reach}, \text{Li\_Reach} \rightarrow \text{St\_Reach}, \text{Ans}_1 \rightarrow \text{St\_Reach},$$

$$\text{Ans}_2 \rightarrow \text{Li\_Reach}, \text{Ans}_3 \rightarrow \text{St\_Reach}\Big\}\Big)$$

Semantics of a Datalog program

What is the database instance over $\mathcal{P}$ that is the answer of a given Datalog program P?

Alternative semantics for positive logic programs are **equivalent**

- The three semantics:
  - **Model-theoretic**—declarative meaning of a program
  - **Fixpoint**—bottom-up implementation of deductive DBs
  - **Proof-theoretic**—SLD resolution and top-down execution
- Semantics for more general programs (e.g. with negation) is more complex

Model-theoretic semantics

Main idea

View P as a **first-order sentence** Σ_P that describes the answer

- Associate a formula to a rule $r = R(\vec{u}) \leftarrow R_1(\vec{u_1}), \ldots, R_n(\vec{u_n})$:

$$\forall x_1, \ldots, x_m \Big(R_1(\vec{u_1}) \wedge \ldots \wedge R_n(\vec{u_n}) \longrightarrow R(\vec{u}) \Big)$$

  where $x_1, \ldots, x_m$ are the variables occurring in the rule

- $P = \{r_1, \ldots, r_q\}$:

$$\Sigma_P = r_1 \wedge \ldots \wedge r_q$$

Model-theoretic semantics (cont'd)

Model $\mathcal{M}$ of Σ_P

$\mathcal{M}$ satisfies Σ_P: $\mathcal{M} \models r_1 \wedge \ldots \wedge r_q$

- $\mathcal{M} \models r$: for each ν such that $R_1(\nu(\vec{u_1})), \ldots, R_n(\nu(\vec{u_n}))$ belong to $\mathcal{M}$, then $R(\nu(\vec{u}))$ also belongs to $\mathcal{M}$

Actually, the answer of P is a singular model of Σ_P

Example - Transitive closure

$$T(x,y) \leftarrow G(x,y)$$
$$T(x,y) \leftarrow G(x,z), T(z,y)$$

$$\mathtt{I} = \{G(0,1), G(1,2), G(2,3)\}$$

$\mathcal{M}$

G		T	
0	1	0	1
1	2	1	2
2	3	2	3
		0	2
		1	3
		0	3
		6	6

$\mathcal{M}'$

G		T	
0	1	0	1
1	2	1	2
2	3	2	3
		0	2
		1	3

$P(\mathtt{I})$

G		T	
0	1	0	1
1	2	1	2
2	3	2	3
		0	2
		1	3
		0	3

$\mathcal{M}'$ is not a model of Σ_P since $P(0,3)$ does not belong to $\mathcal{M}'$

Minimal model of P containing $\mathtt{I}$

Closed World Assumption (CWA)

- Database is assumed to be **complete**
- Known facts are sure (true); other facts are not (false)

Semantics $P(\mathtt{I})$ of program P

- In: a Datalog program P, an instance $\mathtt{I}$ over edb(P)
- Out: a **model** of P, that is, an instance over $\mathcal{P}$ satisfying Σ_P
- $P(\mathtt{I})$, is the minimal model of P containing $\mathtt{I}$
- Problems :
  1. Is this definition **correct**? (existence and uniqueness)
  2. How do we compute it **efficiently**?

An upper bound

- $P(\mathtt{I})$ is an instance over $\mathcal{P}$
- Only a finite set of instances in $\text{adom}(P, \mathtt{I})$
- Consider one of them, $\mathcal{B}(P, \mathtt{I})$ built as follows:
  - For each $R \in \text{edb}(P)$, a fact $R(\vec{u})$ is in $\mathcal{B}(P, \mathtt{I})$ iff it is in $\mathtt{I}$
  - For each $R \in \text{idb}(P)$, each fact $R(\vec{u})$ with constants in $\text{adom}(P, \mathtt{I})$ is in $\mathcal{B}(P, \mathtt{I})$
- $\mathcal{B}(P, \mathtt{I})$ is a model of P containing $\mathtt{I}$
- $P(\mathtt{I}) \subseteq \mathcal{B}(P, \mathtt{I})$

Main result

Theorem

Let P be a Datalog program, $\mathtt{I}$ an instance over $\text{edb}(P)$, and $\mathfrak{M}$ the set of models of P containing $\mathtt{I}$. Then

1. $\bigcap \mathfrak{M}$ *is the minimal model of P containing $\mathtt{I}$, so $P(\mathtt{I})$ is defined*
2. $\text{adom}(P(\mathtt{I})) \subseteq \text{adom}(P, \mathtt{I})$
3. *For each R in $\text{edb}(P)$, $P(\mathtt{I})(R) = \mathtt{I}(R)$*

- Very inefficient algorithm:
  1. Find the set $\mathfrak{M}$ of instances over $\mathcal{P}$ that are subsets of $\mathcal{B}(P, \mathtt{I})$, satisfying Σ_P and containing $\mathtt{I}$
  2. The answer is $\bigcap \mathfrak{M}$

Exercises 1/2

1. Definitions

Datalog, Dependency graph, Model, Model-theoretic semantics, CWA

2. True or False?

i) Answer of a Datalog program is the so-called minimal model.
ii) Datalog dominates RA.
iii) Each goal of a rule is a node in the dependency graph.
iv) CWA states that the database is assumed to be complete.
v) Cycle detection in a graph is undecidable.

Exercises 2/2

3. Misc.

1. Give a Datalog program that yields, for each pairs of stations (a, b), the station c such that c is reachable (1) from both a and b, and (2) from a or b.
2. Prove that Datalog queries are *monotonic*.

4. Problem

Give a proof of each of the three results from the "minimal model" theorem. Hint: First, build upon an intermediate result that states for every valuation of goals of a rule in $\bigcap \mathfrak{M}$, valuation of the head is also in $\bigcap \mathfrak{M}$. And second, Use $\mathcal{B}(P, \mathtt{I}) \in \mathfrak{M}$.

Outline

Fixpoint semantics

A **bottom-up** semantics for Datalog

- Let P a datalog program, $\mathtt{K}$ an instance over $\mathcal{P}$
- A fact F is an **immediate consequence** for $\mathtt{K}$ and P if:
    1. $F \in \mathtt{K}(R)$ for some edb R, or
    2. $F \leftarrow F_1, \ldots, F_n$ is an instantiation of a rule in P and each $F_i \in \mathtt{K}$

Immediate consequence operator

$$\begin{aligned} T_P : \quad \mathrm{inst}(\mathcal{P}) & \rightarrow \mathrm{inst}(\mathcal{P}) \\ \mathtt{K} & \mapsto T_P(\mathtt{K}) = \{\text{immediate consequences for } \mathtt{K} \text{ and } P\} \end{aligned}$$

Some properties

Property 1

T_P is **monotonic**

$$\forall \mathtt{I}, \mathtt{J},\ \mathtt{I} \subseteq \mathtt{J} \ \Rightarrow\ T_P(\mathtt{I}) \subseteq T_P(\mathtt{J})$$

Property 2

$\mathtt{K}$ over $\mathcal{P}$ is a model of Σ_P iff $T_P(\mathtt{K}) \subseteq \mathtt{K}$

Some properties (cont'd)

Definition (Fixpoint)

$\mathtt{K}$ is a **fixpoint** of T_P iff $T_P(\mathtt{K}) = \mathtt{K}$

Property 3

Each fixpoint of T_P ($T_P(\mathtt{K}) = \mathtt{K}$) is a model of Σ_P ($T_P(\mathtt{K}) \subseteq \mathtt{K}$)

The converse does not necessarily hold

Property 4

For each P and $\mathtt{I}$ over $\mathrm{edb}(P)$, T_P has a least fixpoint containing $\mathtt{I}$

Construction

- Compute $T_P(\mathtt{I})$, $T_P(T_P(\mathtt{I}))$, $T_P^3(\mathtt{I})$, etc.
- $\mathtt{I} \subseteq T_P(\mathtt{I}) \subseteq T_P^2(\mathtt{I}) \subseteq T_P^3(\mathtt{I}) \subseteq \cdots \subseteq \mathcal{B}(P, \mathtt{I})$
- $\{T_P^i(\mathtt{I})\}_i$ reaches a fixpoint after at most $N = |\mathcal{B}(P, \mathtt{I})|$ steps:

$$T_P(T_P^N(\mathtt{I})) = T_P^N(\mathtt{I})$$

- We denote this fixpoint $T_P^\omega(\mathtt{I})$

Theorem

The least fixpoint $T_P^\omega(\mathtt{I})$ is equal to the minimal model $P(\mathtt{I})$

Example - Transitive closure

$$T(x, y) \leftarrow G(x, y)$$
$$T(x, y) \leftarrow G(x, z), T(z, y)$$
$$\mathtt{I} = \{G(0, 1), G(1, 2), G(2, 3)\}$$

- It yields to:

$$T_P(\mathtt{I}) = \mathtt{I} \cup \{T(0, 1), T(1, 2), T(2, 3)\}$$
$$T_P^2(\mathtt{I}) = T_P(\mathtt{I}) \cup \{T(0, 2), T(1, 3)\}$$
$$T_P^3(\mathtt{I}) = T_P^2(\mathtt{I}) \cup \{T(0, 3)\} \quad \text{and} \quad T_P^4(\mathtt{I}) = T_P^3(\mathtt{I})$$

- $T_P^\omega(\mathtt{I}) = T_P^3(\mathtt{I})$

Example - Transitive closure (con't)

$T_P^\omega(\mathtt{I}) = T_P^3(\mathtt{I})$

G		T	
0	1	0	1
1	2	1	2
2	3	2	3
		0	2
		1	3
		0	3

$\mathcal{F}$

G		T	
0	1	0	1
1	2	1	2
2	3	2	3
3	2	0	2
		1	3
		0	3
		3	2
		3	3

$\mathcal{M}$

G		T	
0	1	0	1
1	2	1	2
2	3	2	3
		0	2
		1	3
		0	3
		3	2

$\mathcal{M}$ is even not a fixpoint!

$\mathcal{F}$ is not the least fixpoint

Evaluation of Datalog programs

- Lots of research in the late 80'th
- **Top-down** or **bottom-up** evaluation
- Direct evaluation vs. compilation into a more efficient program
- No product!
- Some influence on logic programming
- Renewal with knowledge bases (Linked Data)
- To come:
  1. **Semi-naive bottom-up evaluation**
  2. Top-down: QSQ
  3. Bottom-up: Magic

Reverse-Same-Generation

$$\mathrm{rsg}(x, y) \leftarrow \mathrm{flat}(x, y)$$
$$\mathrm{rsg}(x, y) \leftarrow \mathrm{up}(x, x_1), \mathrm{rsg}(y_1, x_1), \mathrm{down}(y_1, y)$$

up

a	e
a	f
f	m
g	n
h	n
i	o
j	o

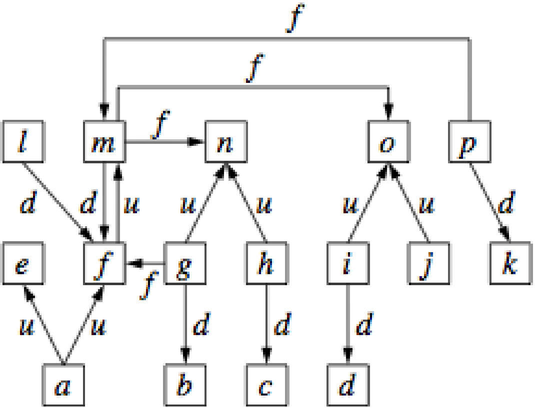
flat

g	f
m	n
m	o
p	m

down

l	f
m	f
g	b
h	c
i	d
p	k

Graphically



Naive algorithm

$$\begin{array}{ll} \text{rsg} & := \emptyset \\ \text{repeat} & \\ \quad \text{rsg} & := \text{rsg} \ \cup \ \text{flat} \ \cup \ \pi_{16}\Big(\sigma_{2=4}(\sigma_{3=5}(\text{up} \times \text{rsg} \times \text{down}))\Big) \\ \text{until} & \text{fixpoint} \end{array}$$

Language equivalence

SPCU + `while` loop = Datalog expressive power

Naive algorithm (cont'd)

One single step computation

$$\text{rsg}^{i+1} = \text{rsg}^{i} \cup \text{ flat } \cup \pi_{16}\Big(\sigma_{2=4}(\sigma_{3=5}(\text{up} \times \text{rsg}^{i} \times \text{down}))\Big)$$

where $\text{rsg}^{i} := T^{i}_{RSG}(\mathtt{I})(\text{rsg})$

- Evaluation example:
  - level 0: $\emptyset$
  - level 1: $\{(g, f), (m, n), (m, o), (p, m)\}$
  - level 2: $\{\text{level } 1\} \cup \{(a, b), (h, f), (i, f), (j, f), (f, k)\}$
  - level 3: $\{\text{level } 2\} \cup \{(a, c), (a, d)\}$
  - level 4: $\{\text{level } 3\}$

Limitations

- **Redundant** computation
  - Each layer recomputes all elements of the previous layer
  - $\mathsf{rsg}^i \subseteq \mathsf{rsg}^{i+1}$
- **Inflation**: both the problem and the solution

Semi-naive algorithm

- Focus on the **new facts** generated at each step
- New expression: RSG'

$$\Delta^1_{\text{rsg}}(x, y) \leftarrow \text{flat}(x, y)$$
$$\left[\quad \Delta^{i+1}_{\text{rsg}}(x, y) \leftarrow \text{up}(x, x_1), \Delta^{i}_{\text{rsg}}(y_1, x_1), \text{down}(y_1, y) \quad \right]_{i \geq 1}$$

- No more recursive, even not a Datalog program. . .
- For each input $\mathtt{I}$, new facts are
$$\text{rsg}^{i+1} - \text{rsg}^{i} \ \subseteq \delta^{i+1}_{\text{rsg}} = RSG'(\mathtt{I})(\Delta^{i+1}_{\text{rsg}}) \subseteq \text{rsg}^{i+1}$$
- Ultimate answer set: $\text{RSG}(\mathtt{I})(\text{rsg}) = \bigcup_{1 \leq i}(\delta^{i}_{\text{rsg}})$
- Much less redundancy

Improved semi-naive algorithm

- Knowing: $\delta^{i+1}_{\text{rsg}} \neq \text{rsg}^{i+1} - \text{rsg}^{i}$

$$\text{e.g., } (g, f) \in \delta^{2}_{\text{rsg}}, \text{not in } \text{rsg}^2 - \text{rsg}^1$$

- Use $\text{rsg}^{i} - \text{rsg}^{i-1}$ instead of Δ^{i}_{rsg} in the second "rule" of RSG'

$$\begin{bmatrix} \Delta^1_{\text{rsg}}(x,y) & \leftarrow & \text{flat}(x,y) \\ \text{rsg}^1 & := & \Delta^1_{\text{rsg}} \end{bmatrix}$$

$$\begin{bmatrix} \text{temp}^{i+1}_{\text{rsg}}(x,y) & \leftarrow & \text{up}(x,x_1), \Delta^{i}_{\text{rsg}}(y_1,x_1), \text{down}(y_1,y) \\ \Delta^{i+1}_{\text{rsg}} & := & \text{temp}^{i+1}_{\text{rsg}} - \text{rsg}^{i} \\ \text{rsg}^{i+1} & := & \text{rsg}^{i} \cup \Delta^{i+1}_{\text{rsg}} \end{bmatrix}_{i \geq 1}$$

Non linear rules

Example: ancestor

$$\mathrm{anc}(x, y) \leftarrow \mathrm{par}(x, y)$$
$$\mathrm{anc}(x, y) \leftarrow \mathrm{anc}(x, z), \mathrm{anc}(z, y)$$

- Semi-naive evaluation:

$$\left[\begin{array}{lcl} \Delta^1_{\mathrm{anc}}(x, y) & \leftarrow & \mathrm{par}(x, y) \\ \mathrm{anc}^1 & := & \Delta^1_{\mathrm{anc}} \end{array}\right]$$
$$\left[\begin{array}{lcl} \mathrm{temp}^{i+1}_{\mathrm{anc}}(x, y) & \leftarrow & \Delta^i_{\mathrm{anc}}(x, z), \mathrm{anc}(z, y) \\ \mathrm{temp}^{i+1}_{\mathrm{anc}}(x, y) & \leftarrow & \mathrm{anc}(x, z), \Delta^i_{\mathrm{anc}}(z, y) \\ \Delta^{i+1}_{\mathrm{anc}} & := & \mathrm{temp}^{i+1}_{\mathrm{anc}} - \mathrm{anc}^i \\ \mathrm{anc}^{i+1} & := & \mathrm{anc}^i \cup \Delta^{i+1}_{\mathrm{anc}} \end{array}\right]_{i \geq 1}$$

Non linear rules (cont'd)

Still some redundancy

Suppose an input instance par $= \{(1,2),(2,3)\}$;

$$\Delta^1_{\text{anc}} = \{(1,2),(2,3)\}$$
$$\text{anc}^1 = \{(1,2),(2,3)\}$$
$$\Delta^2_{\text{anc}} = \{(1,3)\}$$

- Both rules for $\text{temp}^2_{\text{anc}}$ compute the join of tuples $(1,2)$ and $(2,3)$
- **Redundant computation** of $(2,3)$

Non linear rules (cont'd)

New rules

Use instead of the two rules for $\text{temp}^{i+1}_{\text{anc}}$:

$$\text{temp}^{i+1}_{\text{anc}}(x, y) \leftarrow \Delta^{i}_{\text{anc}}(x, z), \text{anc}^{i-1}(z, y)$$
$$\text{temp}^{i+1}_{\text{anc}}(x, y) \leftarrow \text{anc}^{i}(x, z), \Delta^{i}_{\text{anc}}(z, y)$$

Static program analysis

Satisfiability

Is there a db instance $\mathtt{I}$ such that for each R in $P(\mathtt{I})$, $P(\mathtt{I})(R)$ is non-empty?

- Remind that RC query satisfiability is undecidable
- Datalog is somehow both:
  - more powerful than RC (with recursion) and
  - less powerful than RC (w/o negation)
- Satisfiability is **decidable** for Datalog programs!

Static program analysis (cont'd)

Containment

- For each db instance $\mathtt{I}$, is $P(\mathtt{I})(R)$ included into $P'(\mathtt{I})(R)$ for all R?
- Query optimization purpose, coupled with boundedness

Boundedness

- The fixpoint is reached after a bounded number of steps
- More optimization...

Boundedness

Definition (Stage)

The smallest integer i such that $T_P^i(\mathtt{I}) = T_P^\omega(\mathtt{I})$ is called the **stage** for P and $\mathtt{I}$

Property

nr-Datalog programs are **bounded**:

$$\exists d, \forall \mathtt{I} \text{ over } \mathrm{edb}(P), \mathrm{stage}(P, \mathtt{I}) \leq d$$

More about boundedness

Falsely recursive program

$$Q(x,y) \leftarrow R(x,y)$$
$$Q(x,y) \leftarrow S(x), Q(x,y)$$

- R may substitute to Q in the body
- **Bounded** Datalog program

Truly recursive program

$$Q(x,y) \leftarrow R(x,y)$$
$$Q(x,y) \leftarrow T(x,z), Q(z,y)$$

- **Unbounded** Datalog program

A couple of more results and statements

- Containment is **undecidable**
- Boundedness is **undecidable**
- Optimization will be difficult
- Heuristics are welcome!

Exercises 1/2

1. Definitions

Immediate consequence, Fixpoint, Boundedness, Stage, Naive evaluation, Semi-naive evaluation, Non linear rules

2. True or False?

i) Immediate consequence operator is monotonic.
ii) Each model of Σ_P is a fixpoint of T_P.
iii) There are $|\mathcal{B}(P, \mathtt{I})|$ steps to reach the fixpoint.
iv) Semi-naive evaluation does not compute redundant fact.
v) Datalog is equivalent to SPCU.
vi) Satisfiability is decidable for Datalog programs.

Exercises 2/2

3. Reverse-Same-Generation

1. Show all the redondant facts from RSG' w/o improvement.
2. Exhibit an instance I such that RSG' satisfies $\delta^i_{\mathsf{rsg}} \neq \emptyset, \forall i > 0$.

4. Problem

In the presence of many idb's in the program, show that there exists an evaluation ordering that prevents from redondant fact computation.
Hint: analyze the structure of the Datalog program b.t.w. of mutual recursion (predicates onto the same cycle in the dependency graph).

Proof-theoretic semantics

A **top-down** execution of a Datalog program

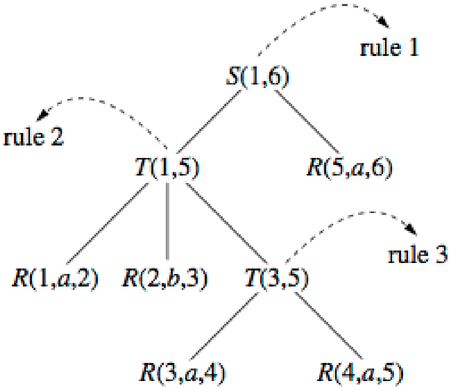
> *Each goal in a rule body is viewed as a call to a procedure defined by other rules*

Example

$$S(x_1, x_3) \leftarrow T(x_1, x_2), R(x_2, a, x_3)$$
$$T(x_1, x_4) \leftarrow R(x_1, a, x_2), R(x_2, b, x_3), T(x_3, x_4)$$
$$T(x_1, x_3) \leftarrow R(x_1, a, x_2), R(x_2, a, x_3)$$
$$\mathtt{I} = \{R(1, a, 2), R(2, b, 3), R(3, a, 4), R(4, a, 5), R(5, a, 6)\}$$

Query: $\leftarrow S(1, 6)$

Proof tree



(a) Datalog proof

Horn clause

Equivalent formula of a Datalog rule $R(\vec{u}) \leftarrow R_1(\vec{u_1}), \ldots R_n(\vec{u_n})$

$$\forall x_1, \ldots, x_m : \ \left(R(\vec{u}) \vee \neg R_1(\vec{u_1}) \vee \ldots \vee \neg R_n(\vec{u_n})\right)$$

Definition (Horn clause)

Disjunction of literals of which at most one is positive

Definite Horn clause has exactly one positive literal

General statement

A Datalog program is a set of definite Horn clauses

Logic programming terminology

More about clauses

definite	$T(x, y) \leftarrow R(x, z), T(z, y)$	$T(x, y) \vee \neg R(x, z) \vee \neg T(z, y)$
empty	$\square$	False
unit	$T(x, y) \leftarrow$	$T(x, y)$
goal	$\leftarrow R(x, z), T(z, y)$	$\neg R(x, z) \vee \neg T(z, y)$

- A **ground** clause has no occurrence of variables

$q : \leftarrow S(1, 6)$

Query q is a goal equivalent to formula $\neg S(1, 6)$

Then, to prove $S(1, 6)$, it suffices to exhibit a **refutation** of q

A way to deriving proofs: SLD resolution

Warm-up

A program with only ground rules and ground facts

$$
\begin{array}{ll}
(1) & S(1,6) \leftarrow T(1,5), R(5,a,6) \\
(2) & T(1,5) \leftarrow R(1,a,2), R(2,b,3), T(3,5) \\
(3) & T(3,5) \leftarrow R(3,a,4), R(4,a,5) \\
(4) & R(1,a,2) \leftarrow \\
(5) & R(2,b,3) \leftarrow \\
(6) & R(3,a,4) \leftarrow \\
(7) & R(4,a,5) \leftarrow \\
(8) & R(5,a,6) \leftarrow
\end{array}
$$

Technique: refutation

	Goal	Used rule
	$\neg S(1,6)$	
$\Rightarrow$	$\neg T(1,5) \vee \neg R(5,a,6)$	(1)
$\Rightarrow$	$\big(\neg R(1,a,2) \vee \neg R(2,b,3) \vee \neg T(3,5)\big) \vee \neg R(5,a,6)$	(2)
$\Rightarrow$	$\neg R(2,b,3) \vee \neg T(3,5) \vee \neg R(5,a,6)$	(4)
$\Rightarrow$	$\neg T(3,5) \vee \neg R(5,a,6)$	(5)
$\Rightarrow$	$\big(\neg R(3,a,4) \vee \neg R(4,a,5)\big) \vee \neg R(5,a,6)$	(3)
$\Rightarrow$	$\neg R(4,a,5) \vee \neg R(5,a,6)$	(6)
$\Rightarrow$	$\neg R(5,a,6)$	(7)
$\Rightarrow$	false	(8)

SLD resolution

We start with a program (inc. the db facts): $P(\mathtt{I}) = P_{\mathtt{I}}(\{\})$

$$
\begin{aligned}
&S(x_1, x_3) \leftarrow T(x_1, x_2), R(x_2, a, x_3)\\
&T(x_1, x_4) \leftarrow R(x_1, a, x_2), R(x_2, b, x_3), T(x_3, x_4)\\
&T(x_1, x_3) \leftarrow R(x_1, a, x_2), R(x_2, a, x_3)\\
&R(1, a, 2) \leftarrow\\
&R(2, b, 3) \leftarrow\\
&R(3, a, 4) \leftarrow\\
&R(4, a, 5) \leftarrow\\
&R(5, a, 6) \leftarrow
\end{aligned}
$$

Resolution

Build all proof trees of idb's by means of **goal unification**

Unification

Unifier for two atoms A, B

A **substitution** θ such that $\theta A = \theta B$

Example: $P(x_1, x_2, a)$ and $P(b, x_3, x_4)$ are unifiable b.t.w. substitution $\alpha = \{x_1/b, x_2/c, x_3/c, x_4/a\}$

Definition (Most general unifier—mgu—θ)

For each unifier α, there is a substitution β such that $\alpha = \theta\beta$

Example cont'd: mgu is $\theta = \{x_1/b, x_2/x_3, x_4/a\}$

Goal unification

One step of resolution

Definition (Resolvent)

Given $r : A \leftarrow B_1, \dots B_n$ and $\leftarrow g$ (no variables in common), if there exists an mgu θ for A and g, then the **resolvent** of r and g is $\leftarrow \theta B_1, \dots, \theta B_n$

Example:

$$S(x_1, x_3) \leftarrow T(x_1, x_2), R(x_2, a, x_3)$$
$$\leftarrow S(1, 6)$$

- mgu $\theta = \{x_1/1, x_3/6\}$
- Resolvent is then $\leftarrow T(1, x_2), R(x_2, a, 6)$

Goal unification (cont'd)

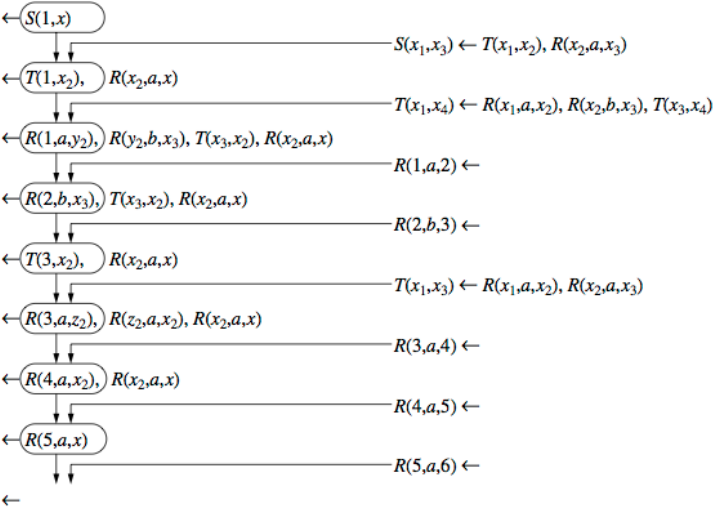
Sketch of the algorithm main loop:

1. Goal: $\leftarrow A_1, \ldots, A_i, \ldots, A_m$
2. Rule: $B_1 \leftarrow B_2, \ldots, B_n$
3. θ: mgu of A_i and B_1
4. New goal: $\leftarrow \theta A_1, \ldots, \theta A_{i-1}, \theta B_2, \ldots, \theta B_n, \theta A_{i+1}, \ldots, \theta A_m$

Stop condition:

- failure (no rule), or
- empty goal, then result $= \theta_1 \ldots \theta_k$init. goal

SLD refutation



SLD refutation (cont'd)

This also provides **binding** for the variables of the original goal

Example

$\leftarrow T(x,y)$ provides all the pairs (x,y) such that $\neg T(x,y)$ is false, i.e., $T(x,y)$ holds

Soundness and completeness

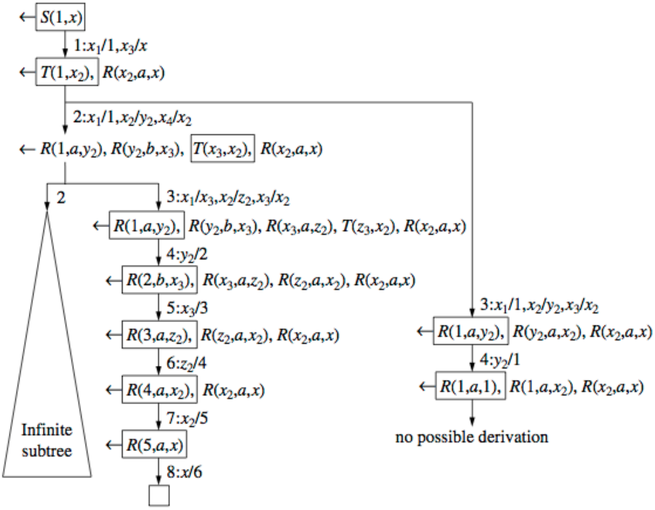
(a,b) is in the answer to $\leftarrow T(x,y)$ iff (a,b) is in $P(\mathtt{I})(T)$

SLD trees

- Top-down technique: success on $\square$ (any branch)
- Nondeterminism:
    1. Which atom to select in the current goal?
    2. Which rule to use?
- Use a **selection rule** to select goal
- Prolog selects the leftmost atom of the goal
- Once an atom has been selected, try **all possible rules**

SLD stands for *Selection rule-driven Linear resolution for Definite clauses*...

Example of a SLD tree



Bottom-up or top-down?

- **Top-down pro's**:
  - Take benefits from constants and constraints in unification
- **Top-down con's**:
  - Infinite loop for transitive closure
- Deductive DB mix bottom-up and top-down approaches in their evaluation techniques

Exercises 1/2

1. Definitions

Horn clause, Proof tree, Refutation, Unification, Mgu, Resolvent

2. True or False?

i) Whatever the semantics, result set remains the same.

ii) SLD resolution is bottom-up.

iii) SLD resolution assumes Edb's.

iv) Goal unification is the basic building block for resolution.

v) Mgu's are unique.

Exercises 2/2

3. SLD refutation

Give an SLD refutation for:

1. $\leftarrow S(x, 6)$
2. $\leftarrow \mathrm{rsg}(a, d)$

4. Problem

Let A, B be atoms; prove the three following properties:

1. If there exists a unifier for A, B, then A, B have an mgu.
2. If θ and θ' are mgu's for A, B then $\theta \sim \theta'$.
3. Let A, B be atoms with mgu θ. Then for each atom C, if $C = \theta_1 A = \theta_2 B$ for substitutions θ_1, θ_2, then $C = \theta_3 \circ \theta(A) = \theta_3 \circ \theta(B)$ for some substitution θ_3.

Outline

The logic of query languages

The power of recursion

Fixpoint semantics

Proof-theoretic semantics

Negation within recursion

SQL recursive queries

Datalog with negation

- Transitive closure

$$T(x, y) \leftarrow G(x, y)$$
$$T(x, y) \leftarrow G(x, z), T(z, y)$$

- Complement CT of T (pairs of disconnected nodes in G)

$$CT(x, y) \leftarrow \neg T(x, y)$$

- For convenience, assume an **active domain** interpretation
- Datalog$^\neg$
  - Allow in bodies of rules, literals of the form $\neg R_i(\vec{u_i})$
  - $\neg = (x, y)$ is denoted by $x \neq y$

Fixpoint semantics

- Notation : $\mathtt{J}|R$ is restriction of $\mathtt{J}$ to R
- Extend the immediate consequence operator
- For $\mathtt{K}$ over $\mathcal{P}$, a fact F is in $T_P(\mathtt{K})$ if
  - $F \in \mathtt{K}|\text{edb}(P)$, or
  - $F \leftarrow F_1, \ldots, F_n$ an instantiation of a rule in P such that
    1. if F_i is a positive literal then $F_i \in \mathtt{K}$
    2. if $F_i = \neg G_i$ then $G_i \notin \mathtt{K}$
- T_P is **no more inflationary**: $\mathtt{K} \not\subseteq T_P(\mathtt{K})$

Problems

- T_P may not have any fixpoint

$$P_1 = \{p \leftarrow \neg p\}$$

- T_P may have several least fixpoints containing $\mathtt{I}$

$$P_2 = \{p \leftarrow \neg q; q \leftarrow \neg p\}$$

  - Two least fixpoints (containing $\emptyset$): $\{p\}$ and $\{q\}$
  - Bottom-up fixpoint evaluation gives $T_{P_2}^{\omega}(\emptyset) = \{p, q\}$

Problems (cont'd)

- T_P has a least fixpoint but sequence $\{T_P^i(\emptyset)\}_{i>0}$ diverges

$$P_3 = \{p \leftarrow \neg r; r \leftarrow \neg p; p \leftarrow \neg p, r\}$$

  - T_{P_3} has a least fixpoint $\{p\}$
  - $\{T_{P_3}^i(\emptyset)\}_{i>0}$ alternates between $\emptyset$ and $\{p, r\}$
- T_P has a least fixpoint and $\{T_P^i(\emptyset)\}_{i>0}$ converges to something else

$$P_4 = p \leftarrow p, q \leftarrow q, p \leftarrow \neg p, q \leftarrow \neg p$$

  - $\{T_{P_4}^i(\emptyset)\}_{i>0}$ converges to $\{p, q\}$
  - Least fixpoint of T_{P_4} is $\{p\}$

Problems (cont'd)

Rules of the form $P(x, y) \leftarrow P(x, y)$

- Change the semantics of program
- Force T_P to be inflationary so force convergence
- Correspond to tautologies $p \vee \neg p$

Model-theoretic semantics: Problems

- Some programs have no model
- Some have no minimal model containing `I`
- When a program has several minimal models, choose between them

Potpourri

Let P be a program without negation then

1. intersection of two models of P is a model of P
2. P has a minimal model
3. T_P is monotonic

On the other hand consider the program $P = \{a \leftarrow \neg b\}$:

1. $\{a\}$ and $\{b\}$ are models of P but $\emptyset = \{a\} \cap \{b\}$ is not
2. $\{a\}$ and $\{b\}$ are two distinct minimal models of P
3. $T_P(\emptyset) = \{a\}$, and $T_P(\{b\}) = \emptyset$. Thus T_P is not monotonic ($\emptyset \subseteq \{b\}$, but $\{a\} \not\subseteq \emptyset$)

Semi-positive datalog$^\neg$

- Only apply negation to edb relations
- Semi-positive program that is neither in Datalog nor in RC:

$$
\begin{aligned}
T(x,y) &\leftarrow \neg G(x,y), \mathrm{Adom}(x), \mathrm{Adom}(y)\\
T(x,y) &\leftarrow \neg G(x,z), T(z,y), \mathrm{Adom}(x)
\end{aligned}
$$

Intuition

One could eliminate negation from semi-positive programs by adding, for each edb relation R, a new edb relation $\bar{R}$ holding the complement of R (w.r.t. the active domain), and replacing $\neg R(x)$ by $\bar{R}(x)$

Semi-positive datalog¬ (cont'd)

Inheritance

Many nice properties of positive Datalog

- Σ_P has a unique minimal model $\mathtt{J}$ satisfying $\mathtt{J}|\,\mathrm{edb}(P) = \mathtt{I}$
- T_P has a unique least fixpoint $\mathtt{J}$ satisfying $\mathtt{J}|\,\mathrm{edb}(P) = \mathtt{I}$
- Both semantics coincide

Limitation

$$
\begin{aligned}
r_1 :& T(x, y) \leftarrow G(x, y)\\
r_2 :& T(x, y) \leftarrow G(x, z), T(z, y)\\
r_3 :& CT(x, y) \leftarrow \neg T(x, y), \mathrm{Adom}(x), \mathrm{Adom}(y)
\end{aligned}
$$

- CT **is not** a semi-positive program

Extension

Key idea

Parse P as a sequence of semi-positive subprograms $(P^1, \ldots, P^n)$

- CT is $(\{r_1, r_2\}, \{r_3\})$
- Closure under composition
- **Stratified** Datalog$^{\neg}$

Stratified Datalog¬

Definition (Stratification of a Datalog¬ program P)

Sequence of Datalog¬ programs $(P^1, \ldots, P^n)$ and some mapping σ from idb(P) to $[1..n]$ such that:

(i) $\{P^1, \ldots, P^n\}$ is a partition of P

(ii) For each R, all rules defining R are in $P^{\sigma(R)}$

(iii) If $R(\vec{u}) \leftarrow \ldots S(\vec{v}) \ldots$ is a rule in P, and S is an idb relation, then $\sigma(S) \leq \sigma(R)$

(iv) If $R(\vec{u}) \leftarrow \ldots \neg S(\vec{v}) \ldots$ is a rule in P, and S is an idb relation, then $\sigma(S) < \sigma(R)$

- Each P^i is called a **stratum**

Stratification example

- Stratification of CT

$$\begin{aligned} r_1 &: T(x,y) \leftarrow G(x,y) \\ r_2 &: T(x,y) \leftarrow G(x,z), T(z,y) \\ r_3 &: CT(x,y) \leftarrow \neg T(x,y), \mathrm{Adom}(x), \mathrm{Adom}(y) \end{aligned}$$

- Constraints:
  - r_1 and r_2 are both in $P^{\sigma(T)}$ by (ii)
  - $\sigma(T) < \sigma(CT)$ by (iv)
- Strata:
  - First stratum: $P^1 = \{r_1, r_2\}$ (defining T)
  - Second stratum: $P^2 = \{r_3\}$ (defining CT using T)

More Stratification examples

- P_7 defined by

$$
\begin{aligned}
r_1: &\quad S(x) \leftarrow R_2(x), \neg R_1(x)\\
r_2: &\quad T(x) \leftarrow R_3(x), \neg R_1(x)\\
r_3: &\quad U(x) \leftarrow R_4(x), \neg T(x)\\
r_4: &\quad V(x) \leftarrow R_5(x), \neg S(x), \neg U(x)
\end{aligned}
$$

- Then P_7 has 5 distinct stratifications, namely,

$$
\begin{aligned}
&(\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\})\\
&(\{r_2\}, \{r_1\}, \{r_3\}, \{r_4\})\\
&(\{r_2\}, \{r_3\}, \{r_1\}, \{r_4\})\\
&(\{r_1, r_2\}, \{r_3\}, \{r_4\})\\
&(\{r_2\}, \{r_1, r_3\}, \{r_4\})
\end{aligned}
$$

- $P_2 = \{p \leftarrow \neg q, q \leftarrow \neg p\}$ has no stratification

Testing stratification

Definition (Precedence graph $\mathcal{G}_P$ of P revisited)

- Vertices are the idb's of P
- Edge (R, S) with label '$+$' if S is used positively in some rule defining R
- Edge (R, S) with label '-' if S occurs negatively in some rule defining R

P is **stratifiable** iff $\mathcal{G}_P$ has no cycle containing a negative edge

Testing stratification (cont'd)

Sketch of the proof

- Let P be a program whose precedence graph $\mathcal{G}_P$ has no cycle with negative edges;
- $C_1, \ldots, C_n$ are the strongly connected components of $\mathcal{G}_P$
- $C_i \prec C_j$ if there is an edge from C_i to some node of C_j, where $\prec$ is acyclic
- Turn this partial order into a sort $(C_{i_1}, \ldots, C_{i_n})$
- This provides a stratification

Stratification: semantics

- Given P a program with stratification $\sigma = (P^1, \ldots, P^n)$ and $\mathtt{I}$ an instance
  - $\mathtt{J}_0 = \mathtt{I}$
  - $\mathtt{J}_i = \mathtt{J}_{i-1} \cup P^i(\mathtt{J}_{i-1}|\,\mathrm{edb}(P^i))$
    where $P^i(\mathtt{J})$ is the semi-positive semantics
  - $\mathtt{J}_n$ is denoted $\sigma(\mathtt{I})$, the semantics of P under σ

Theorem

All stratifications σ of a Datalog$^\neg$ program P are equivalent

Actually, we denote $P^{\mathrm{strat}}(\mathtt{I})$ the semantics of a stratified Datalog$^\neg$ program P, whatever would be its stratification σ

Results

Given P a stratified Datalog$^\neg$ program, and $\mathtt{I}$ an instance

1. $P^{\text{strat}}(\mathtt{I})$ is a **minimal model** of Σ_P whose restriction to edb(P) equals $\mathtt{I}$
2. $P^{\text{strat}}(\mathtt{I})$ is a **least fixpoint** of T_P whose restriction to edb(P) equals $\mathtt{I}$
3. $P^{\text{strat}}(\mathtt{I})$ is a "supported" model of P relative to $\mathtt{I}$ ($P^{\text{strat}}(\mathtt{I}) \subseteq T_P(P^{\text{strat}}(\mathtt{I})) \cup \mathtt{I}$)

- Limited power w.r.t.—full—Datalog$^\neg$

Exercises 1/2

1. Definitions

T_P, Inflationary property, Semi-positive Datalog$^\neg$, Stratified Datalog$^\neg$, Stratum, Precedence graph

2. True or False?

i) T_P for Datalog$^\neg$ remains inflationary.
ii) T_P for Datalog$^\neg$ may have several least fixpoints.
iii) T_P for semi-positive Datalog$^\neg$ remains inflationary.
iv) Stratified Datalog$^\neg$ is equivalent to Datalog$^\neg$.
v) Semantics depends on the choice for a stratification.

Exercises 2/2

3. Practical Datalog$^\neg$ ✏

1. Draw the precedence graph for P_2, P_7 and CT.
2. Compute CT on $\mathtt{I} = \{G(0,1), G(1,2), G(2,3)\}$ and follow the stratified semantics.

4. Problem

1. Exhibit a Datalog$^\neg$ program P and an instance $\mathtt{K}$ over $\mathcal{P}$ such that $\mathtt{K}$ is a model of Σ_P but not a fixpoint of T_P.
2. Show that, for Datalog$^\neg$ programs P, a least fixpoint of T_P is not necessarily a minimal model of Σ_P and, conversely, a minimal model of Σ_P is not necessarily a least fixpoint of T_P.

Outline

SQL-99 recursion

- Datalog recursion has inspired the addition of recursion to the SQL-99 standard
- Tricky, because SQL allows negation, grouping and aggregation, which interact with recursion in strange ways
- Form of SQL recursive queries:

```
WITH <stuff that looks like Datalog rules>
<a SQL query about EDB, IDB> ;
```

- "Datalog rule":

```
[RECURSIVE] <name>(<arguments>) AS <query>
```

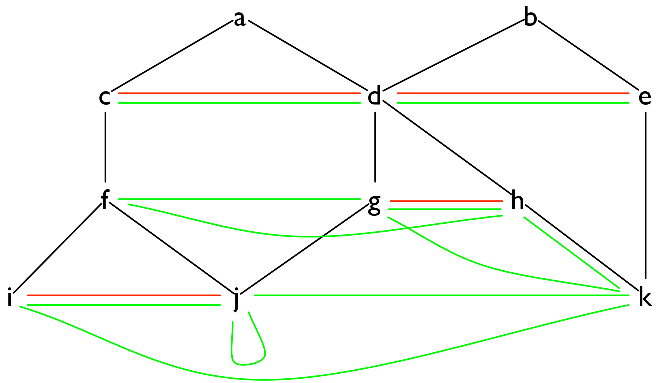
From Datalog to SQL

- Datalog program P_{cous}:

$$\begin{array}{lll} r_1: & \text{sib}(x,y) & \leftarrow \text{par}(x,z), \text{par}(y,z), x \neq y \\ r_2: & \text{cousin}(x,y) & \leftarrow \text{sib}(x,y) \\ r_3: & \text{cousin}(x,y) & \leftarrow \text{par}(x,x_1), \text{par}(y,y_1), \text{cousin}(x_1,y_1) \end{array}$$

- Generalized cousins: people with common ancestors

P_{cous} on instance `I` of par



black lines are instance `I`, red lines are *sib* and green lines are *cousin*

Example: SQL Recursion

- Find Sally's cousins
- par(child,parent) is edb(P_{cous})

```
WITH sib(x,y) AS
    SELECT p1.child, p2.child
    FROM par p1, par p2
    WHERE p1.parent = p2.parent AND
          p1.child <> p2.child,
```

- Translates r_1

Example: SQL Recursion (cont'd)

- Recursive part:

```
WITH ...
RECURSIVE cousin(x,y) AS
        ( SELECT * FROM Sib )
              UNION ALL
        ( SELECT p1.child, p2.child
          FROM par p1, par p2, cousin
          WHERE p1.parent = cousin.x AND
                p2.parent = cousin.y )
```

- Translates $r_2 \cup r_3$

Example: SQL Recursion (cont'd)

With those definitions, we can add the query, which is about the virtual view cousin(x, y):

```
WITH ...
SELECT y FROM cousin WHERE x = 'Sally';
```

Legal SQL recursion

- It is possible to syntactically define SQL recursions that does not have a meaning
- The SQL standard restricts recursion so there is a meaning
- And that meaning can be obtained by **semi-naive evaluation with restrictions**

Key point

Legal SQL recursion is given by stratified programs only

Stratification of SQL recursive queries

- Vertices are
  - idb's declared in `WITH` clause
  - SQL subqueries in the body of the "rules" (any level of nesting)
- Edges (P, Q):
  1. P is a rule head and Q is a relation (but subqueries) in the `FROM` list
  2. P is a rule head and Q is an immediate subquery of that rule
  3. P is a subquery, and Q is a relation in its `FROM` list or an immediate subquery (like 1 and 2)
- Negation symbol whenever P is **not monotonic** in Q

Monotonicity

If relation P is a function of relation Q, we say P is monotonic in Q if inserting tuples into Q cannot cause any tuple to be deleted from P

Examples

Monotonic dependency:

$$P = Q \cup R$$
$$P = \sigma_{a=10}(Q)$$

Non-monotonic dependency:

```
SELECT AVG(grade) FROM Enroll WHERE course='cs143';
```

Nice Shot from David Fetter, 2009

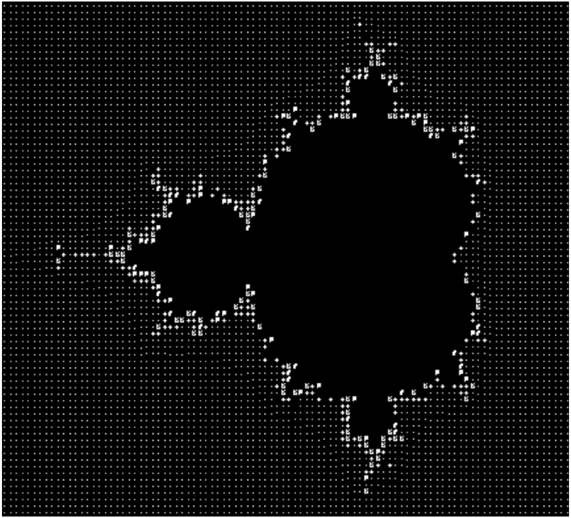
Mandelbrot-set written entirely in SQL2008-conformant SQL and powered by PostgreSQL 8.4

```
WITH RECURSIVE
x(i)
AS (
    VALUES(0)
UNION ALL
    SELECT i + 1 FROM x WHERE i < 101
),

Z(Ix, Iy, Cx, Cy, X, Y, I)
AS (
    SELECT Ix, Iy, X::float, Y::float, X::float, Y::float, 0
    FROM
        (SELECT -2.2 + 0.031 * i, i FROM x) AS xgen(x,ix)
    CROSS JOIN
        (SELECT -1.5 + 0.031 * i, i FROM x) AS ygen(y,iy)
    UNION ALL
    SELECT Ix, Iy, Cx, Cy, X * X - Y * Y + Cx AS X,
           Y * X * 2 + Cy, I + 1
    FROM Z
    WHERE X * X + Y * Y < 16.0
    AND I < 27
),
```

Nice Shot from David Fetter (cont'd)

```
Zt (Ix, Iy, I) AS (
    SELECT Ix, Iy, MAX(I) AS I
    FROM Z
    GROUP BY Iy, Ix
    ORDER BY Iy, Ix
)

SELECT array_to_string(
    array_agg(
        SUBSTRING(
            ' .,,,-----++++%%%%@@@@#### ',
            GREATEST(I,1),
            1
        )
    ),''
)
FROM Zt
GROUP BY Iy
ORDER BY Iy;
```

Nice shot from David Fetter (cont'd)



Exercises 1/2

1. Definitions

WITH statement, Legal SQL recursion, Monotonicity, SQL Precedence graph

2. True or False?

i) SQL “WITH” queries could be expressed by regular S-F-W queries.
ii) Agregate functions (avg, sum, max, min) are not allowed in SQL recursive queries.
iii) Semantics of a SQL recursive query is the one from the stratified Datalog¬.

Exercises 2/2

3. Misc.

1. Draw the precedence graph for program cousin.
2. Build a non-legal SQL recursive query.

4. Problem

Given a relation Vertice(from, to, cost) that represents a non-directed labelled graph, give the SQL query that computes the shortest path from node A.

Readings

- Serge Abiteboul, Richard Hull, and Victor Vianu. **Foundations of Databases**. Addison-Wesley, 1995.

  Chapters 12, 13, 15.

- Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom. **Database Systems: The Complete Book** (2nd edition), Prentice Hall, 2008.

  Chapters 5.3, 5.4.