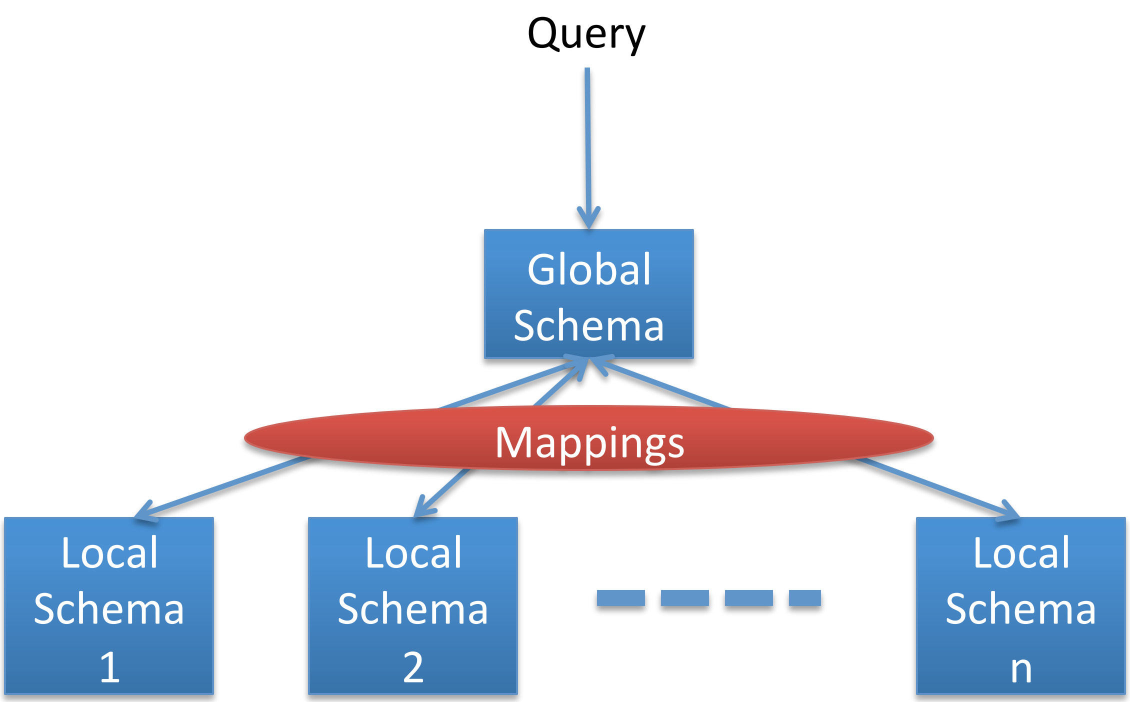
Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) **Virtual Data Integration**
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance

- **Virtual Data Integration**



ILLINOIS INSTITUTE OF TECHNOLOGY

Problems:

- How to create mappings?
  - Discussed in previous part of the course
- How to compute query Q
  - This is the main focus of this part

4. Query Answering with Views

- **How to compute query Q over global schema based on source schemas only?**
  - What language is used to express mappings?
  - What language due we allow for Q?
  - What language(s) can we use to query local sources?
  - What language can we use to compute Q from query results returned by local sources?
  - How to deal with incompleteness?

4.1 Query Answering with Views

Example: Solutions

Local Schema

```
Person
    Name
    Address

Address
    Id
    City
    Office-contact
```

Global Schema

```
Person
    Name
    Address
    Office-phone
    Office-address
    Home-phone
```

$$\forall x, y, z, a : Person(x, y) \wedge Address(y, z, a) \rightarrow \exists b, c : Person(x, z, a, b, c)$$

Query: `Q(Name) :- Person(Name, A, OP, OA, HP).`

Name	Address
Peter	1
Alice	2
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

4.1 Query Answering with Views

Example: Solutions

Local Schema

Name	Address
Peter	1
Alice	2
Bob	3

Global Schema

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

```
City
Office-contact
```

$$\forall x, y, z, a : Person(x, y) \wedge Address(y, z, a) \rightarrow \exists b, c : Person(x, z, a, b, c)$$

Query: `Q(Name) :- Person(Name, A, OP, OA, HP).`

Rewritten query over the source:

```
Q(Name) :- Person(Name, AI),
           Address(AI,A,OP).
```

Name
Peter
Alice
Bob

Example: Solutions

Local Schema

```
Person
    Name
    Address

Address
```

Global Schema

```
Person
    Name
    Address
    Office-phone
    Office-address
    Home-phone
```

Values of home-phone are not available in the source

$$\forall x, y, z, a : Person(x, y) \wedge Address(y, z, a) \rightarrow \exists b, c : Person(x, z, a, b, c)$$

Query: `Q(Home-ph) :- Person(N, A, OP, OA, Home-ph).`

Name	Address
Peter	1
Alice	2
Bob	3

Id	City	Office-contact
1	Chicago	(312) 123 4343
2	Chicago	(312) 555 7777
3	New York	(465) 123 1234

4. Query Answering with Views

- **Problems**
  - How to determine whether query can be answered at all?
  - Given a rewriting of the query using views, how do we know it is correct?
  - What to do if views can only return some of the query results?

Motivating Example (Part 1)

ILLINOIS INSTITUTE OF TECHNOLOGY

Movie(ID,title,year,genre)
Director(ID,director)
Actor(ID, actor)

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_1(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_1 \supseteq Q \quad \Rightarrow \quad Q'(T,Y,D) :- V_1(T,Y,D), Y \geq 1950$$

Containment is enough to show that V_1 can be used to answer Q.

Motivating Example (Part 2)

ILLINOIS INSTITUTE OF TECHNOLOGY

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_2(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_3(I,D) :- Director(I,D), Actor(ID,D)$$

Containment does not hold, but intuitively, V_2 and V_3 are useful for answering Q.

$$Q''(T,Y,D) :- V_2(I,T,Y), V_3(I,D)$$

How do we express that intuition?

Answering queries using views!

Problem Definition

ILLINOIS INSTITUTE OF TECHNOLOGY

Input: Query Q

View definitions: $V_1,\ldots,V_n$

A rewriting: a query Q' that refers only to the views and interpreted predicates (comparisons)

An equivalent rewriting of Q using $V_1,\ldots,V_n$: a rewriting Q', such that $Q' \Leftrightarrow Q$

Naïve approach

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Given Q and views**
  - Randomly combine views into a query Q'
  - Check equivalence of Q' and Q
  - If Q' is equivalent we are done
  - Else repeat
- **Why is this not good?**
  - There are infinitely many ways of combining views
    - E.g., V, V x V, V x V x V, …
  - We are not using any information in the query

Motivating Example (Part 3)

Movie(ID,title,year,genre)
Director(ID,director)
Actor(ID, actor)

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_4(I,T,Y) :- Movie(I,T,Y,G), \underline{Y \geq 1960}, G = "comedy"$$
$$V_3(I,D) :- Director(I,D), Actor(ID,D)$$
$$Q'''(T,Y,D) :- \underline{V_4(I,T,Y)}, V_3(I,D)$$

maximally-contained rewriting

Maximally-Contained Rewritings

Input: Query Q

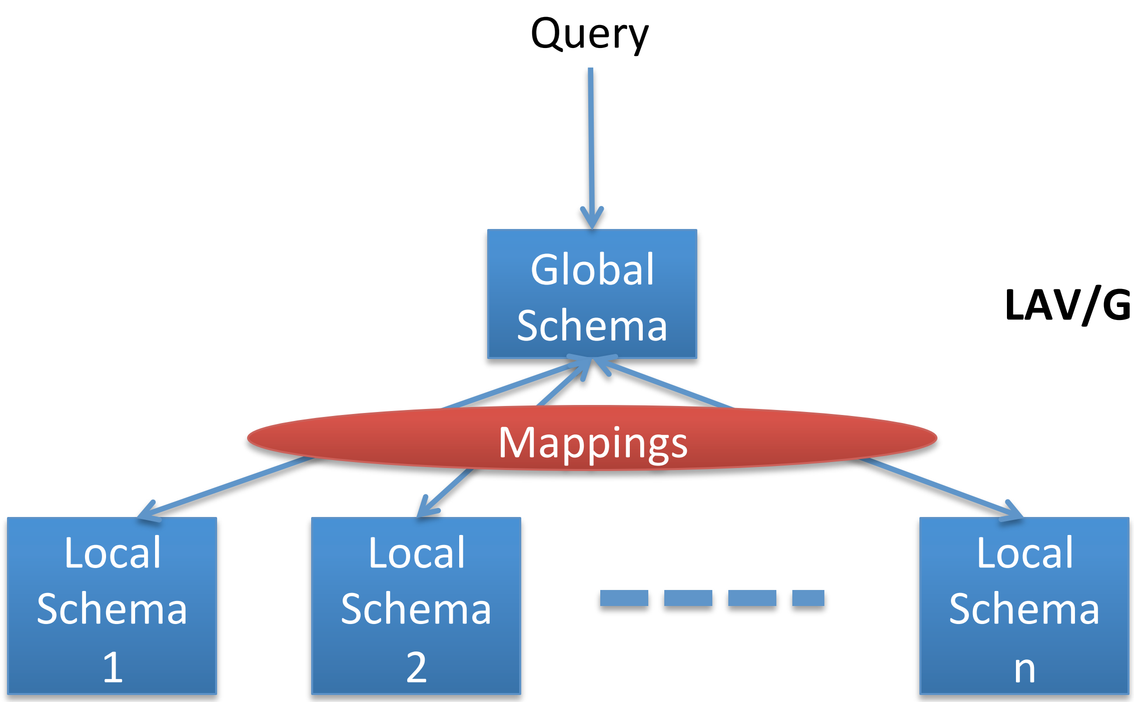
Rewriting query language $\boldsymbol{L}$

View definitions: $V_1,\ldots,V_n$

Q' is a maximally-contained rewriting of Q given $V_1,\ldots,V_n$ *and* $\boldsymbol{L}$ if:

1. $Q' \in \boldsymbol{L}$,
2. $Q' \subseteq Q$, *and*
3. *there is no* Q'' *in* $\boldsymbol{L}$ *such that* $Q'' \subseteq Q$ *and* $Q' \subset Q''$

Why again?



LAV/GLAV!

Other use-cases

- Query optimization with materialized views
  - Need equivalent rewritings
  - Implemented in many commercial DBMS
  - Here interest is cost: how to speed-up query processing by using materialized views

Exercise: which of these views can be used to answer *Q*?

ILLINOIS INSTITUTE OF TECHNOLOGY

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_2(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_3(I,D) :- Director(I,D), Actor(I,D)$$

$$V_6(T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_7(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950,$$
$$G = "comedy", Award(I,W)$$

$$V_8(I,T) :- Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$

Algorithms for answering queries using views

- **Step 1**: we' ll bound the space of possible query rewritings we need to consider (no comparisons)
- **Step 2**: we' ll find efficient methods for searching the space of rewritings
  - **Bucket** Algorithm, **MiniCon** Algorithm
- **Step 2b**: we consider "logical approaches" to the problem:
  - The **Inverse-Rules** Algorithm

Bounding the Rewriting Length

Theorem: if there is an equivalent rewriting, there is one with at most *n* subgoals.

Query: $$Q(\overline{X}) :- p_1(\overline{X_1}),\ldots,p_n(\overline{X_n})$$

Rewriting: $$Q'(\overline{X}) :- V_1(\overline{X_1}),\ldots,V_m(\overline{X_m})$$

φ

Expansion: $$Q''(\overline{X}) :- \underbrace{g_1^1,\ldots g_k^1}, \ldots, \underbrace{g_1^m,\ldots,g_j^m}$$

Proof: Only *n* subgoals in *Q* can contribute to the image of the containment mapping φ

Complexity Result [LMSS, 1995]

- Applies to queries with no interpreted predicates.
- Finding an equivalent rewriting of a query using views is NP-complete
  - Need only consider rewritings of query length or less.
- Maximally-contained rewriting:
  - Union of all conjunctive rewritings of length *n* or less.

The Bucket Algorithm

Key idea:

- Create a bucket for each subgoal *g* in the query.
- The bucket contains views that contribute to *g*.
- Create rewritings from the Cartesian product of the buckets (select one view for each goal)

- **Step 1**: assign views with renamed vars to buckets
- **Step 2**: create rewritings, refine them, until equivalent/all contained rewriting(s) are found

The Bucket Algorithm

ILLINOIS INSTITUTE OF TECHNOLOGY

Step 1:

- We want to construct buckets with views that have partially mapped variables
- For each goal **g** = R in query
- For each view **V**
- For each goal **v** = R in **V**
  - If the goal has head variables in the same places as g then
    - rename the view head variables to match the query goal vars
    - choose a new unique name for each other var
    - add the resulting view atom to the bucket

The Bucket Algorithm

Step 1 Intuition

- A view can only be used to provide information about a goal R(X) if it has a goal R(Y)
  - `Q(X) :- R(X,Y)`
  - `V(X) :- S(X,Y)`
- If the query goal contains variables that are in the head of the query, then the view is only useful if it gives access to these values (they are in the head)
  - `Q(X) :- R(X,Y)`
  - `V(X) :- S(X,Y), R(Y,Z)`

Bucket Algorithm in Action

$$Q(ID, Dir) :- Movie(ID, title, year, genre), Revenues(ID, amount),$$
$$Director(ID, dir), amount \geq \$100M$$

$$V_1(I, Y) :- Movie(I, T, Y, G), Revenues(I, A), I \geq 5000, A \geq \$200M$$
$$V_2(I, A) :- Movie(I, T, Y, G), Revenues(I, A)$$
$$V_3(I, A) :- Revenues(I, A), A \leq \$50M$$
$$V_4(I, D, Y) :- Movie(I, T, Y, G), Director(I, D), I \leq 3000$$

View atoms that can contribute to *Movie*:

V_1(**ID**,year'), V_2(**ID**,A'), V_4(**ID**,D',year'')

Buckets and Cartesian product

Movie(ID,title, year,genre)	**Revenues(ID, amount)**	**Director(ID,dir)**
V_1(**ID**,year)	V_1(**ID**,Y')	V_4(**ID**,**Dir**,Y')
V_2(**ID**,A')	V_2(**ID**,amount)	
V_4(**ID**,D',year)		

Consider first candidate rewriting: first V1 subgoal is redundant, and V1 and V4 are mutually exclusive.

$$q_1'(ID,dir) :- V_1(ID,\cancel{year}),V_1(ID,y'),V_4(ID,dir,y')$$

Next Candidate Rewriting

Movie(ID,title,year,genre)	Revenues(ID,amount)	Director(ID,dir)
V_1(**ID**,year)	V_1(**ID**,Y')	V_4(**ID**,Dir,Y')
V_2(**ID**,A')	V_2(**ID**,amount)	
V_4(**ID**,D',year)		

$$q_2'(ID,dir) :- V_2(ID,A'), V_2(ID,amount), V_4(ID,dir,y')$$

$$q_2'(ID,dir) :- V_2(ID,amount), V_4(ID,dir,y'),$$
$$amount \geq \$100M$$

The Bucket Algorithm

ILLINOIS INSTITUTE OF TECHNOLOGY

Step 2:

- For each combination of one element of each bucket:
- Create query Q' with query Q's head and list all these view atoms in the body
- If Q' equivalent to Q (or contained in Q)
  - Done (equivalent)
  - Add to union of CQs for contained case
- If not try to add comparisons

The Bucket Algorithm: Summary

ILLINOIS INSTITUTE OF TECHNOLOGY

- Cuts down the number of rewriting that need to be considered, especially if views apply many interpreted predicates.
- The search space can still be large because the algorithm does not consider the interactions between different subgoals.
  – See next example.

The MiniCon Algorithm

ILLINOIS INSTITUTE OF TECHNOLOGY

$$Q(title, year, dir) :- Movie(ID, title, year, genre), Director(ID, dir), Actor(ID, dir)$$

$$V_5(D, A) :- Director(I, D), Actor(I, A)$$

Intuition: The variable *I* is not in the head of V_5, hence V_5 cannot be used in a rewriting.
MiniCon discards this option early on, while the Bucket algorithm does not notice the interaction.

MinCon Algorithm Steps

ILLINOIS INSTITUTE OF TECHNOLOGY

- **1) Create MiniCon descriptions (MCDs)**:
  - Homomorphism on view heads
  - Each MCD covers a set of subgoals in the query with a set of subgoals in a view
- **2) Combination step**:
  - Any set of MCDs that covers the query subgoals (without overlap) is a rewriting
  - No need for an additional containment check!

MiniCon Descriptions (MCDs)

An atomic fragment of the ultimate containment mapping

$$Q(title, act, dir) :- Movie(ID, title, year, genre),$$
$$Director(ID, dir), Actor(ID, act)$$

$$V(I, D, A) :- Director(I, D), Actor(I, A)$$

MCD:

mapping:

$$ID \rightarrow I$$
$$dir \rightarrow D$$
$$act \rightarrow A$$

covered subgoals of *Q: {2,3}*

MCDs: Detail 1

$$Q(title, year, dir) :- Movie(ID, title, year, genre), \\ Director(ID, dir), Actor(ID, dir)$$

$$V(I, D, A) :- Director(I, D), Actor(I, A)$$

Need to specialize the view first:

$$V'(I, D, D) :- Director(I, D), Actor(I, D)$$

MCD:

mapping:

$$ID \rightarrow I$$

$$dir \rightarrow D$$

covered subgoals of Q: *{2,3}*

MCDs: Detail 2

ILLINOIS INSTITUTE OF TECHNOLOGY

$$Q(title, year, dir) :- Movie(ID, title, year, genre), \\ Director(ID, dir), Actor(ID, dir)$$

$$V(I, D, D) :- Director(I, D), Actor(I, D), \\ Movie(I, T, Y, G)$$

Note: the third subgoal of the view is *not* included in the MCD.

MCD:

mapping:

$$ID \rightarrow I$$

$$dir \rightarrow D$$

covered subgoals of Q *still:* {2,3}

Inverse-Rules Algorithm

- A "logical" approach to AQUV
- Produces maximally-contained rewriting in polynomial time
  - To check whether the rewriting is equivalent to the query, you still need a containment check.
- Conceptually simple and elegant
  - Depending on your comfort with Skolem functions…

Inverse Rules by Example

Given the following view:

$$V_7(I,T,Y,G) :- Movie(I,T,Y,G), Director(I,D), Actor(I,D)$$

And the following tuple in V_7:

V_7(79,Manhattan,1979,Comedy)

Then we can infer the tuple:

Movie(79,Manhattan,1979,Comedy)

Hence, the following 'rule' is sound:

IN_1: *Movie(I,T,Y,G) :-* V_7*(I,T,Y,G)*

Skolem Functions

$$V_7(I,T,Y,G) :- Movie(I,T,Y,G), Director(I,D), Actor(I,D)$$

Now suppose we have the tuple

V_7(79,Manhattan,1979,Comedy)

Then we can infer that there exists *some* director. Hence, the following rules hold (note that they both use the same Skolem function):

IN_2: *Director(I,f_1(I,T,Y,G)):- V_7(I,T,Y,G)*

IN_3: *Actor(I,f_1(I,T,Y,G)):- V_7(I,T,Y,G)*

Inverse Rules in General
Rewriting = Inverse Rules + Query

ILLINOIS INSTITUTE OF TECHNOLOGY

$$Q_2(title, year, genre) :- Movie(ID, title, year, genre)$$

Given Q2, the rewriting would include:

IN_1, IN_2, IN_3, Q_2.

Given input: V_7(79,Manhattan,1979,Comedy)

Inverse rules produce:

Movie(79,Manhattan,1979,Comedy)

Director(79,*f_1(79,Manhattan,1979,Comedy)*)

Actor(79,*f_1(79,Manhattan,1979,Comedy)*)

Movie(Manhattan,1979,Comedy)

(the last tuple is produced by applying Q_2).

Comparing Algorithms

- Bucket algorithm:
  - Good if there are many interpreted predicates
  - Requires containment check. Cartesian product can be big
- MiniCon:
  - Good at detecting interactions between subgoals

Algorithm Comparison (Continued)

- Inverse-rules algorithm:
  - Conceptually clean
  - Can be used in other contexts (see later)
  - But may produce inefficient rewritings because it "undoes" the joins in the views (see next slide)
- Experiments show MiniCon is most efficient.
- Even faster:

Konstantinidis, G. and Ambite, J.L, ***Scalable query rewriting: a graph-based approach. SIGMOD '11***

Inverse Rules Inefficiency Example

ILLINOIS INSTITUTE OF TECHNOLOGY

$$Query \quad and \quad view:$$

$$Q(X,Y) :- e_1(X,Z), e_2(Z,Y)$$

$$V(A,B) :- e_1(A,C), e_2(C,B)$$

$$Inverse \quad rules:$$

$$e_1(A, f_1(A,B)) :- V(A,B)$$

$$e_2(f_1(A,B), B) :- V(A,B)$$

Now we need to re-compute the join…

View-Based Query Answering

- Maximally-contained rewritings are parameterized by query language.
- More general question:
  - Given a set of view definitions, view instances and a query, what are **all** the answers we can find?
- We introduce **certain answers** as a mechanism for providing a formal answer.

View Instances = Possible DB's

Consider the two views:

$$V_8(dir) :- Movie(ID, dir, actor)$$

$$V_9(actor) :- Movie(ID, dir, actor)$$

And suppose the extensions of the views are:

V_8: {Allen, Copolla}
V_9: {Keaton, Pacino}

Possible Databases

There are multiple databases that satisfy the above view definitions: (we ignore the first argument of *Movie* below)

DB1. {(Allen, Keaton), (Coppola, Pacino)}
DB2. {(Allen, Pacino), (Coppola, Keaton)}

If we ask whether Allen directed a movie in which Keaton acted, we can't be sure.

Certain answers are those true in *all* databases that are consistent with the views and their extensions.

Certain Answers: Formal Definition [Open-world Assumption]

- Given:
  - Views: $V_1,\dots,V_n$
  - View extensions $v_1,\dots v_n$
  - A query Q
- A tuple t is a certain answer to Q under the open-world assumption if $t \in Q(D)$ for all databases D such that:
  - $V_i(D) \subseteq v_i$ *for all* i.

Certain Answers [Closed-world Assumption]

ILLINOIS INSTITUTE OF TECHNOLOGY

- Given:
  - Views: $V_1,\ldots,V_n$
  - View extensions $v_1,\ldots v_n$
  - A query Q
- A tuple t is a certain answer to Q under the open-world assumption if $t \in Q(D)$ for all databases D such that:
  - $V_i(D) = v_i$ *for all* i.

Certain Answers: Example

$$V_8(dir) :- Director(ID, dir)$$

$$V_9(actor) :- Actor(ID, actor)$$

V8: {Allen}

V9: {Keaton}

$$Q(dir, actor) :- Director(ID, dir), Actor(ID, actor)$$

Under closed-world assumption:

single DB possible $\Rightarrow$ (Allen, Keaton)

Under open-world assumption:

no certain answers.

The Good News

ILLINOIS INSTITUTE OF TECHNOLOGY

- The MiniCon and Inverse-rules algorithms produce all certain answers
  - Assuming no interpreted predicates in the query (ok to have them in the views)
  - Under open-world assumption
  - Corollary: they produce a maximally-contained rewriting

In Other News…

- Under closed-world assumption finding all certain answers is co-NP hard!

Proof: *encode a graph - G = (V,E)*

$$v_1(X) :- color(X,Y) \qquad I(V_1) = V$$

$$v_2(Y) :- color(X,Y) \qquad I(V_2) = \{red, green, blue\}$$

$$v_3(X,Y) :- edge(X,Y) \qquad I(V_3) = E$$

$$q() :- edge(X,Y), color(X,Z), color(Y,Z)$$

q has a certain tuple iff G is not 3-colorable

Interpreted Predicates

- In the views: no problem (all results hold)
- In the query Q:
  - If the query contains interpreted predicates, finding all certain answers is co-NP-hard even under open-world assumption
  - Proof: reduction to CNF.

Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance