Notes for ECS 289F: Foundations of Relational Databases

Winter 2010

Revision: March 6, 2010

1 Monday, 1/4/10 (scribe: Sven)

Introduction/Administrivia

Webpage:

`http://www.cs.ucdavis.edu/~green/courses/cse289f`

Textbook (optional, use as reference): *Foundation of Databases* by Abiteboul, Hull, and Vianu

Historical Background

(see handout from 1975 book by Date)

Hierarchical Approach

Problem: Redundancy, awkward when data not hierarchical, strong distinction between records and links, low-level query languages

Network Model

Problem: More flexible, but links are still seperate elements

Relational Approach

Everything (links, records) modeled as *relations*. High-level, declarative query language based on *first-order logic*.

2 Wednesday, 1/6/10 (scribe: Sarah)

Review: Syntax and Semantics of First-Order Logic (FO)

Definition 1. A *relational schema* σ is a collection of *predicate* symbols (relations) (denoted $P_1, P_2, \ldots$) each with an associated arity (number of columns).

A σ-*instance* I is a collection of *interpretations* of predicate symbols, one for each P_i in σ. i.e., if P_i is a k-ary predicate symbol then P_i^I (the *contents* of P_i) $\subseteq \mathbb{D}^k$ where $\mathbb{D}$ is the *domain* of the database and P_i^I is finite.

More generally, this is a *structure/model* where the vocabulary is a schema and the structure is an instance of a table (see below definitions).

Definition 2. A *vocabulary* σ is a collection of *constant* symbols (denoted $c_1, c_2, \dots$), *relation* or *predicate* symbols ($P_1, P_2, \dots$), and *function* symbols ($f_1, f_2, \dots$). Each relation and function symbol has an associated arity.

A σ-*structure* (also called a *model*)

$$\mathfrak{A} = \langle A, \{c_i^{\mathfrak{A}}\}, \{P_i^{\mathfrak{A}}\}, \{f_i^{\mathfrak{A}}\}\rangle$$

consists of a *universe* A together with an interpretation of

- each constant symbol c_i from σ as an element $c_i^{\mathfrak{A}} \in A$;
- each k-ary relation symbol P_i from σ as a k-ary relation on A; that is, a set $P_i^{\mathfrak{A}} \subseteq A^k$; and
- each k-ary function symbol f_i from σ as a function $f_i^{\mathfrak{A}} : A^k \to A$.

Next, we define first-order (FO) formulae, free and bound variables, and the semantics of FO formulae.

Definition 3 (Syntax of FO)**.** We assume a countably infinite set of variables $\{x_1, x_2, \dots\}$. We inductively define *terms* and *formulae* of the *first-order predicate calculus* over vocabulary σ as follows:

A *term* is defined as either:

- a variable x;
- a constant c;
- $f(t_1, t_2, \dots\ t_k)$ where t_i is a term for all i.

A *formula* is defined as either:

- $t_1 = t_2$ where t_1, t_2 are terms;
- $P(t_1, t_2, \dots\ t_k)$ where all t_i are terms and P is a predicate;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ or $\neg\varphi_1$ where φ_1, φ_2 are formulae;
- $\exists x \varphi$ or $\forall x \varphi$ where φ is a formula.

Note: $\varphi \to \psi$ means $\neg\varphi \vee \psi$ (implication)

Note: a formula of no free variables is called a *sentence*. (ex: $\forall x P_1(x) \vee \neg P_2(x)$).

Definition 4 (Semantics of FO)**.** Given a σ-structure $\mathfrak{A}$, we define inductively for each term t with free variables $(x_1, \dots, x_n)$ the value $t^{\mathfrak{A}}(\bar{a})$, where $\bar{a} \in A^n$, and for each formula $\varphi(x_1, \dots, x_n)$, the notion of $\mathfrak{A} \models \varphi(\bar{a})$ (i.e., $\varphi(\bar{a})$ is true in $\mathfrak{A}$), read $\mathfrak{A}$ *satisfies* $\varphi(\bar{a})$.

For *terms*:

- if t is a constant c, then $t^{\mathfrak{A}} = c^{\mathfrak{A}}$;
- if t is a variable x, then $t^{\mathfrak{A}}(\bar{a})$ is a_i;
- if $t = f(t_1, t_2, \dots t_k)$, then $t^{\mathfrak{A}}(\bar{a}) = f^{\mathfrak{A}}(t_1^{\mathfrak{A}}(\bar{a}), t_2^{\mathfrak{A}}(\bar{a}), \dots t_k^{\mathfrak{A}}(\bar{a}))$;

For *formulae*:

- if φ is $(t_1 = t_2)$, then $\mathfrak{A} \models \varphi(\bar{a})$ iff $t_1^{\mathfrak{A}}(\bar{a}) = t_1^{\mathfrak{A}}(\bar{a})$;
- if φ is $P(t_1, t_2, \dots t_k)$, then $\mathfrak{A} \models \varphi(\bar{a})$ iff $(t_1^{\mathfrak{A}}(\bar{a}), t_2^{\mathfrak{A}}(\bar{a}), \dots t_k^{\mathfrak{A}}(\bar{a})) \in P^{\mathfrak{A}}$;
- if φ is $\varphi_1 \wedge \varphi_2$, then $\mathfrak{A} \models \varphi(\bar{a}) \wedge \varphi(\bar{a})$ iff $\mathfrak{A} \models \varphi_1(\bar{a})$ AND $\mathfrak{A} \models \varphi(\bar{a})$; (etc for $\vee$ and $\neg$.)
- if φ is $\exists y \varphi$, then $\mathfrak{A} \models \exists y \varphi(y, \bar{a})$ iff $\mathfrak{A} \models \varphi(a', \bar{a})$ for some $a' \in A$. (etc for $\forall$.)

Definition 5 (Relational calculus query.)**.** A k-ary *relational calculus query* over schema σ is a mapping $Q : \sigma\text{-instances} \to \mathbb{D}^n$ given by an expression of the form

$$\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$$

where $\varphi(x_1, \ldots, x_n)$ is a FO formula. The *evaluation* $[\![Q]\!]^I$ of Q on σ-instance I is

$$[\![Q]\!]^I \stackrel{\text{def}}{=} \{(a_1, \ldots, a_n) \in \mathbb{D}^n \mid I \models \varphi(a_1, \ldots, a_n)\}$$

3 Friday, 1/8/10 (scribe: Anand)

Examples of relational calculus queries

Let us consider a schema having 3 relations: a ternary *Class* relation (first column "class ID", second column "class name", third column "room number"), a binary *Student* relation (first column "student ID", second column "student name"), and a binary *Takes* relation (first column "student ID", second column "class ID"). We also assume constants for names, IDs, etc.

Example 6. Q1: Find names of all students taking a class meeting in Wellman 201.

$$\{(x) \mid \exists s \exists c \exists n\ \mathit{Student}(s, x) \wedge \mathit{Takes}(s, c) \wedge \mathit{Class}(c, n, \text{“Wellman 201”})\}$$

Example 7. Q2: Find all pairs of students not taking a class together.

$$\{(x, y) \mid \exists s, s'\ \mathit{Student}(s, x) \wedge \mathit{Student}(s', y) \wedge \neg\exists c(\mathit{Takes}(s, c) \wedge \mathit{Takes}(s', c))\}$$

Review of first-order logic, continued

- Satisfaction of FO formulae (with free variables): $\mathfrak{A} \models \varphi(\bar{x})$ means $\mathfrak{A} \models \varphi(\bar{a})$ for all $\bar{a} \in A^k$
- If Γ be a set of FO sentences, then say $\Gamma \models \varphi$ holds if any structure $\mathfrak{A}$ satisfying Γ also satisfies φ
- Validity: $\models \varphi$ means $\mathfrak{A} \models \varphi$ for any structure $\mathfrak{A}$ (i.e., φ is *valid*)
- $\vdash$ means "provable"; $\Gamma \vdash \varphi$ means φ is provable from Γ.

Theorem 8 (**Gödel's Completeness Theorem**)**.**

$$\Gamma \models \varphi \quad \textit{iff} \quad \Gamma \vdash \varphi$$

As a corollary, we have:

$$\{(\Gamma, \varphi) \mid \Gamma \models \varphi\} \text{is} \quad \text{r.e.}$$

In particular, taking VALID to be the set of all valid FO sentences

$$\text{VALID} \stackrel{\text{def}}{=} \{\varphi \ \mid \models \varphi\},$$

we have

Corollary 9. *VALID is r.e.*

Even better would be if we could actually *decide* validity of FO sentences. However, a negative answer was provided to this question was Church and Turing:

Theorem 10 (**Church-Turing Undecidability Theorem**)**.** *VALID is undecidable.*

In database theory, however, we are not concerned with validity over *all* structures, but rather validity over finite structures (i.e., over all databases).

$$\text{FIN-VALID} \stackrel{\text{def}}{=} \{\varphi \mid \mathfrak{A} \models \varphi \text{ for all finite } \mathfrak{A}\}$$

Unforunately, this language turns out to be undecidable as well:

Theorem 11 (Trakhtenbrot's Theorem). *FIN-VALID is undecidable.*

In fact, FIN-VALID is not even r.e.! This is easy to see since FIN-VALID is clearly co-r.e. (to check whether a sentence φ is *not* in FIN-VALID, it suffices to just enumerate finite structures $\mathfrak{A}$ checking $\mathfrak{A} \models \varphi$ until a counterexample is found).

Definition 12. Relational calculus queries Q, Q' are *equivalent* if for any database instance I, we have $[\![Q]\!]^I = [\![Q']\!]^I$. As a homework exercise, we will prove that equivalence is undecidable as a consequence of results above.

Proof. (of Church-Turing Theorem) By reduction from the String Rewriting Problem (SRP). We know that SRP is undecidable. So if we can show that SRP $\leq$ VALID then we can prove VALID is undecidable as well.

We will describe the total computible function that performs the reduction SRP $\leq$ VALID by the program which computes it. The input to this program is of the form (R, u, v) where R is a finite set of string rewrite rules over an alphabet Σ and u, v are strings over Σ. The reduction produces as output an FO sentence Φ. For this to be a many-one reduction, we need to show that $u \overset{R}{\twoheadrightarrow} v$ iff $\models \Phi$.

First the reduction has to construct the FO vocabulary over which Φ is built. This contains: a binary predicate symbol *Rew*; a unary function symbol f_α for each $\alpha \in \Sigma$; and a single constant symbol c.

Next, for each rewrite rule $r \epsilon R$

$$r : \alpha_1 \cdots \alpha_m \to \beta_1 \cdots \beta_n$$

the reduction will construct a sentence

$$\varphi_r \stackrel{\text{def}}{=} \mathit{Rew}(f_{\alpha_1}(\ldots f_{\alpha_m}(x) \cdots), f_{\beta_1}(\ldots f_{\beta_n}(x) \cdots))$$

and then if $R = \{r_1, \ldots, r_k\}$, the reduction will construct

$$\varphi_R \stackrel{\text{def}}{=} \varphi_{r_1} \wedge \cdots \wedge \varphi_{r_k}.$$

Now, if $u = \delta_1 \cdots \delta_p$ and $v = \epsilon_1 \cdots \epsilon_q$, the reduction will construct

$$\varphi_{u,v} \stackrel{\text{def}}{=} \mathit{Rew}(f_{\delta_1}(\cdots f_{\delta_p}(c) \cdots), f_{\epsilon_1}(\cdots f_{\epsilon_q}(c) \cdots))$$

Next, we need to some add some assertions that specify that *Rew* behaves like a rewrite relation:

$$\varphi_{Rew} \stackrel{\text{def}}{=} (\forall x\, \mathit{Rew}(x,x) \wedge (\forall x, y, z \mathit{Rew}(x,y) \wedge \mathit{Rew}(y,z) \to \mathit{Rew}(x,z)) \wedge$$

$$\bigwedge_{\alpha \epsilon \Sigma} (\forall x, y \mathit{Rew}(x,y) \to \mathit{Rew}(f_\alpha(x), f_\alpha(y)))$$

Finally, the reduction will construct

$$\Phi \stackrel{\text{def}}{=} (\varphi_R \wedge \varphi_{Rew}) \to \varphi_{u,v}$$

It remains to show that $u \overset{R}{\twoheadrightarrow} v$ iff $\models \Phi$. For proof of this claim, see the Friendly Logics notes.

□

4 Monday, 1/11/10 (scribe: TJ)

See the "Friendly Logics" notes, Section 4.

5 Wednesday, 1/13/10 (scribe: TJ)

See the "Friendly Logics" notes, Section 5.

6 Friday, 1/15/10 (scribe: TJ)

See the "Friendly Logics" notes, Section 6.

7 Wednesday, 1/20/10 (scribe: Vu)

Domain Independence

Recall that a *relational calculus query* (or FO query) has the form

$$q \equiv \{\bar{x} \mid \varphi(\bar{x})\}$$

The output of relational calculus queries might be infinite. For example, the output of relational calculus query

$$\{x \mid \neg R(x)\}$$

is infinite since

$$[\![\{x \mid \neg R(x)\}]\!]^{\mathcal{I}} = \mathbb{D} - R^{\mathcal{I}}.$$

Another infinite-output query is $\{(x, y)|R(x) \vee S(y)\}$.

More subtly, the output of query

$$\{x|\forall y R(x, y)\}$$

is finite, but *depends on the contents of* $\mathbb{D}$ (and not just the contents of the predicates in the database instance).

All the queries above are *domain-dependent queries.* Domain-dependence is generally considered an undesirable property of query languages.

Let $\mathcal{I}$ is a structure with domain $\mathbb{D}$. The *active domain* of database instance $\mathcal{I}$ is the set $adom(\mathcal{I})$ of all values from $\mathbb{D}$ occurring in an interpretation of some predicate $P^{\mathcal{I}}$. We have $adom(\mathcal{I}) \subseteq \mathbb{D}$. Likewise if Q is a relational calculus query, then $adom(Q)$ is the finite set of constants occurring in Q.

The restriction of a database instance $\mathcal{I}$ to the universe $D \supseteq adom(I)$ is

$$\mathcal{I}/D \stackrel{\text{def}}{=} \langle D, \{P_i^{\mathcal{I}/D}\}, \emptyset, \{c\}\rangle$$

where $P_i^{\mathcal{I}/D} = P_i^D$.

A relational calculus query Q is *domain-independent* if for any database instance $\mathcal{I}$ and any D_1, D_2 such that

$$adom(\mathcal{I}) \cup adom(Q) \subseteq D_1 \subseteq D_2 \subseteq \mathbb{D}$$

we have

$$[\![Q]\!]^{\mathcal{I}/D_1} = [\![Q]\!]^{\mathcal{I}/D_2} \big(= [\![Q]\!]^{\mathcal{I}} = [\![Q]\!]^{adom(\mathcal{I}) \cup adom(Q)}\big)$$

Theorem 13. *Domain-Independence of a relational calculus query is undecidable.*

Proof. Reduce FIN-VALID to Domain-Independence.

$$\varphi \longmapsto q \equiv \{x|\neg\varphi \wedge \neg R(x)\}$$

where R does not occur in φ.

Suppose $\varphi \in$ FIN-VALID (and thus $\neg\varphi$ is unsatisfiable) then

$$[\![Q]\!]^{\mathcal{I}} = \{\}, \text{ for any } \mathcal{I}$$

Hence, Q is domain-dependent.
Suppose $\varphi \notin$ FIN-VALID then

$$\exists \mathcal{A} \text{ s.t. } \mathcal{A} \models \neg\varphi$$

Construct database instance $\mathcal{I}$ which in $\mathcal{A}$ along with interpretation of R

$$R^{\mathcal{I}} = \{\}$$

Then $[\![Q]\!]^{\mathcal{I}} = \mathbb{D}$.
Moreover, for any $\mathbb{D}$ s.t. $adom(\mathcal{I}) \subseteq D \subseteq \mathbb{D}$ we have $[Q]\!]^{\mathcal{I}/D} = \mathbb{D}$.
Thus Q is not domain-independent. □

Relational Algebra

The *relational algebra* has operators $P, \{a\}, \sigma, \pi, \times, -$. The semantics of a relational algebra expression E is defined inductively as follows.

- **Predicate** P: $[\![P]\!]^{\mathcal{I}} = P^{\mathcal{I}}$
- **Constant** $\{c\}$: $[\![\{c\}]\!]^{\mathcal{I}} = \{c\}$
- **Selection-1** $\sigma_{i=j}E : [\![\sigma_{i=j}E]\!]^{\mathcal{I}} = \{\bar{x} \mid \bar{x} \in [\![E]\!]^{\mathcal{I}} \text{ and } x_i = x_j\}$
- **Selection-2** $\sigma_{i=c}E : [\![\sigma_{i=c}E]\!]^{\mathcal{I}} = \{\bar{x} \,|]\, \bar{x} \in [\![E]\!]^{\mathcal{I}} \text{ and } x_i = c\}$
- **Projection** $\pi_{i_1,\ldots,i_k}E : [\![\pi_{i_1,\ldots,i_k}E]\!]^{\mathcal{I}} = \{(x_{i_1}, ..., x_{i_k}) \mid \bar{x} \in [\![E]\!]^{\mathcal{I}}\}$
- **Cross Product** $E_1 \times E_2 : [\![E_1 \times E_2]\!]^{\mathcal{I}} = \{\bar{x}, \bar{y} \mid \bar{x} \in [\![E_1]\!]^{\mathcal{I}} \text{ and } \bar{y} \in [\![E_2]\!]^{\mathcal{I}}\}$
- **Union** $E_1 \cup E_2 : [\![E_1 \cup E_2]\!]^{\mathcal{I}} = \{\bar{x} \mid \bar{x} \in [\![E_1]\!]^{\mathcal{I}} \text{ or } \bar{x} \in [\![E_2]\!]^{\mathcal{I}}\}$
- **Difference** $E_1 - E_2 : [\![E_1 - E_2]\!]^{\mathcal{I}} = \{\bar{x} \mid \bar{x} \in [\![E_1]\!]^{\mathcal{I}} \text{ and } \bar{x} \notin [\![E_2]\!]^{\mathcal{I}}\}$

Note that intersection is not included in the list above as it can be defined using cross product, selection, and projection.

8 Friday, 1/22/10 (scribe: Zhongxian)

Paper Presentation

Students are going to present papers from the following four directions at the last two weeks.

- Data exchange (2) [Zhongxian, Vu]
- Bag semantics [Sarah]
- Probabilistic/incomplete databases (2) [Thanh, Mingming]
- Data provenance [Sven]

Data exchange

Assume we have a source database schema and a target database schema. How can we exchange data from the source to the target. One approach is to use schema mappings(logical mappings).

Bag semantics

Bag semantics deal with the problem of duplicate rows generated by queries.

Probabilistic/incomplete database

Databases with not sure of null values in rows.

Data provenance

Data provenance studies unified formalism of data representation.

Relational algebra query captures domain independent FO query

We'd like to prove that :domain independent $FO = RA$.

Theorem 14. *For any E(expression $\in RA$, we can compute an equivalent domain independent FO query.*

proof (via structural reduction)

- case: $P \longmapsto \{\bar{x}|P(\bar{x})\}, \bar{x}$ fresh
- case: $c \longmapsto \{x|x = c\}, x$ fresh
- case: Assuming $E \longmapsto \{\bar{x}|\varphi(\bar{x})\}, \sigma_{i=j}(E) \longmapsto \{\bar{x}|\varphi(\bar{x}) \wedge x_i = x_j\}$
- case: Assuming $E_1 \longmapsto \{\bar{x}|\varphi(\bar{x})\}, E_2 \longmapsto \{\bar{y}|\psi(\bar{y})\}, E_1 \times E_2 \longmapsto \{\bar{x}, \bar{y}|\varphi(\bar{x}) \wedge \psi(\bar{y})\}$
- case: Assuming $E \longmapsto \{\bar{x}|\varphi(\bar{x})\}, \pi_{i_1,..,i_k}(E) \longmapsto \{y_1, ..., y_k|\varphi(\bar{x}) \wedge y_1 = x_1 \wedge ... \wedge y_k = x_k\}$
- case: Assuming $E_1 \longmapsto \{\bar{x}|\varphi(\bar{x})\}, E_2 \longmapsto \{\bar{y}|\psi(\bar{y})\}. E_1 \cup E_2 \longmapsto \{\bar{x}|\varphi(\bar{x}) \vee \psi(\bar{x})\}$
- case: Assuming $E_1 \longmapsto \{\bar{x}|\varphi(\bar{x})\}, E_2 \longmapsto \{\bar{y}|\psi(\bar{y})\}. E_1 - E_2 \longmapsto \{\bar{x}|\varphi(\bar{x}) \wedge \neg\psi(\bar{x})\}$

Now, we prove the other direction: Domain independent FO query can be expressed using relational algebra. We use two steps to prove that.

Step 1: For any FO query Q, can compute a RA query E over schema σ of Q extended with a unary predicate D, such that for any database instance I, we have $[\![Q]\!]^{I/D^I} = [\![E]\!]^I$.

Step 2: Recall Q is domain independent, then $[\![Q]\!]^I = [\![Q]\!]^{I/adom(I) \cup adom(Q)}$. Plugin $adom(I) \cup adom(Q) = D^I$. Observe that $adom(I) \cup adom(Q)$ can be computed using a RA query E_{ad}.

Say $\sigma = \{R(,), S(,,)\}$ and $adom(Q)$ has constants a,b. Then define

$E_a d \stackrel{def}{=} \pi_1(R) \cup \pi_2(R) \cup pi_1(S) \cup pi_2(S) \cup pi_3(S) \cup \{a\} \cup \{b\}$.

proof (step 1, by induction on query)

- case: $\{x_1, ..., x_n | x_i = c\} \longmapsto \sigma_{i=c}(D \times D \times ... \times D)(ntimes)$
- case: $\{x_1, ..., x_n | \neg\varphi(x_1, ..., x_n)\} \longmapsto D^n - E$
- case: Assuming $\{\bar{x}|\varphi(\bar{x})\} \longmapsto E_1, \{\bar{x}|\psi(\bar{x})\} \longmapsto E_2, \{\bar{x}|\varphi(\bar{x}) \wedge \psi(\bar{x})\} \longmapsto E_1 \wedge E_2$
- case: Assuming without losing generosity, all variables are distinct, $\{x_1, ..., x_n | P(x_1, ..., x_n)\} \longmapsto P$
- case: Assuming $\{x_1, ..., x_m, z, y_1, ..., y_n | \varphi(...)\} \longmapsto E, \{x_1, ..., x_m, y_1, ..., y_n | \exists z, \varphi(x_1, ...x_m, z, y_1, ...y_n)\} \longmapsto \pi_{1,...,m,m+2,..,m+n+1}(E)$

Theorem 15. *Equivalence of RA query is undecidable. So do satisfiability and validity of RA query.*

9 Monday, 1/25/10 (scribe: Mingmin)

Model-checking problem

Given ψ *and* $\mathfrak{A}$, to check if $\mathfrak{A} \models \psi$

Definition 16. (recognition problem) Given an FO query Q, database instance I, and an output tuple $\overline{a}$, is $\overline{a} \in [\![Q]\!]^I$?

Theorem 17. *If Q is domain-independent, then the recognition problem is LOGSPACE (data complexity) PSPACE-complete (combined complexity). Then we no longer have to encode size of universe in representation of I.*

Conjunctive queries

Essentially, FO queries of the form

$$\{\overline{x} \mid \exists \overline{y}\ \psi(\overline{x}, \overline{y})\}$$

where ψ is a conjunction of atoms(relational or equality).

↙ ↘

predicate, P(x,y) x=y or x=c

Example 18. $\{(x, z) \mid \exists y R(x, y) \wedge z = y \wedge S(y, x))\}$
$\{(x, y) \mid R(x)\}$ *or* $\{(x, y) \mid \exists z\ R(x) \wedge y = z)\}$ → needs to be ruled out(via range-restriction).

Conjunctive queries in SQL

SELECT-FROM-WHERE fragment of SQL

select R1,S2
from R,S
where $\underbrace{\text{R1=c and S1=R1 and R3=S2}}_{\text{conjunction of equalities.}}$

Definition 19. A *tableau* T is a set of atoms (rel. or eq.) that is *range-restricted* i.e. for any variable x∈vars(T), either T⊢x=c or T⊢x=x′ where x′ occurs in a relational atom in T.

$$eg \quad \{R(y), y = z, z = x\} \vdash\ y = x$$

Definition 20. A *conjunctive query* is given by a pair $\langle \overline{u}, \mathrm{T} \rangle$ where $\overline{u}$ is a tuple (the "output" tuple), T is a tableau, and vars($\overline{u}$)$\subseteq$vars(T).
T=$\{A_1 \dots A_n\}$, $\{\overline{u}$ — $A_1 \wedge \dots \wedge A_n\}$.

Rule-based, Prolog-style syntax for CQs. (Datalog-style)

$$Q(x,y) : -R(x,z), x = c, S(x,y,z) \quad \longrightarrow \quad Q(c,y) : -R(c,z), S(c,y,z)$$

$\downarrow$ $\downarrow$

output tuple or <u>*head*</u> *tableau or* <u>*body*</u>

This can *always* be done, provided the query is *satisfiable*, i.e. it is not the case that T$\vdash c_1 = c_2$ for distict constants c_1, c_2.

Can define sem. of CQs via translation to FO queries. But can define (equivalent) semantics directly via notion of valuations.

Definition 21. Given tableau T, d.b. instance I, a *valuation* for T in I is a mapping β: vars(T) $\rightarrow \mathbb{D}$, extended to map constants to themselves. We say that β satisfies T in I if

* for any atom R($\overline{x}$) in T, $\beta(\overline{x}) \in R^I$

* for any atom e=e′ in T, we have $\beta(e) = \beta(e')$

Proposition 22. *For any CQ Q=$\langle \overline{u}, T \rangle$, and d.b. instance I.*

$$[\![Q]\!]^I = \{\beta(\overline{u}) \mid valuateion\ \beta\ satisfies\ T\}$$

Q is viewed as FO query.

10 Wednesday, 1/27/10 (scribe: Thanh)

Property 23. A conjuntive querry $Q =< \bar{u}, T >$ is satisfiable iff for all c_i, c_j if $T \vdash c_i = c_j$ then $c_i = c_j$

We give a sketch proof by an example.

Example 24. We prove the querry $< (x,c), \{R(x,c), S(x,y), S(y,z)\} >$ is satisfiable.
Database instance I:

R	
c_x	c

S	
c_x	c_y
c_y	c_z

then valuate $(c_x, c) \in [\![Q]\!]^I$.

Theorem 25. *The expression (and combined) complexity of recognition problem for CQs is NP-complete. (Recall for FO, PSPACE-complete).*

<u>Proof:</u> We have two staments to show:
1. Combined complexity in NP: use as witness satisfying valuation $\beta : vars(Q) \rightarrow \mathbb{D}$.
2. Expression complexity is NP-hard: we use reduction from 3-coloring problem.

<u>Useful fact of Graph theory:</u>
A directed graph $G = (V, E)$ is k-colorable iff there exists a graph homomorphism $h : G \rightarrow C_k$ where C_k is the complete graph on k vertices.
Given $G = (V, E)$ and $G' = (V', E')$. A mapping is called a graph homomorphism if for all $u, v \in V$

$$(u,v) \in E \Rightarrow (h(u), h(v)) \in E'$$

Proof: Given $G = (V, E)$. Construct CQ $Q =< (), \{P(u,v) | (u,v) \in E\} >$ and a database instance I defined:

$$P^I = \{(r,g), (g,r), (b,r), (g,b), (b,g)\}$$

Claim: $() \in [\![Q]\!]^I$ iff G is 3-colorable (iff exists graph homomorphism $h : G \to C_3$).
SPC () in input
Then have set valuation $\beta : vars(Q) \to \mathbb{D}$.

Theorem 26. *CQs and SPC queries are expressively equivalent.*

Proof: (Sketch)
$CQs \to SPC$
Illustrate by example:

$$Q =< (x,d), \{R(x,y), S(y,c,z)\} >$$
Normal form: $\pi_{2,1}(\{d\} \times \sigma_{4=c}(\sigma_{2=3}(R \times S)))$

$SPC \to CQs$
Using HW2 and HW3 can always put SPC querry in normal form the "read" as CQ.

Definition 27. A querry Q is contained in another querry Q', written $Q \sqsubseteq Q'$ if for any database instance I, we have $[\![Q]\!]^I \subseteq [\![Q']\!]^I$. Moreover, Q and Q' are equivalent, written $Q \equiv Q'$ if

$$\forall I \; [\![Q]\!]^I = [\![Q']\!]^I$$

(Note that $Q \equiv Q'$ if $Q \sqsubseteq Q'$ and $Q' \sqsubseteq Q$).
We'll show for CQs, $Q \sqsubseteq Q'$ is decidable (and NP-complete).

Definition 28. Given CQs $Q =< \bar{u}, T >$ and $Q' =< \bar{v}, T' >$, a containment mapping $h : Q' \to Q$ is a mapping $h : vars(T') \to \mathbb{D}$, extended to map any constant to itself and such that:
* $h(\bar{V}) = \bar{u}$
* for any relational atom $P(\bar{e})$ in T', we have $P(h(\bar{e})) \in T$.

Example 29.
$$\begin{aligned} &Q =< (x,y), R(x,y,c), R(x,z,c) > \\ &Q' =< (u,v), R(u,v,c) > \\ &h : Q' \to Q : \quad u \to x \\ &\qquad\qquad\quad\; v \to y \\ &\qquad\qquad\quad\; c \to c \end{aligned}$$

Lemma 30. *1. The identity mapping is a containment mapping id $Q \to Q$*
2. Containment mappings compose:

$$h : Q \to Q', \; g : Q' \to Q''$$

$$g \circ h : Q \to Q''$$

Definition 31. If $Q =< \bar{u}, T >$ is a CQ, the canonical database denoted can(Q) for Q is the database instance obtained b viewing T as a database instance.

Example 32. $T = \{R(x,y), S(y,z,c)\}$
can(Q) =

R	S
x y	y z c

"frozen" variables

Theorem 33 (Chandra and Merlin, 1977)**.** *For CQs Q,Q' the following statements are equivalent:*
1. $Q \sqsubseteq Q'$
2. *There is a containment mapping* $h : Q' \to Q$
3. $[\![Q]\!]^{can(Q)} \subseteq [\![Q']\!]^{can(Q)}$

11 Friday, 1/29/10 (scribe: Sarah)

proof: of Chandra and Merlin $(1) \Rightarrow (3)$: is self-evident.
$(3) \Rightarrow (2)$: suppose $\bar{u} \in [\![Q']\!]^{can(Q)}$, then $\exists$ a satisfying valuation $\beta : vars(Q') \to \mathbb{D}$, ie $\beta(\bar{v}) = \bar{u}$. For every atom $P(\bar{v}) \in T'$ we have $\beta(\bar{v}) \in P^{can(Q)}$. Now we can view β as a containment mapping $\beta : Q' \to Q$.
$(2) \Rightarrow (2)$: suppose we have a containment mapping $h : Q \to Q'$. Consier an arbitrary I and $\bar{a}$. If $\bar{a} \in [\![Q]\!]^I$ then we have a satisfying valuation $\beta : vars(Q) \to \mathbb{D}$, so $\beta \circ h$ is also a satisfying valuation, $\beta \circ h : vars(Q) \to \mathbb{D}$. $\therefore h(\bar{v}) = \bar{u}, \beta(\bar{u}) = \bar{a}, \beta \circ h(\bar{v}) = \bar{a}$
$\therefore P(\bar{v}) \in T', P(h(\bar{v})) \in T, \beta(h(barv)) \in P^I$.
corollary: testing $Q \sqsubseteq Q'$ is NP-complete. proof: (sketch) Guessing a containment mapping h is NP, and hard because it is essentially a recognition problem, ie is $\bar{a} \in [\![Q]\!]^I$? let $I \mapsto T$, so consider I a "frozen query." let $\bar{a} \mapsto \bar{u}$ where $Q' =< \bar{u}, T >$. $Q' \sqsubseteq Q$ iff $\bar{u} \in [\![Q]\!]^{can(Q')}$ and iff $\bar{a} \in [\![Q]\!]^I$

Query Minimization

Example 34. $Q(x.y) : -R(x,y), R(x,z) \downarrow Q'(x,y) : -R(x,y)$

Definition 35. Given a conjunctive query $Q =< \bar{u}, T >$, a subquery is a conjunctive query of the form: $Q_S =< \bar{u}, S >$ where $S \subseteq T$.

note: if Q_S is a subquery of Q, then $Q \sqsubseteq Q_S$ because we can define $h : Q_S \to Q$ as a containment mapping. This seems backwards, so here's an example:

Example 36. $Q(x,y) : -R(x,z), R(z,u), S(z,y) \downarrow Q_S(x,y) : -R(x,z), S(z,y)$
note the original query is more restrictive and wants a more specific part of the database. It is therefore contained by the less restrictive subquery.

We are interested in cases where $Q_S \sqsubseteq Q$ because we can use this to infer $Q \equiv Q_S$.

Definition 37. A locally minimal conjunctive query $Q =< \bar{u}, T >$ has no strict subquery Q_S such that $Q_S \equiv Q$.

Lemma 38. *Let h be a containment mapping from $< \bar{u}, T >$ to $< \bar{v}, T' >$. Then:*

- *if h is injective (ie one-to-one) on variables, then it is injective on atoms.*
- *if h is surjective (ie onto) on atoms, then it is surjective on variables.*

Note that the converses of these statements need not hold.

Definition 39. An isomorphism is a containment mapping that is bijective on variables and atoms. (ie, a renaming of variables)

Lemma 40. *A conjunctive query Q is locally minimal iff any containment mapping h from Q to itself is an isomorphism.*

proof: (of converse) Consider $Q_S =< \bar{u}, S >$ a subquery of $Q =< \bar{u}, T >$ such that $Q_S \equiv Q$. Since $Q_S \sqsubseteq Q$, we have a containment mapping $h : Q \to Q_S$. We can also write $h : Q \to Q$ since Q and Q_S are equivalent, so $T = H(T)$. Now $T \subseteq S$ because of containment, and $S \subseteq T$ because of the subset, so $S = T$ and $Q = Q_S$, hence Q is locally minimal. (of lemma) Let $h : Q \to Q_S$ be a containment mapping, and $Q' =< \bar{u}, h(T) >$ be a conjunctive query, where Q' is a subquery of Q. Because of h we know $Q' \sqsubseteq Q$, and $Q' \equiv Q$. Since Q is locally minimal, $Q = Q'$. So h must be a surjection $T \to T$ and $var(T) \to var(T)$. Since T and $vars(T)$ are finite, a surjection from T to itself or $vars(T)$ to itself is also injective, ie an isomorphism $\therefore h$ is also an isomorphism.

12 Monday, 2/1/10 (scribe: Daniel)

Proposition 41. *If a CQ Q is locally minimal then any containment mapping $h : Q \to Q$ is an isomorphism.*

Theorem 42. *Consider a CQ Q. Any locally minimal CQ Q' equivalent to Q is isomorphic to some subquery of Q.*

Proof. Let $Q =< \bar{u}, T >$, $Q' =< \bar{u}, T' >$. Since $Q \equiv Q'$, we have containment mappings $h : Q \to Q'$ and $h : Q' \to Q$. Moreover, $h \circ g : Q' \to Q'$ is a containment mapping, in fact by Prop. 41, it is an isomorphism. Since $h \circ g$ is an isomorphism, g has to be injective (on variables and atoms).
Now, consider $Q_s =< \bar{u}, g(T') >$, a subquery of Q. We claim, g is an isomorphism from Q' to Q'. This is true, since (1) g is a containment mapping, (2) g is injective on atoms and variables, (3) g is surjective on atoms by construction, and (4) due to Lemma 38, also surjective. □

Corollary 43. *If two CQs Q and Q' are locally minimal and equivalent, then they are isomorphic.*

Proof. By theorem 42, Q is isomorphic to some subquery of Q'; but Q' is locally minimal and thus this subquery has to be Q', thus Q isomorphic to Q'. □

Optimization Procedure:
Given CQ $Q =< \bar{u}, T >$
Set $Q_s := Q$
While exists a containment mapping from Q_s to a strict subquery Q'_s of Q_s
Set $Q_s := Q'_s$
Return Q_s.

Definition 44. A CQ is said to be *globally minimal* if it has the smallest number of atoms of any CQ equivalent to it.

Proposition 45. *A CQ is locally minimal iff it is globally minimal.*

Example 46 (CQ Minimization)**.**

$$Q(x, y, z) : -R(x, y_2, z_2), R(x_1, y, z_1), R(x_2, y_2, z), R(x, y_1, z_1), R(x_2, y_1, z)$$

We guess that a good containment mapping $h : Q \to Q$ would be one that changes $z_2 \mapsto z_1$ and $y_2 \mapsto y_1$. With this, Q' would be:

$$Q'(x, y, z) : -R(x, y_1, z_1), R(x_1, y, z_1), R(x_2, y_1, z), R(x, y_1, z_1), R(x_2, y_1, z)$$

The last two atoms are redundant as they already occur in Q'. Also, each of the atoms in Q' is present in Q, so h actually was a containment mapping. Without the redundant atoms we get:

$$Q''(x, y, z) : -R(x, y_1, z_1), R(x_1, y, z_1), R(x_2, y_1, z)$$

Through exhaustive search, we can verify that this query is actually minimal.

For a query Q, a minimal query Q' is also called the *core* of Q. This terminology comes from graph theory.

Integrity Constraints

A commonly used constraint is a *key constraint*: Given a relational schema $R(A, B, C)$, we say "A is a key for R", if the value of A identifies the tuple, i.e., if there is a functional dependency $A \to B, C$. Keys can also contain two or more attributes: $A, B \to C$ or "A, B is a superkey (or compound key) of R".
A further constraint type is a *foreign key constraint*: Given $R(A, B, C)$ and $S(C, D)$ we say "C is a foreign key in S referencing R" if for every tuple $(a, b, c) \in R$, there exists a tuple $(c, d) \in S$ for some d.

We will see that under constraints, non-equivalent queries can become equivalent. This will lead to further optimization opportunities. Consider $Q(x,y,z) : -R(x,y,z)$ and $Q'(x,y,z) : -R(x,y,z), S(z,u)$. These are clearly not equivalent; however under the constraint that the first column in S is a foreign key referencing the third column in R, the queries are equivalent.
Dependencies in general, are FO logical assertions, i.e., sentences of various forms.

Definition 47. An *embedded dependency (ed)* is a FO sentence of the form

$$\forall \bar{x} \varphi(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y}),$$

where φ is a conjunction of relational atoms, and ψ is a conjunction of relational atoms or equality atoms.

Example 48.
"A is key in $R(A,B,C)$" would be $\forall x,y,z,y',z' \, R(x,y,z) \wedge R(x,y',z') \rightarrow y = y' \wedge z = z'$.
"C is a foreign key in $S(C,D)$ referencing $R(A,B,C)$" would be $\forall x,y,z \, R(x,y,z) \rightarrow \exists u \, S(z,u)$.

Definition 49. A *conjunctive containment dependency (ccd)* is an assertion of the form:

$$Q \sqsubseteq Q',$$

where Q, Q' are conjunctive queries.

Proposition 50. *Embedded dependencies and conjunctive containment dependencies are equally expressive.*

Proof. **Sketch.** (1) Given an ed $d = \forall \bar{x} \varphi(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y})$. Let $Q =< \bar{x}, \varphi >$ (where φ is viewed as a tableau, i.e., as a set of atoms). Let further $Q' =< \bar{x}, \varphi \cup \psi >$ (again, with φ and ψ viewed as sets of atoms). We now claim that for any database instance I: $I \models d$ iff $[\![Q]\!]^I \subseteq [\![Q']\!]^I$.
(2) Given a ccd $Q \sqsubseteq Q'$ with $Q =< \bar{u}, T >$ nd $Q' =< \bar{u}, T' >$. Let $\mathrm{cont}(Q,Q')$ be the embedded dependency. Further, let $\bar{z}$ be the variables in T but not in $\bar{u}$ and let $\bar{z}'$ be the variables in T' but not in $\bar{u}$. Now,

$$\mathrm{cont}(Q,Q') \stackrel{\mathrm{def}}{=} \forall \bar{u} \forall \bar{z} \, T \; \rightarrow \exists \bar{u}' \exists \bar{z}' \, T' \wedge \bar{u} = \bar{u}'$$

where T and T' are viewed as conjunctions of atoms. We now claim that for any database instance I: $[\![Q]\!]^I \subseteq [\![Q']\!]^I$ iff $I \models \mathrm{cont}(Q,Q')$. □

Corollary 51. *Testing whether an ed or a ccd holds in all instances is NP complete.*

13 Wednesday, 2/3/10 (scribe: Mingmin)

Example 52. Consider a CQ

$$Q(y,z) : - \; R(x,y,z'), R(x,y',z)$$

As is, Q is minimal. But suppose we assume the (key) dependency

$$\forall x,y,z,y',z' R(x,y,z) \wedge R(x,y',z') \rightarrow y = y' \wedge z = z'$$

Then, assuming dependency holds, Q is equivalent to

$$Q'(y,z) : - \; R(x,y,z) \; .$$

Example 53. Consider a CQ

$$Q(x,y) : - \; R(x,y), S(x,z)$$

As is, Q is minimal. But if we assume the (foreign key) dependency

$$\forall x,y R(x,y) \rightarrow \exists z S(x,z)$$

Then, we can minimize Q to

$$Q'(x,y) : - \; R(x,y) \; .$$

Main Technique: the **chase**

Example 54. Consider a CQ (in relational calculus notation)

$$Q = \{\bar{x} \mid \exists \bar{y}\psi(\bar{x}, \bar{y})\}$$

and an e.d.

$$d = \forall \bar{x}, \bar{y}\psi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\varphi(\bar{x}, \bar{y}, \bar{z})$$

then a **chase step** from Q with d, written $Q \xrightarrow{d} Q'$, produces CQ

$$Q' = \{\bar{x} \mid \exists \bar{y}\exists \bar{z}\psi(\bar{x}, \bar{y}) \vee \varphi(\bar{x}, \bar{y}, \bar{z})\}$$

Definition 55. A **homomorhisim of tableaux** is a mapping h:$T \rightarrow T'$ s.t. if atom A$\in$ T, then h(A)$\in$T$'$. R(x,y)$\Rightarrow$R(h(x),h(y))$\in$T$'$, i.e. they are just c.m.s, minus the requirement to map output tuple to output tuple.

Definition 56. Consider the e.e. d $\overset{def}{=}$ $\forall \bar{x}\psi(\bar{x}) \rightarrow \exists \bar{y}\varphi\bar{x}, \bar{y}$ and a tableau T, we say that the **chase** is *applicable* to T if there exists a homomorphism h:$\psi \rightarrow T$ that cannot be extended to map $\psi \cup \varphi \rightarrow T$, i.e. there does not exist a homomorphism h$'$: $\psi \cup \varphi \rightarrow T$, where h$'$=h on ψ where the chase is applicable the *result* of one step of chase of T with d is the tableau T$'$ $\overset{def}{=}$T$\cup\psi[\bar{x} \longmapsto h(\bar{x})]$.

Example 57. $d \overset{def}{=} \forall x, yR(x, y) \rightarrow \exists zS(x, z)$.
T={R(u,v)}
T$\xrightarrow{d}$T$'$ where T$'$ ={R(u,v),S(u,z))}   S(x,z)[x$\longmapsto$u]=S(u,z)
Note: In the definition, we assume vars in d and T are disjunct (always can be accomplished by renaming).

Definition 58. **chase** on CQs is defined in terms of their underlying tableau
i.e. if T$\rightarrow$T$'$ and Q=$<\bar{u}$,T>, then Q$\xrightarrow{d}$Q$'$ where Q$'$=$<\bar{u}, T'>$.

Lemma 59. *If the chase with d is not applicable to a (satisfiable) tableau T, then Inst(T)$\models$d.*
Proof. *exercise.*

Lemma 60. *If* $Q\xrightarrow{d}Q'$, *then* $d\models Q\equiv Q'$

Proof. $\models$Q$'$ $\sqsubseteq$Q easy since T$\subseteq$T$'$, so identity is a containment mapping Q$\rightarrow$Q$'$.
Now, wts. d$\models$Q$\sqsubseteq$Q$'$, i.e. for any DB instance I s.t. d$\models$T, we have $[\![Q]\!]^I \subseteq [\![Q']\!]^I$.
Let d=$\forall \bar{x}\psi(\bar{x}) \rightarrow \exists \bar{y}\varphi(\bar{x}, \bar{y})$. Let h:$\psi$ $\rightarrow$T be the hom. used in chase step. Hence T$'$=T$\cup\psi[\bar{x} \longmapsto h(\bar{x})]$.
Since the output tuple is the same in Q and Q$'$, it suffices to show that any sat. val. for T in I can be extended to a sat. val. for T$'$.
Let β:T$\rightarrow$I be such a sat. val. It follows that $\beta \circ h$: ψ $\rightarrow$I satisfies ψ in I. And since d$\models$I, we can extend $\beta \circ h$ to a sat. val. $\gamma : \psi \cup \varphi$ $\rightarrow$I. Now, define β':T$'$ $\rightarrow$I as follows:

$$\beta'(z) = \begin{cases} \gamma(z) & if\ z \in vars(\bar{y}) \\ \beta(z) & otherwise \end{cases}$$

So β' extends β, remains to show that β' satisfies T$'$ in I.
Clearly, β' satisfies the atoms that are also in T.
Now,suppose e.g. that R(h(x),y), where x$\in \bar{x}$ and y $\in \bar{y}$, is an atom in T$' \rightarrow$T, e.e., in $\varphi[\bar{x} \longmapsto h(\bar{x})]$.
We have $(\beta'(h(x)), \beta'(y)) = (\beta(h(x)), \gamma(y)) = (\gamma(x), \gamma(y)) \in R^I$, since γ is a sat. val. for φ in I.
Similarly for eq. atoms. ☐

Definition 61. Let Q be a CQ and D a set of e.d.s.
A **terminating chase sequence** of Q with D is a sequence of chase steps:

$$Q \stackrel{d_1}{\rightarrow} Q_1 \stackrel{d_2}{\rightarrow} \ldots \stackrel{d_n}{\rightarrow} Q_n$$

where $d_1, \ldots, d_n \in$ D and no chase with deps from D is applicable to Q$_n$.

Theorem 62. *Let Q, Q$'$ $\in$CQ and D is a set of e.d.s. Suppose exists a term chase sequence of Q with D producing Q_n. Then*

$$D \models Q \sqsubseteq Q' \; iff \; Q_n \sqsubseteq Q'$$

14 Friday, 2/05/10 (scribe: Zhongxian)

Theorem 63. *Let Q, Q$'$ be CQs and D a set of eds. Suppose Q_n is the result of a terminating chase sequence from Q with D. Then:*

$$D \models Q \sqsubseteq Q' \quad \textit{iff} \quad Q_n \sqsubseteq Q'$$

Proof. **Sketch.** "$\Leftarrow$" is immediate, since $D \models Q \equiv Q_n$. (Lemma 59)
"$\Rightarrow$": Suppose $D \models Q \sqsubseteq Q'$, let T_n be the tableau underlying Q_n.
Suppose Q_n is unsatisfiable, then $Q_n \sqsubseteq X$ or CQ X in particular $Q_n \sqsubseteq Q'$.
Assume $Q =< \bar{u}, T >, Q' =< \bar{u'}, T' >, Q_n =< \bar{u_n}, T_n >$. Now suppose Q_n is satisfiable. Assuming Q, Q', Q_n do not contain equality atoms. Let $I_n = Inst(T_n)$. According to Lemma 58, we have $\bar{u} \in [\![Q]\!]^{I_n}$, also $I_n \models D$, Since $D \models Q \sqsubseteq Q', \bar{u} \in [\![Q']\!]^{I_n}$. Then there exists a satisfiable valuation $\beta : Q' \rightarrow dom(I_n)$, such that $\beta(\bar{u'}) = \bar{u}$. It's easy to see that this yields the required containment mapping $h : Q' \rightarrow Q_n$. ☐

Essential Question: When does the Chase terminate ?
Example: $d_1 \stackrel{def}{=} \forall x, y, R(x, y) \rightarrow \exists z S(z, y)$
$d_2 \stackrel{def}{=} \forall x, y, S(x, y) \rightarrow \exists z R(z, y)$.
Consider a tableau $T = \{R(u_0, u_1)\}$.

Chase Step	yields
d_1	$S(u_2, u_1)$
d_2	$R(u_1, u_3)$
d_1	$S(u_3, u_4)$
d_2	$R(u_3, u_5)$
...	...

There is just one chase sequence (up to variable renaming, and it is infinite.
$d_3 \stackrel{def}{=} \forall x, y, R(x, y) \rightarrow x = y$
$d_4 \stackrel{def}{=} \forall x, y, S(x, y) \rightarrow x = y$
So, we have $R\{u_0, u_1\}S\{\} \stackrel{d_3}{\rightarrow} R\{u_0, u_0\}S\{\} \stackrel{d_1}{\rightarrow} R\{u_0, u_0\}S\{u_1, u_0\} \stackrel{d_4}{\rightarrow} R\{u_0, u_0\}S\{u_0, u_0\}$. For given D and T, we can have finite and infinite chases sequences.

Theorem 64. *(Dentsch, Nash, Remel, 2008) Termination of the chase is undecidable.*

Theorem 65. *$D \models Q \sqsubseteq Q'$ is undecidable.*

Proof. **Sketch.** $Q_E \sqsubseteq Q'$ is *ccd*, hence an *ed*. So $D \models d$ is undecidable. □

Several variants of the (classical) chase proposed, to make it "more deterministic".
Culmilating in the core Chase(PNR 2008).
Idea: at each step of the classical chase, several (but finitely many) dependences apply, each with several (but finitely many) associated tableau homomophisms. At each step of the core chases, do two things:

- "Fire" all applicable dependences in parallel, produce T'.
- Minimize T' by computing its core.

Theorem 66.

- *Preserve equivalence with respect to D.*
- *Deterministic(up to renaming variables).*
- *If there is any terminating classical chase sequence from T with D, then the core chase with T and D terminates.*

15 Monday, 2/8/10 (scribe: Vu)

Definition 67. Given D, focus on tuple-generating dependencies in D (t.g.d.s).

$$\forall \bar{x} \varphi(\bar{x}) \implies \exists \bar{y} \psi(\bar{x}, \bar{y})$$

where ψ contains only predicate atoms (no equality) (contrast to equality-generating dependencies (e.g.d.s)).

$$\forall \bar{x} \varphi(\bar{x}) \implies x_i = x_j$$

Fact: any e.d. can be expressed using just t.g.d.s and e.g.d.s.

Definition 68. Given set D of t.g.d.s, build the chase-flow graph for D as follows:

- vertices: for each predicate symbol of arity k in schema, G has a vertex (P, i) for each $1 \leq i \leq k$.
- for each t.g.d. and each variable x that occurs in position i in an R-atom of the premise of the t.g.d. and in position j in an S-atom in the conclusion, G has an edge from (R, i) to (S, j).
- for each t.g.d. and each vertex x that occurs in position i in a R-atom of the premise and x occurs somewhere in the conclusion and each variable y that is existentially quantified in a S-atom at position j of the conclusion, G has an edge from (R, i) to (S, j) labeled with a *.

We say D is weakly acylic iff its chase-flow graph has no cycle through an edge, marked *.
Note: t.g.d. $\forall \bar{x} \underbrace{\varphi(\bar{x})}_{\text{premise}} \implies \exists \bar{y} \underbrace{\psi(\bar{x}, \bar{y})}_{\text{conclusion}}$

Theorem 69. *(Deutsch, Popa, Tannen 2003) If D is weakly acylic, then chasing Q with D always terminates (in a number of steps polynomial in $|Q|$ and $|D|$, assuming fixed schema).*

Definition 70. (Minimality under constraints) Given set D of e.d.s, a CQ Q is D-minimal if there are no CQs S_1, S_2 where

- S_1 is obtained from Q by replacing zero or more variables in Q with other variables of Q
- S_2 is a strict subquery of S_1 such that $D \models Q \equiv S_1 \equiv S_2$.

For example,

$$Q(x,y) : -R(z,x), R(z,y)$$
$$\forall x,y,zR(x,y) \wedge R(x,z) \implies y = z$$

$$Q(x,y) : -R(z,x), R(z,y), x = y$$
$$S_1(x,y) := R(z,x), R(z,x)$$
$$S_2(x,y) := R(z,x)$$

Definition 71. (Chase & backchase - Deutsch, Popa, Tannen 2003) Given a CQ Q, set D of e.d.s, assume D is weakly acyclic.

1. (Chase): chase Q with D, producing $\mathbb{U}$, the universal plan for Q.
2. (Backchase):
   for each subquery Q_s of $\mathbb{U}$
   if Q_s is D-minimal, and $Q_s D \implies Q_s \equiv \mathbb{U}$ (tested by chasing) then output Q_s

Theorem 72. *Chase and backchase output precisely the D-minimal rewriting of Q. (If $D \models Q' \equiv Q$ and Q' is D-minimal, then Q' is isomorphic to a subquery of $\mathbb{U}$)*

Chase & backchase is interesting because it is a unifying and general technique for:

- optimizing queries using views
- answering queries using views

Given a CQ Q and a set of views definitions $\mathcal{V}$, find an efficient plan to answer Q (possibly using views in $\mathcal{V}$). For example:

$$Q(x,y) : -R(x,u), R(u,v), R(y,y)$$
$$V(x,y) : -R(x,u), R(u,y)$$
$$\implies Q'(x,y) : -V(x,z), R(z,y)$$

Encode using e.d.s

$$\forall x,y,zR(x,z) \wedge R(z,y) \implies V(x,y)$$
$$\forall x,yV(x,y) \implies \exists zR(x,z) \wedge R(z,y)$$

16 Wednesday, 2/10/10 (scribe: Vu)

Example for chase & backchase

$$Q(x,y) : -R(x,y), R(x,z), S(y,u), S(z,u)$$

$$D \begin{cases} d_1 \stackrel{\text{def}}{=} \forall u,v,wR(u,v) \wedge R(u,w) \implies v = w & \text{(e.g.d.)} \\ d_2 \stackrel{\text{def}}{=} \forall u,vR(u,v) \implies \exists wS(v,w) & \text{(t.g.d.)} \end{cases}$$

Step 1: Chase Q with D

$$Q \xrightarrow{d_1} Q_1 : Q_1(x,y) : -R(x,y), R(x,z), S(y,u), S(z,u), y = z$$

i.e.,

$$Q_1(x,y) : -R(x,y), S(y,u)$$

And the chase terminates with Q_1 as the universal plan.
Step 2: Backchase

$$Q_s(x,y) :- R(x,y)$$

$$Q_s \xrightarrow{d_2} Q' : Q'(x,y) :- R(x,y), S(y,w)$$

and the chase terminates.
Since $Q' \cong Q_1$, hence $D \models Q_s \equiv Q_1 \equiv Q$.
Output Q_s as the minimal rewriting.
Another application for chase & backchase: answering queries with views.
Two flavors: Given CQ Q and CQ view $\mathcal{V}$

1. Find rewriting of Q using any combination of source and view predicates - for performance.

2. Find rewriting of Q using only view predicates

Eg, when source database is remote/unavailable/non-existant.
Chase & backchase applies to both scenerios.
Given a CQ view

$$V(\bar{u}) :- \varphi(\bar{u},\bar{v})$$

we can model the view using a pair of t.g.d.s.

$$\forall \bar{u},\bar{v}\varphi(\bar{u},\bar{v}) \implies V(\bar{u})$$
$$\forall \bar{u}V(\bar{u}) \implies \exists \bar{v}\varphi(\bar{u},\bar{v})$$

For example,

$$Q(x,y) :- R(x,u), R(u,v), R(v,y)$$
$$V(u,v) :- R(u,w), R(w,v)$$

$$D \begin{cases} d_1 \stackrel{\text{def}}{=} \forall u,v,wR(u,w) \wedge R(w,v) \implies V(u,v) \\ d_2 \stackrel{\text{def}}{=} \forall u,vV(u,v) \implies \exists wR(u,w) \wedge R(w,v) \end{cases}$$

Step 1: chase Q with D

$$Q \xrightarrow{d_1} Q_1 : Q_1(x,y) :- R(x,u), R(u,v), R(v,y), V(x,v)$$
$$Q_1 \xrightarrow{d_1} Q_2 : Q_2(x,y) :- R(x,u), R(u,v), R(v,y), V(x,v), V(u,y)$$

and the chase terminates with Q_2 as the universal plan.
Step 2: backchase: out put of backchase step will be

$$Q'(x,y) :- R(x,u), V(u,y)$$
$$Q''(x,y) :- V(x,v), R(v,y)$$
$$Q'''(x,y) :- R(x,u), R(u,v), R(v,y)$$

Backchase with

$$Q' \xrightarrow{d_2} Q'_1 : Q'_1(x,y) :- R(x,u), V(u,y), R(u,w), R(w,y)$$
$$Q'_1 \xrightarrow{d_1} Q'_2 : Q_2(x,y) :- R(x,u), V(u,y), R(u,w), R(w,y), V(x,w)$$

Note that $Q'_2 \cong Q_2$, so $D \models Q' \equiv Q_2 \equiv Q$.
On the other hand, consider

$$Q_{bad}(x,y) :- R(x,u), R(v,y)$$

Backchase terminates after 0 steps: $Q_{bad} \not\cong Q_2$, so $D \not\models Q_{bad} \equiv Q_2$
Consider the query TC which computes transitive closure of a graph G

$$TC(G) \stackrel{\text{def}}{=} \{(x, y)| \text{ there exists a path from } x \text{ to } y \text{ in } G\}$$

Theorem 73. *(Compactness) A theory T (i.e., a set - possibly infinite set - of FO sentences) is consistent (i.e., $\exists A$ s.t. $A \models T$) iff every finite subset of T is consistent.*
(A corollary of Godël's Completeness Theorem)

Connectivity query:

$$conn(G) \stackrel{\text{def}}{=} \text{true iff } G \text{ is connected}$$

Ex: if TC is FO-expressible, then so is connectivity.

Proposition 74. *Connectivity over arbitrary (e.g., possibly infinite) graphs is not FO-expressible.*

Proof. Assume towards a contradiction that it is expressible, via FO sentence Φ.
Vocabulary $\sigma = \{E, c_1, c_2\}$. Now for every n, let ψ_n be the FO sentence.

$$\psi_n \stackrel{\text{def}}{=} \neg(\exists x_1, ..., x_n(E(c_1, x_1) \wedge E(x_1, x_2) \wedge ... \wedge E(x_n, c_2)))$$

i.e., ψ_n says there is no path of length n from c_1 to c_2.
Let T be the theory.

$$T = \{\psi_n|n > 0\} \cup \{\neg(c_1 = c_2), \neg E(c_1, c_2)\} \cup \Phi$$

Claim: T is consistent.
By compactness, we need to show every finite $T' \subseteq T$ is consistent. Let N be s.t. forall $\psi_n \in T', n < N$.
Then a connected graph in which the shortest path from c_1 to c_2 has length $N + 1$ is a model of T'.
Since C is consistent, it has a model, say $G \models T$. G is connected, but has no path of any length from c_1 to c_2 - a contradiction. ☐

17 Friday, 2/12/10 (scribe: Thanh)

Compactness: Any theory T is consistent if every finite subset of T is consistent.

Proposition 75. *Compactness fails for finite models, i.e, there exists a theory T which is unsatisfiable by any finite model, but every finite subset of T is finitely satisfiable.*

Proof: Let $T = \{\lambda_n|n > 0\}$ where λ_n is defined (over vocabulary $\sigma = \Phi$)

$$\lambda_n = \exists x_1, x_2, ..., x_n \bigwedge_{i \neq j} x_i \neq x_j$$

λ_n says the universe has at least n distinct finite elements. Clearly, T is not finitely satisfiable. On the other hand, any finite subset $T' \subset T$ if stisfiable by a model with universe > N where N is $max\{i|\lambda_i \in T'\}$

Ehrenfeucht-Fraisse Games (EF-games):

Two players, the spoiler and the duplicator. Given two finite structures, $\mathfrak{A}$ and $\mathfrak{B}$. In each round,
1. Spoiler picks one of $\mathfrak{A}$ or $\mathfrak{B}$ and an element of that structure.
2. Duplicator picks an element of the other structure.
After n rounds, let $(a_1, a_2, ..., a_n)$ and $(b_1, b_2, ..., b_n)$ be the elements chosen so far. We say that $(\bar{a}, \bar{b})$ is minimizing position for the duplicator if $(\bar{a}, \bar{c}^A)$, $(\bar{b}, \bar{c}^B)$ is the partial isomomorphism between A and B ($\bar{c}$ is the constansts in σ). We say that the duplicator was an n-round winning statergy if the duplicator can play in a way that guarantee a winning position after n rounds (other wise the spoiler has an n-round winning startergy).
If the duplicator has an n-round winning statergy we wrote $A \equiv_n B$ (Note that $A \equiv_n B \Rightarrow A \equiv_k B$ for $k \leq n$).

Definition 76. Let A,B be finite σ-structures, σ is relational and $\bar{a} = (a_1, a_2, ..., a_n)$ and $\bar{b} = (b_1, b_2, ..., b_n)$ two tuples over elements of A and B, repectively, then $(\bar{a}, \bar{b})$ defines a partial isomomorphism between A and B if following holds:
* For every $i, j \leq n$ $a_i = a_j$ if $b_i = b_j$
* For every constant symbol $c \in \sigma$ and every $i \leq n$, $a_i = c^A$ if $b_i = c^B$
* For every k-arry predicate symbol $P \in \sigma$ and every sequence $(i_1, i_2, ..., i_k)$ of (not nesscessarily distinct) number from [1,n]

$(a_{i_1}, a_{i_2}, ..., a_{i_k}) \in P^A$ iff $(b_{i_1}, ..., b_{i_k}) \in P^B$

Definition 77. The quantifier rank of a FO sentence ϕ is the depth of quantifier nesting $qr(\varphi)$
* If φ is atomic, $qr(\varphi) = 0$
* $qr(\phi \wedge \psi) = qr(\varphi \vee \psi) = max(qr(\varphi), qr(\vee))$
* $qr(\neg\varphi) = qr(\varphi)$
* $qr(\forall x\varphi) = qr(\exists x\varphi) = qr(\varphi) + 1$

Example 78. $qr(\exists x(\forall y R(x))) \vee \exists z P(z) = 2$

Denote FO[k] is the fragment of FO sentences $\varphi \in FO | qr(\varphi) \leq k$

Theorem 79. *Let $\mathfrak{A}, \mathfrak{B}$ be finite σ-structures, then the following are equivalent:*
1. A and B agree on FO[k], i.e. for any $\varphi \in FO[k]$, $A \models \varphi$ iff $B \models \varphi$
2. $A \equiv_k B$
A property P of finite σ-structures is not expressible in FO iff for any $k \in \mathbb{N}$, then exist two finite σ-structures A_k and B_k such that
* *$A_k \equiv_k B$*
* *A_k has property P, but B_k does not.*

Proof: Supposes φ expresses P, $k = qr(\varphi)$ and suppose have A_k, B_k such that A_k has P, B_k does not have P but $A_k \equiv_k B_k$. Then, $\varphi \models A_k$ iff $\varphi \models B_k$, a contradiction.

Example 80. games on sets vocabulary $\sigma = \Phi$. Supposes $|A|, |B| \geq n$. Then, $A \equiv_n B$.

Example 81. games on linear order. Let $\sigma = \{\lambda\}$ interpreted as a linear oreder. Supposes L_1, L_2 are two linear orders of size $\geq$ n. It is not true that $L_1 \equiv_n L_2$ even for n =2.
Let $L_1 = a < b < c$
Let $L_2 = u < v$ First round, spoiler picks L_1 and b. Duplicator picks u or v. Second round, if u picked, spoiler picks a (if v then c).

Example 82. If L_1 and L_2 are linear orders of size at least 2^k, then $L_1 \equiv_k L_2$
for: EVEN is not FO-expressible even if $\sigma = \{<\}$ and ¡ is a linear order where $EVEN^A = true$ if $|A|$ has even number of elements.
Proof: Supposes $\varphi \in FO$ expresses EVEN. Let $k = qr(\varphi)$.
Pick L_1 is linear order of size 2^k
Pick L_2 is linear order of size 2^{k+1}
Then, $L_1 \equiv_k L_2$, a contradiction.

18 Wednesday, 2/15/10 (scribe: Mingmin)

Theorem 83. *If L_1 and L_2 are linear orders (with σ=$\{<\}$) of size $\geq 2^k$ then $L_1 \equiv_k L_2$.*

Proof. $L_1 = \{1, \ldots, n\}$, $L_2 = \{1, \ldots, m\}$ where $n, m \geq 2^k$.
Define *distance* between x and y d(x,y)$\stackrel{def}{=}$ |x-y|.
Assume the vocabulary σ has constants *min* and *max*.
We claim that the dup. has a strategy s.t. after round i:

* Let $\bar{a} = (a_{-1},\ a_0,\ a_1, \ldots, a_i)$ and $\bar{b} = (b_{-1},\ b_0,\ b_1, \ldots, b_i)$ be moves played in L_1, L_2 respectively.
  $a_{-1} = min^{L_1} = 1,\ a_0 = max^{L_1} = n; b_{-1} = min^{L_2} = 1,\ b_0 = max^{L_2} = m.$

* Then for $-1 \leq j, l \leq i$
  IH{
  1. if $d(a_j, a_l) < 2^{k-i}$, then $d(a_j, a_l) = d(b_j, b_l)$.
  2. if $d(a_j, a_l) \geq 2^{k-i}$, then $d(b_j, b_l) \geq 2^{k-l}$.
  3. $a_j \leq a_l \iff b_j \leq b_l \implies$ the partial isomorphism condition.

Proof of claim: By induction on L:

base case: (i=0) immeiate since $d(a_{-1}, a_0), d(b_{-1}, b_0) \geq 2^k$ by assumption.
induction case: Sps spoiler picks L_1 in round i+1, plays a_{i+1} (case for L_2 is symmetric)
case 1: $a_{i+1} = a_j$ for some j≤i⇒dup. responds with b_j.
case 2: spoiler plays new point a_{i+1}, falling into some interval, $a_j < a_{i+1} < a_l$, s.t. no other moves previously played in the same interval. By IH(3), the corresponding interval (b_j, b_l) also does not contain any previously played elements.

There are two (sub)cases:

(a) $d(a_j, a_l) < 2^{k-i}$, dup. responds with the element b_{j+1} in interval (b_j, b_l) s.t. $d(b_j, b_{i+1}) = d(a_j, a_{i+1})$.

(b) $d(a_j, a_l) \geq 2^{k-i}$. There are 3 cases:

  * $d(a_j, a_{i+1}) < 2^{k-(i+1)}$ and $d(b_{i+1}, b_l) \geq 2^{k-(i+1)}$.
  * $d(a_{i+1}, a_l) < 2^{k-(i+1)}$ and $d(a_j, a_{i+1}) \geq 2^{k-(i+1)}$. Similar to first case.
  * $d(a_j, a_{i+1}), d(a_{i+1}, a_l \geq 2^{k-(i+1)})$.

□

Corollary 84. *Graph connectivity (and also transitive closure) is not FO-expressible even over graphs with a linearly-ordered domain.*

Proof. Define a successor relation $\text{succ}(x, y) \equiv x < y \land \forall z(z \leq x \lor z \geq y)$.
Define $\gamma(x, y)$ as the FO formula s.t. $\gamma(x, y)$ holds iff one of the following is true:

* y is the succ. of the succ of x.

* x is the predecessor of the last element and y is the first element.

* x is the last element, and y is the succ. of the first element.

We can show that when underlying order is even, γ-graph is disconnected; odd, γ-graph is connected.
Now, sps connectivity is FO-expressible via sentence Φ over vocab. $\sigma = \{<, E\}$.
Take Φ replace every occurrence of E by γ
Call the result Ψ. But Ψ expresses EVEN, a contradiction. □

Datalog:

T(x,y):- E(x,y) T(x,y):- E(x,z),T(z,y)

19 Friday, 2/17/10 and Monday, 2/20/10 (scribe: Bertram)

Reading. Sections 12.1–12.3, Foundations of Databases, Abiteboul, Hull, Vianu, 1995 (e-version available).

Datalog Syntax. We already know conjunctive queries which can be expressed as rules r of the form

- $A_0(\bar{x}_0) :- A_1(\bar{x}_1), \ldots, A_n(\bar{x}_n)$

Here each $A_i(\bar{x}_i)$ is a logic *atom*, with A_i a k-ary relation symbol and $\bar{x}_i$ a k-tuple of variables or constants. All variables occurring in the *head* $A_0(\bar{x}_0)$ of r must also occur in the *body* (the rhs) of r.[1]
A finite set of rules P of the above form is called a *Datalog program.* The relations occurring only in the body of rules of P are called *edb*-relations; those that occur in the head (and possibly in the body) are called *idb*-relations. Thus, we can associate with a Datalog program P a schema $sch(P)$ as follows:

- $sch(P) = edb(P) \cup idb(P)$.

As we shall see, at Datalog program P can be given a query semantics, i.e., it can be viewed as a mapping from instances of $edb(P)$ to instances of $idb(P)$.

Example. Consider the following Datalog program P_{tc} with edb-relation G and idb-relation T:

$$\begin{array}{l} T(x,y) :- G(x,y). \\ T(x,y) :- G(x,z), T(z,y). \end{array}$$

It maps instances of the edge relation $G/2$ of a graph to instances of another relation $T/2$, the transitive closure of G.

Model-theoretic Semantics of Datalog. We can view a Datalog program P as a set of first-order logic formulas Φ_P; e.g. for P_{tc} we get[2]

- $\Phi_{P_{tc}} = \{\forall x, y : T(x,y) \leftarrow G(x,y), \quad \forall x, y, z : T(x,y) \leftarrow G(x,z) \wedge T(z,y)\}$.

We can ask ourselves: what are the *models* of $\Phi_{P_{tc}}$, given a particular input instance of $G/2$? We can focus our attention on *Herbrand Models.*

Herbrand Models. A (Herbrand) model M of a set of (closed) formulas Φ is a (Herbrand) interpretation that satisfies all sentences in Φ, denoted $M \models \Phi$. A *Herbrand interpretation* is one where symbols are interpreted "syntactically", i.e., they stand for themselves (in general, an interpretation maps symbols from the syntactic domain to a semantic domain). Thus, a Herbrand interpretation is built over a *Herbrand universe*, consisting of constants (and if function symbols are allowed: *ground terms*, i.e., function terms containing only constants as arguments, but no variables). Using constants (and other ground terms), the *Herbrand base* $\mathcal{B}_P$ of ground atoms is built.
To simplify the presentation, we sometimes denote by P a Datalog program together with a set of (ground) facts, i.e., and instance of $edb(P)$. For example we might have $P =$

$$\begin{array}{l} T(x,y) :- G(x,y). \\ T(x,y) :- G(x,z), T(z,y). \\ \\ G(a,b). \\ G(b,c). \\ G(c,d). \end{array}$$

Here we have as set of constants $\mathcal{C} = \{a, b, c, d\}$ and relation symbols $\mathcal{R} = \{G/2, T/2\}$. The Herbrand base $\mathcal{B}_P$ of P consists of all ground atoms that can be built using $\mathcal{C}$ and $\mathcal{R}$; here:

[1] Later, when allowing negated literals $\neg B_i(\bar{x}_i)$ in the body, we require that all variables in a rule appear in some *positive* literal of the body. Such rules are called *range-restricted*; this implies they are domain-independent.

[2] The arrow $A \leftarrow B$ is just a shorthand for $\neg B \vee A$. The $\forall z$ in the second formula can be moved inside as follows: $\forall x, y : T(x,y) \leftarrow \exists z : G(x,z) \wedge T(z,y)$.

$\{G(a,a), G(a,b), \ldots G(d,d), T(a,a), \ldots, T(d,d)\}$. Since we consider only finite database instances, $\mathcal{B}_P$ will be finite as well.[3]
For Herbrand interpretations, we simply view constants symbols as distinct domain elements, and any subset $I \subseteq \mathcal{B}_P$ can be viewed as an interpretation assigning *True* to all ground atoms $A(\bar{c}) \in I$ and *False* to all $A(\bar{c}) \notin I$.

Immediate Consequence Operator T_P. Given a Datalog program (with *edb*) P and an interpretation $I \subseteq \mathcal{B}_P$, we can compute the set of *immediate consequences*

$$T_P(I) := \{head(r) \mid I \models body(r), r \in ground(P)\}$$

Here, $ground(P)$ is the finite set of ground rules that can be obtained from P by substituting variables in all possible ways by constants from $\mathcal{C}$. So given I, we find the ground rules whose body is satisfied, then derive the ground atom in the head of the rule.

Fact: An interpretation I is a model of P, denoted $I \models P$ iff $T_p(I) \subseteq I$. To see this, note that if there was a head atom $A(\bar{c})$ derived by T_P but not in I, then the corresponding ground rule would be violated (the body of that rule must be true for the head to be derived; yet the head is not in I.)

Fact: $\mathcal{B}_P$ is a (Herbrand) model of P (or of Φ_P to be more precise). To see this, note that $\mathcal{B}_P$ makes true all heads of rules of P, for all possible ground instances of those rules.[4]
Clearly $\mathcal{B}_P$ is usually not the desired model. In the example, it contains "unsupported" atoms such as $G(a,a)$. We can also "make up" models by adding "self-supporting" atoms to $T/2$, without the underlying graph $G/2$ necessarily having a corresponding transitive path.

Minimal Model M_P. Consider a Datalog program P (with *edb*). Let $I \subseteq \mathcal{B}_P$ be any Herbrand interpretation of P. We define M_P as the intersection of all models of P, i.e.,

$$M_P = \bigcap_{I \subseteq \mathcal{B}_P} \{\ A \in I \ \mid\ I \models P\ \}$$

Note that M_P is unique and minimal (no proper subset of M_P is a model of P).

Fixpoint Semantics. Computing the intersection of a large (even if finite) set of interpretations is highly inefficient. Instead we can compute the semantics of P bottom up using T_P. First, note that T_P is *monotone*, i.e., $I \subseteq J$ implies $T_P(I) \subseteq T_P(J)$.[5] With this, and the fact that $\mathcal{B}_P$ is finite, we can see that the following iteration reaches a fixpoint:

- $T_P^0 = \emptyset$,
- $T_P^{n+1} = T_P(T_P^n)$.

Note: In the base step T_P^0, we already derive all facts (= edb-instance).

Theorem. The fixpoint T_P^ω coincides with M_P.

[3]Exercise:What is the size of $\mathcal{B}_P$? Is it polynomially bounded by the size of the given edb?
[4]$G/2$ are viewed as implications of the form $G(\ldots) \leftarrow true$.
[5]This is no longer the case, if Datalog rules have negative literals $\neg B$ in the body.

Proof-theoretic Semantics. The fixpoint semantics can be implemented in an obvious way in a bottom-up, set-oriented way (example on blackboard).
There is an alternative, top-down and tuple-oriented way. For that, we view the rules of a Datalog program P as inference rules. We start with a fact that we want to derive, e.g. $T(a, d)$ in the example. We then apply the rules "backwards" to obtain new proof obligations. This yields a proof-tree with leaves that are either successful (we can fulfill our proof obligations using the given (*edb*) facts, or that fail (the fact we need to assume is not in the *edb*). For details see [AHV95, Section 12.4].

Adding Negation

Datalog can express certain recursive queries, but not negation. First-order logic can express negation but not recursion. The combination of "recursion through negation" can be problematic (and in fact has fueled the research areas of KR and Nonmonotonic Reasoning and Logic Programming in the 1990). The problems can be illustrated with ground (propositional) rules.

Syntax of Datalog$^\neg$**.** If we allow negated literals $\neg A(\bar{x})$ in the body (not in the head!) of rules, we obtain Datalog$^\neg$. We also require that every variable in a Datalog$^\neg$ rule appears positively in the body, i.e., we assume rules are *range-restricted*.
Note that for Datalog$^\neg$, T_P is no longer monotone!
To illustrate the problems with negation, we can focus on propositional cases (after all, we can replace a Datalog$^{(\neg)}$ program with *edb* by its finite ground instantiated version.

Multiple Minimal Models (Stratified Case). Consider the program $P_1 = \{p :- q,\ q :- \neg r\}$. It has two different minimal models $M_1 = \{p, q\}$ and $M_2 = \{r\}$. For M_1 we can assume r to be false (there is no fact or rule that could derive r), hence q and thus p must be true. This models seems to be the "right" (intended) one. On the other hand, M_2 is a minimal model, too: If r is true, then the body of the second rule is false, so we do not have to derive q (and thus we don't need to derive p either). So somehow we want to say that M_1 is preferred over M_2. The notion of stratification will do the job (see below/later).

Multiple Minimal Models (Non-Stratified Case). Now consider the following program $P_2 = \{p :- \neg q,\ q :- \neg p\}$. Again we have two minimal models; here: $M_1 = \{p\}$ and $M_2 = \{q\}$. But now we cannot say that one model is "better" than the other (for they are complete symmetrical). The STABLE semantics, assigns to any program a set of models. The stratified semantics excludes such programs from consideration (because the program contains a negative cylic dependency). The well-founded model semantics (WFM) assigns a third thruth value to atoms that are "ambiguous". Here WFM would make p and q undefined.

Stratified Datalog$^\neg$**.** A Datalog$^\neg$ program P is called *stratified* if its dependency graph G does not contain negative cycles. The nodes of the dependency graph are the relation symbols of P. There is an edge from R to S in G if there is a rule in P such that R occurs in the body and S occurs in the head. The edge is called *negative* and labled with "$\neg$" if R occurs under a negation symbol in the body.
It is easy to see that if a program P is stratified, then we can create a partition of rules $P = P_1 \cup \cdots \cup P_n$, such that the different "layers" of rules P_i can be evaluated one after another. That is, the relations defined by rules in P_i only depend on relations defined by P_j with $j \leq i$, and only depend negatively on relations p_j with $j < i$ (i.e., from lower strata).

Beyond Stratified Datalog. There are certain queries that involve negation and recursion that cannot be expressed in stratified Datalog$^\neg$ (or S-Datalog for short). There are several extensions that are strictly more expressive:
Inflationary Datalog (I-Datalog) corresponds to IFP (inflationary fixpoint logic), which itself is equivalent ot LFP. The idea of I-Datalog is to allow negation in the head of rules and evaluate this rules

similar to the T_P iteration, but (somewhat artificially) keeping all previously derived facts $T'_P(I) = T_P(I) \cup I$.
Non-inflationary Datalog (P-Datalog) corresponds to PFP (partial fixpoint logic). Again we allow negation in the head (which is interpreted as "deletion") but w allow the fixpoint iteration to diverge (we use T_P instead of T'_P).
I-Datalog and P-Datalog have been considered by the database and finite model theory community (as syntactic variants of IFP and PFP) to study the expressive power and computational complexity of query languages. For KR and NMR purposes, however, they have not gained traction. Instead, the **well-founded Datalog** (WF-Datalog) semantics and the **stable semantics** (sets of stable models, ST-Datalog) have been adopted by the KR, NMR, and LP communities.
A model $M \models P$ is called *stable* if after replacing in $ground(P)$ all atoms in the rule bodies with their truth value in M, the resulting reduced program (now a program without negation!) has as its unique minimal model exactly M. A program can have zero or more stable models (e.g. $p :- \neg p$ has not stable models, whereas P_2 above has the indicated two stable models).
The WF-Datalog semantics, on the other hand, is a 3-valued model. It can be computed iteratively by an alternating fixpoint construction. The basic idea is to keep track of two subsets of $\mathcal{B}_P$, i.e., those atoms that are definitely true and those that are definitely false (or equivalently those that are true or undefined). One can then define a fixpoint operator Γ^2_P whose lfp and complement of gfp yield the definitely true and definitely false facts, respectively; all remaining atoms are assigned the truth-value *undefined*.
The well-founded semantics yields the intended meaning of non-stratified programs such as

$$win(X) :- move(X, Y), \neg win(Y)$$

In fact, it can be shown that all Fixpoint queries (LFP, IFP) can be expressed in the form of a win-move game, i.e., with a quantifier-free FO formula for *move* and a single recursive rule

$$win(\bar{X}) :- move(\bar{X}, \bar{Y}), \neg win(\bar{Y})$$

and that this win-move game can even be reduced to a draw-free game (Flum, Kubierschky, Ludäscher, ICDT'97).

20 Friday, 2/26/10 (scribe: Zhongxian)

Theorem 85. *Containment/equivalence of Datalog program is undecidable.*

- *containment: (CGKV 1988, Shmueli 1987)*
- *equivalence: (Shmueli 1993)*

Theorem 86. *Equivalence of Datalog and NR-Datalog (non-recursive) is decidable, but complete for 3-EXPTIME (Chaudhuri and Vardi, 1992)*

Theorem 87. *Boundedness of Datalog program is undecidable. (GMSV 1993)*

Theorem 88. *WF-Datalog¬ + order captures PTIME. (Vardi 1982)*

Theorem 89. *(Fagin 1976)*

- $\exists SO = NP$
- $\forall SO = CO - NP$

21 Monday, 3/01/10 (scribe: Thanh)

Bag-theoretic Sematics

Definition 90. a bag (or multiset) is a group of annotated elements

Example 91. {(CS140,Olsson);[2],(CS235,chen);[1]}

$$\underset{\text{multiplicity}}{\uparrow}$$

Definition 92. A sub bag B_s of bag B (written $\leq_b$) where:
1) every element $b \in B_s$ is $\in B$
2) multiplicity of b —b— in B $\geq$ —b— in B_s
* This is more restrictive than subset *

Optimizing querries

- Containment:

Definition 93. Q is bag-contained in Q' (writtten Q $\leq_b$ Q') iff $Q(I) \leq_b Q'(I)$ for any bag database instance I
* strictly stronger than set containment *

Proposition 94. *1) For any Q, Q' if Q $\leq_b$ Q' then Q $\leq_b$ Q'*
2) $\exists$ CQs Q,Q' s.t Q $\leq_s$ Q' but Q $\nleq_b$ Q'

Example 95. two relations {(R,x;[3]} and {(A,x;[2]),B,y;[1]),(c,x;[1])}
join: cross product
{(R,x,A,x;[6]), ..., (R,x,C,x;[3])}

Complexity: bag-containment is more complex, possibly undecidable.

Equivalence:

Definition 96. Two CQs Q is bag-equivalent to Q' (written Q $\equiv_b$ Q') iff Q(I) $=_b$ Q'(I) for any bag-database I, that is, Q $\leq_b$ Q' and Q' $\leq_b$ Q

Theorem 97. *Q $\equiv_b$ Q' iff Q and Q' are graph isomorphic.*

Corollary 98. *$\equiv_b$ is polynomially equivalent to graph isomorphism (not to be known in NP-complete).*

This means we can not reduce bag querries!
Complexity: NP, but not known NP-complete or in PTIME ?
UCQs: on SQL: A UNION ALL B

Definition 99. A bag union ($\sqcup$) B $\sqcup$ B' is: { unique elements of B} $\cup$ {unique elements of B'} $\cup$ {shared elements b, $|b|$ in B + $|b|$ in B' }

Proposition 100. *$\exists$ CQs Q,Q',Q" s.t*
Q $\leq_b$ Q' $\sqcup$ Q" but
Q $\nleq_b$ Q' and Q $\nleq_b$ Q"

Complexity:

Theorem 101. *bag containment of UCQs is undecidable*

Proof:

Definition 102. a Diophantine Equation is of the form: $\Lambda(x_1, x_2, ..., x_k) = \Phi$, a polynomial of variables with no constant and integer cofficients

Solve for integer-vallues roots.

Theorem 103. *the solution to Diophantine is undeciable rewritten as:* $\forall\ x_1, ..., x_k)\ \Phi_1(x_1, ..., x_k) \leq \Phi_2(x_1, ..., x_k)$ *such that* Φ_1, Φ_2 *are positive*

Example 104. $2x^2y + yz \leq x^2y + 2xy + x^3$
take predicate P, constants a,b,c
Q() :- P(a), P(a), P(b)
    :- P(a), P(a), P(b)
    :- P(b), P(c)
Q'() :- P(a), P(a), P(b)
    :- P(a), P(b)
    :- P(a), P(b)
    :- P(a), P(b), P(a)
$\therefore$ bag-containment of UCQ is undecidable

22 Wednesday, 3/03/10 (scribe: Mingmin)

Incomplete & Probabilistic information

Example 105. Codd table:

R

0	1	@
@	@	1
2	0	@

where @ means null value

Definition 106. The incomplete database represented by a table defined as follows

$$rep(T) = \{v(T) \mid v\ is\ a\ valuation\ of\ variables\ in\ T\}$$

Definition 107. Consider a table T and a query q. For each $I \in rep(T)$, q produces an answer q(I). The set of all possible answer q(rep(T)) is an incomplete database.

Goal: For each representation T and a query q, there exists a representation. $\overline{q}(T)$ such that $rep(\overline{q}(T)) = q(rep(T))$.

Definition 108. If some representation system $\mathcal{T}$ has the property described for a query language L, then we say that $\mathcal{T}$ is a strong representation system for $\mathcal{L}$.

Definition 109. For a table T and a query q, the set of sure fact $sure(q,\ T)$ is defined as

$$sure(q,\ T) = \bigcap\{q(I)|I \in rep(I)\}.$$

Example 110. $sure(q,\ T) = \emptyset,\ q' = \Pi_{1,2}(R)$
$sure(q'(q(rep(T)))) = \{< 2, 0 >\},\ q'(sure(q, T)) = \emptyset.$

Definition 111. $\mathcal{L}$ is a query language, two incomplete DBs I, J are L equivalent, $I \equiv_L J$ if for each q in L we have

$$\bigcap\{q(I)|I \in \mathcal{I}\} = \bigcap\{q(J)|J \in \mathcal{I}\}$$

Definition 112. A representation system is weak for $\mathcal{L}$ if for each T of a DB and each q in $\mathcal{L}$, there exists a representation $\overline{q}(T)$ s.t.

$$rep(\overline{q}(T)) \equiv_{\mathcal{L}} q(rep(T)).$$

Theorem 113. *Codd tables form a weak representation system for selection-project*

$$\overline{\sigma}_{cond}(T) = \{t|t \in T \ and \ cond(v(t)) \ holds \ for \ all \ valuations \ of \ vars \ in \ T\}$$

Example 114. $\overline{\sigma}_{1=`2'}(T) = \{(2, 0, @)\}$.

v-table:

R		
0	1	x
x	z	1
2	0	v

Theorem 115. *A condition is conjunct of equality/inequality atoms, x=y,x=c,x≠y,x≠c, where x,y are vars and c constant..*

Definition 116. Φ is condition, a valuation V satisfies Φ if its assignment of const. to var. makes the form be true.

Definition 117. A condition table is triple (T, Φ_T, φ)

- T is a v-table
- Φ_T is global condition
- φ is a mapping over T that associates a local condition φ_t with each tuple $t \in T$

Theorem 118. *For each c-table T and a relational algebra query q, one constant c-table $\overline{q}(T)$ such that $rep(\overline{q}(T)) = q(rep(T))$.*

Proposition 119.

- $[\![E_1 \times E_2]\!](t) = [\![E_1]\!](t) \wedge [\![E_2]\!](t)$
- $[\![E_1 \cap E_2]\!](t) = [\![E_1]\!](t) \vee [\![E_2]\!](t)$
- $[\![\sigma_{i=`c'} E_1]\!](t) = [\![E_1]\!](t) \wedge t[i] = `c'$
- $[\![\Pi_{i_1,i_2,\ldots,i_k} E_1]\!](t) = \bigvee_{t' \ s.t. \ \Pi_{i_1,i_2,\ldots,i_k}} E_1(t') = t[\![E_1]\!](t')$