Logical Information Integration

Jose Luis Ambite
University of Southern California

Information Integration

- Information Integration: provide semantically *unified view* to collection of data stored in *multiple, autonomous*, and *heterogeneous sources*
- Unified view achieved by defining global schema
- Two approaches to realize global schema:
  - Data Exchange: materialized database
  - Data Integration: answer user queries over global schema at runtime, by rewriting and executing queries over sources

[based on slide by Lenzerini]

Information Integration Challenges

- Syntactic (Access/Format) heterogeneity:
  - Structured Sources: DBMS → ODBC, JDBC
  - Semistructured Sources: HTML, text, pdf, XML
    → Wrappers, Information Extraction
  - Web services → XML, SOAP, WSDL, REST
  - Geospatial Data → Maps, Images, Vector Data
- Semantic heterogeneity
  - Schema → Describe sources in common domain schema
  - Data → Record Linkage
- Scalability:
  - Mediation
  - Record Linkage
  - Efficient Query Execution

Schema Heterogeneity

- Schema heterogeneity is a fact of life
  - Whenever schemas are designed by different people/organizations, they will be different, even if they model the *same* domain!
- The goal of schema mappings is to reconcile schema heterogeneity
  - Mostly between the mediated schema and the schema of the data sources.

[based on slide by Doan, Halevy, Ives]

Schema Heterogeneity by Example

Mediated Schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Sources

S1
Movie(title, director, year, genre)
Actor(AID, firstName, lastName, nationality, yearofBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2
Cinemas(place, movie, start)

S3
NYCCinemas(name, title, startTime)

S4
Reviews(title, date, grade, review)

S5
MovieGenres(title, genre)

S6
MovieDirectors(title, dir)

S7
MovieYears(title, year)

[based on slide by Doan, Halevy, Ives]

Table and Attribute Names

Mediated Schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Table and attribute names

Sources

S1
Movie(title, director, year, genre)
Actor(AID, firstName, lastName, nationality, yearofBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2
Cinemas(place, movie, start)

S3
NYCCinemas(name, title, startTime)

S4
Reviews(title, date, grade, review)

S5
MovieGenres(title, genre)

S6
MovieDirectors(title, dir)

S7
MovieYears(title, year)

[based on slide by Doan, Halevy, Ives]

Organization of Schema

Mediated Schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Different tabular organization

Sources

S1
Movie(title, director, year, genre)
Actor(AID, firstName, lastName, nationality, yearofBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2
Cinemas(place, movie, start)

S3
NYCCinemas(name, title, startTime)

S4
Reviews(title, date, grade, review)

S5
MovieGenres(title, genre)

S6
MovieDirectors(title, dir)

S7
MovieYears(title, year)

[based on slide by Doan, Halevy, Ives]

Schema Coverage

Mediated Schema

Movie: title, director, year, genre
Actors: title, name
Plays: movie, location, startTime
Reviews: title, rating, description

Sources

Different coverage and detail

S1
Movie(title, director, year, genre)
Actor(AID, firstName, lastName, nationality, yearofBirth)
ActorPlays(AID, MID)
MovieDetails(MID, director, genre, year)

S2
Cinemas(place, movie, start)

S3
NYCCinemas(name, title, startTime)

S4
Reviews(title, date, grade, review)

S5
MovieGenres(title, genre)

S6
MovieDirectors(title, dir)

S7
MovieYears(title, year)

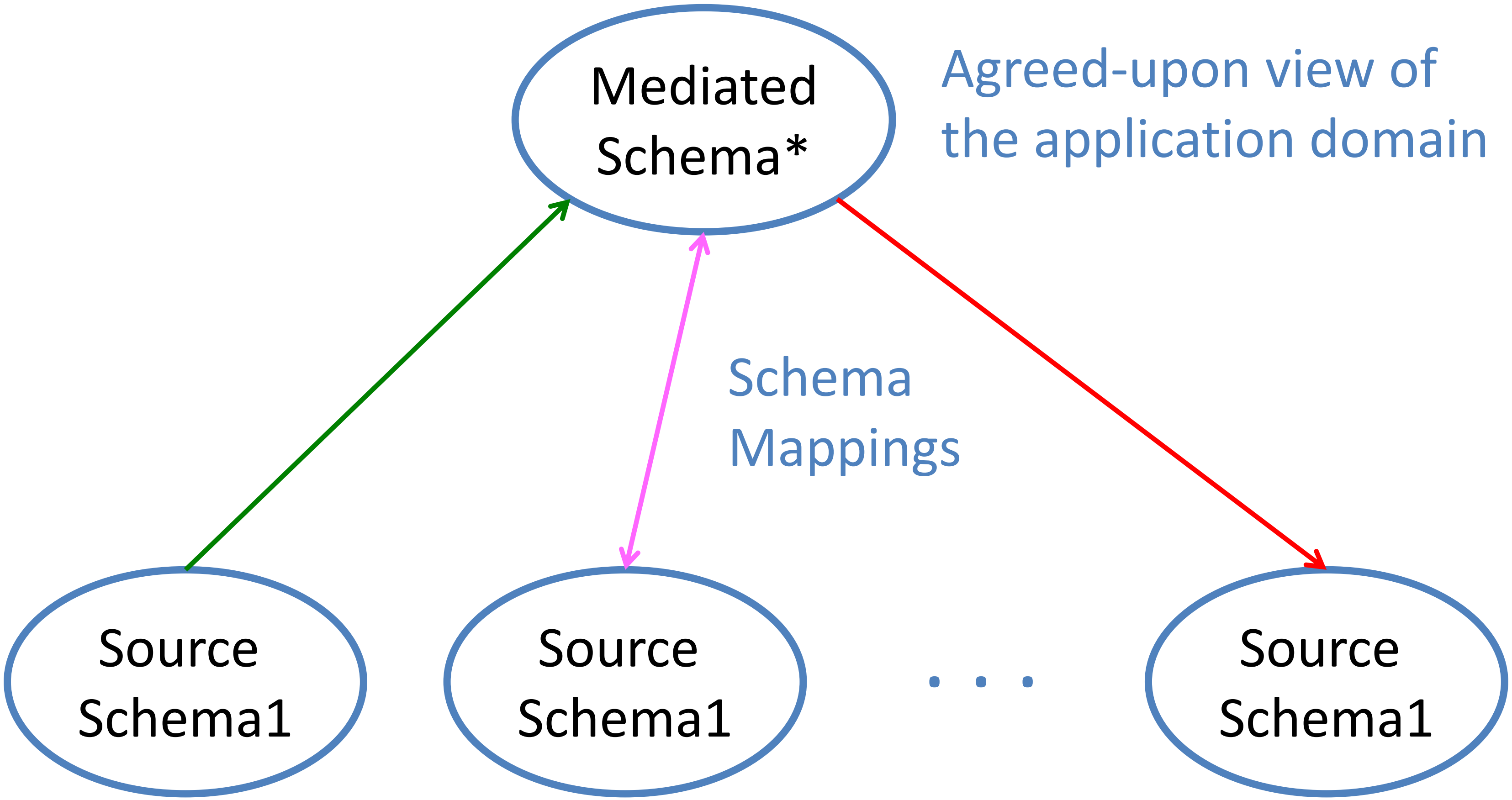
[based on slide by Doan, Halevy, Ives]

Semantic Heterogeneity Summary

- Differences in:
  - Naming of schema elements
  - Organization of predicates (tables)
  - Coverage and detail of schema
  - Objects/Data values (IBM vs. International Business Machines) … [Record Linkage]
- Reason: different perspectives
  - Schemas probably designed for different applications/contexts

[based on slide by Doan, Halevy, Ives]

Integration: Schema Mappings



*Mediated/global/domain/target schema/ontology

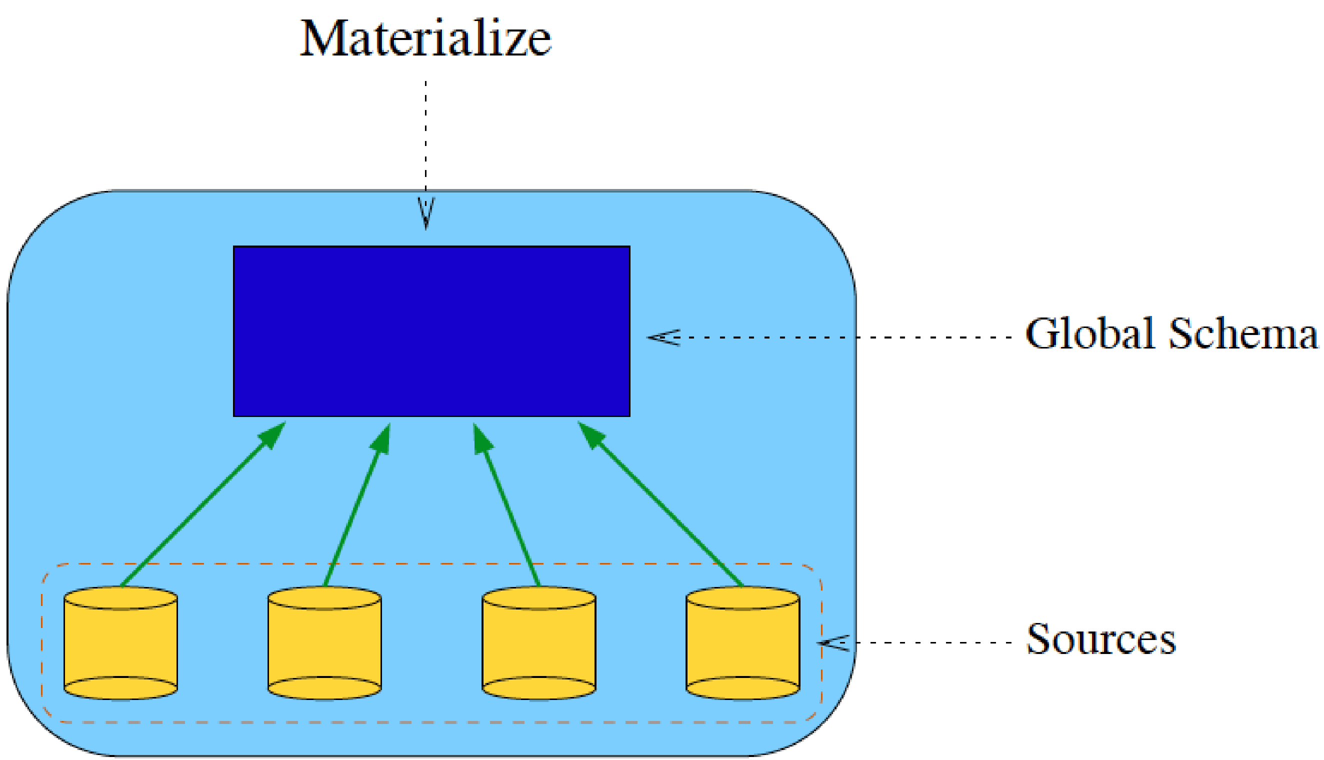
Dimensions of Data Integration

- Query / Update
- Architecture
  - Warehouse/ETL
  - Virtual Data Integration
- Schema Mappings
  - Global-as-View (GAV)
  - Local-as-View (LAV)
  - GLAV (st-tgds)
- Language for mappings and queries
  - conjunctive queries (CQ)
  - union of CQs (UCQ)
  - first-order logic ($\wedge,\vee,\neg$)
  - Datalog (recursion)
  - XML, RDF, OWL (description logics) …
- Source data models
  - Relational, XML, RDF, …
- Global Schema Constraints
  - Inclusion, functional, ontologies, …
- Source Capabilities:
  - Input/Output restrictions
- Source type
  - RDBMS, triplestore, XML DB, web services, formatted files, …

Data Exchange

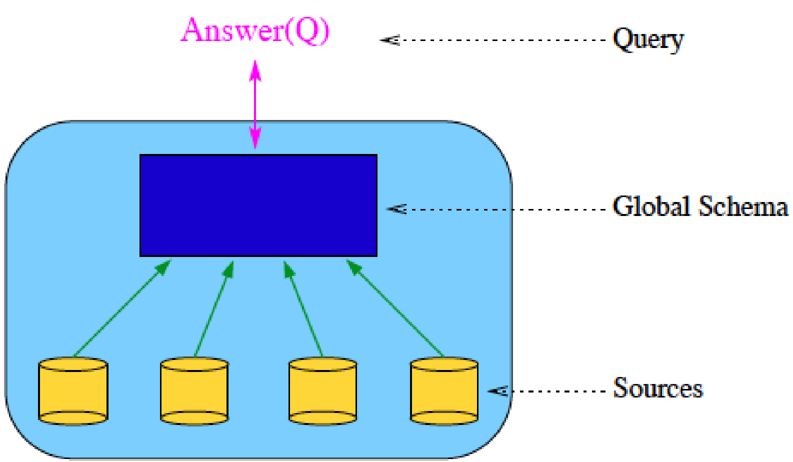
- materialization of the global view
- allows for query answering without accessing the sources

data warehousing



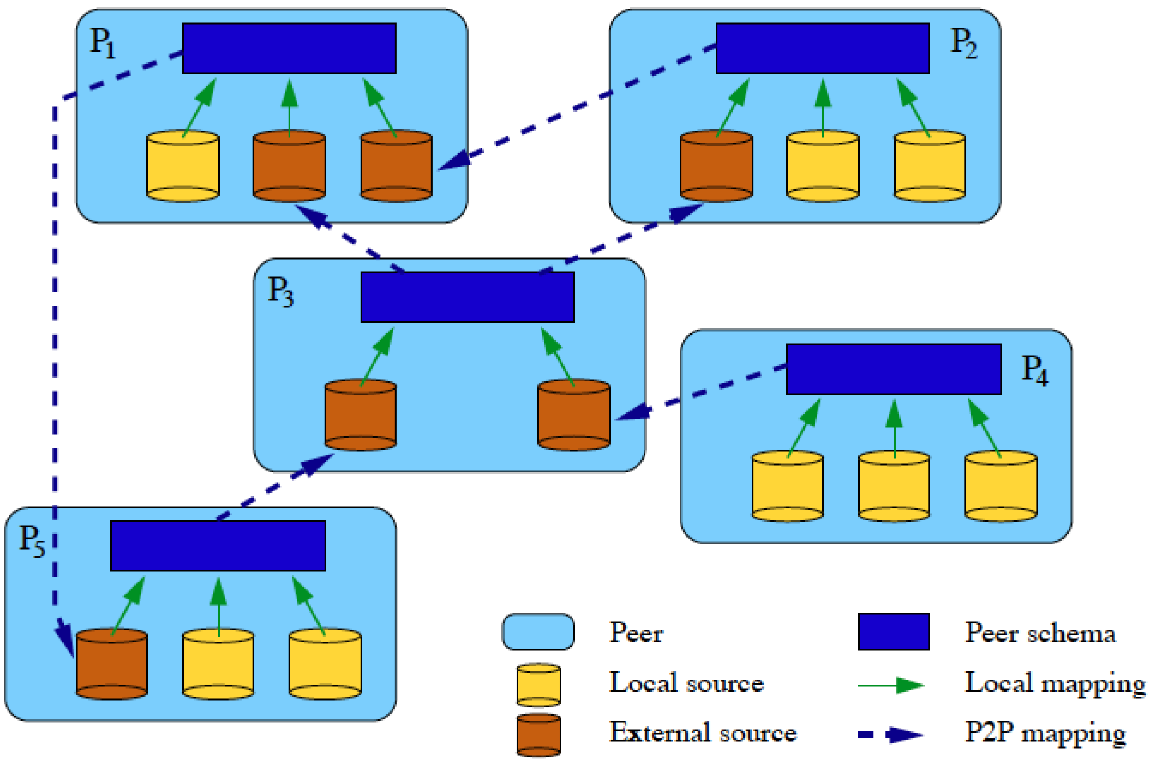
Virtual Data Integration

- Queries are expressed over a **global schema** (a.k.a. mediated schema, enterprise model, domain model, target schema/ontology/model)
- Data are stored in a set of sources
- **Wrappers** access the sources (provide a view in a uniform data model of the data stored in the sources)
- **Mediators** combine answers coming from wrappers and/or other mediators



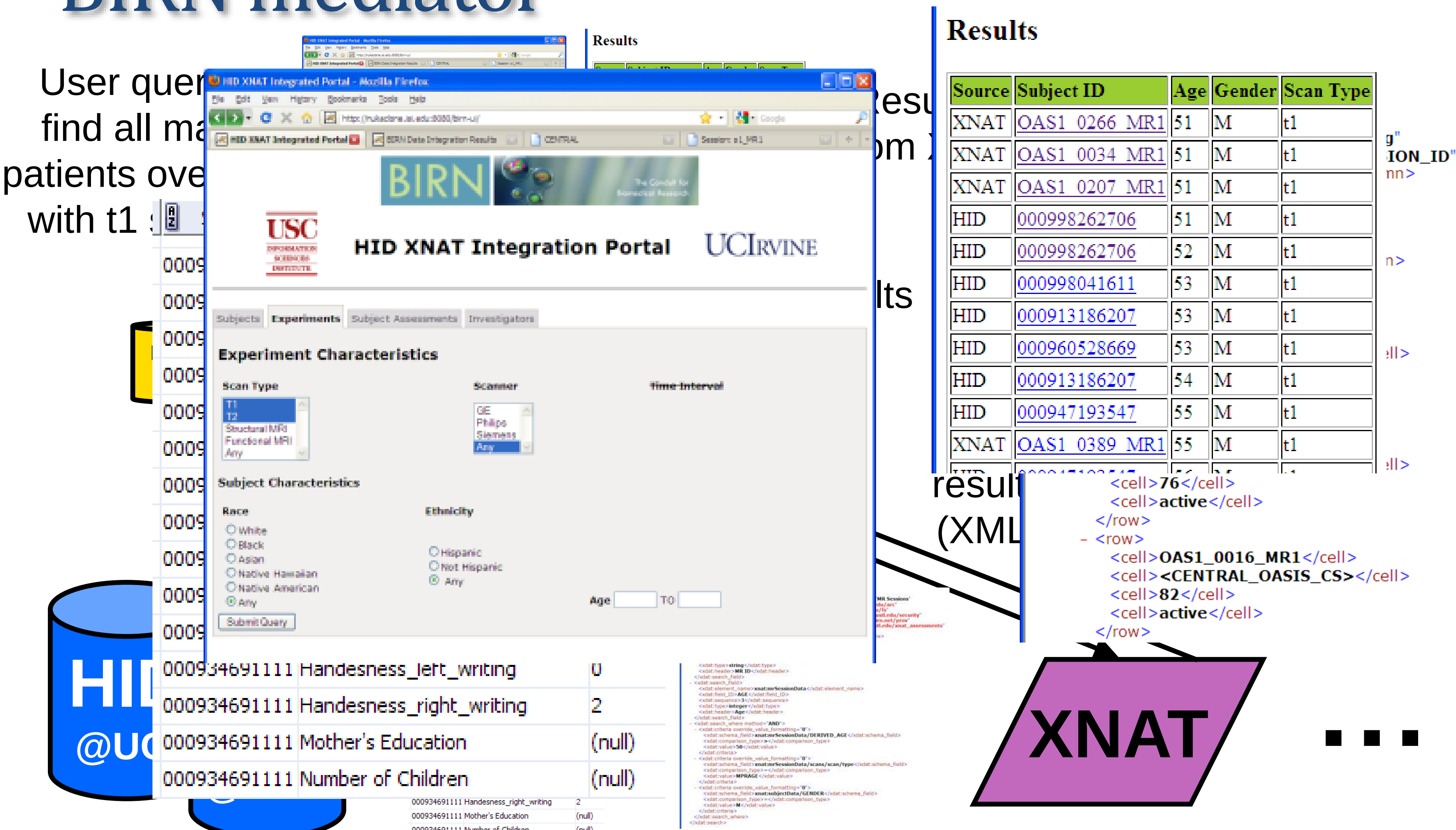
Peer-to-Peer Data Integration

- several peers
- each peer with local and external sources
- queries over one peer



[Ashish, Ambite, et al. Frontiers in Neuroinformatics 4:118, 2010]

Virtual Data Integration: BIRN mediator



Human **I**maging **D**atabase(s)
Oracle DB

EXtensible **N**euroimaging **A**rchive **T**oolkit
Web service API

Query Rewriting (aka query reformulation)

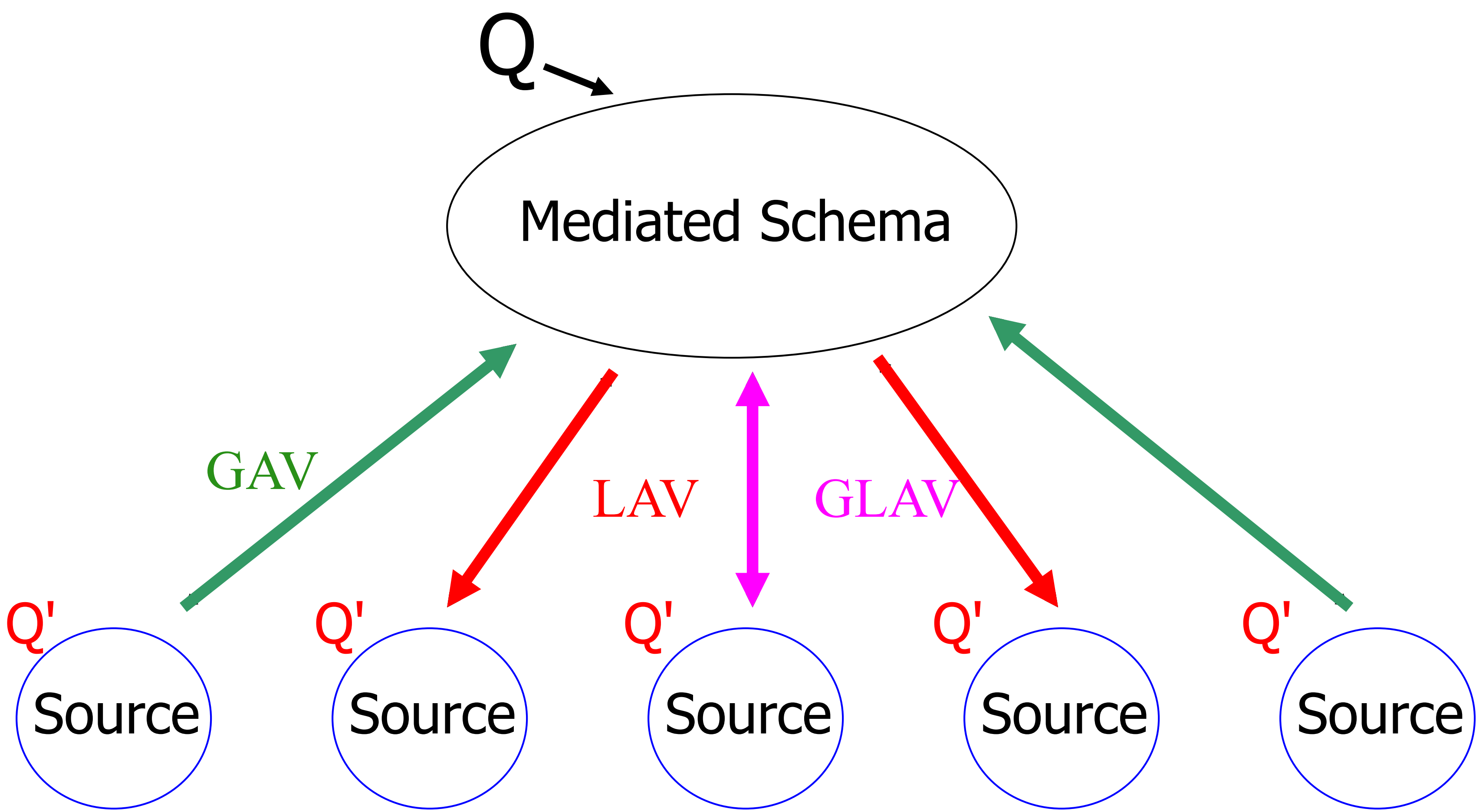
- Problem: rewrite user query expressed in mediated schema into query expressed in source schemas
- Given
  - Query Q in terms of the mediated-schema relations, and
  - Schema mappings (source descriptions)
  - (Mediated schema constraints)
- Find
  - Query Q' that uses only the source relations, such that
    - Q' provides only correct answers to Q, (i.e, $Q' \models Q$, $Q' \subseteq Q$), and
    - Q' provides all possible answers to Q given the sources

Note: I'll use the terms "schema mappings" and "source descriptions" interchangeably

Desiderata for Source Descriptions

- Expressive power: distinguish between sources with closely related data. Hence, be able to prune access to irrelevant sources.
- Easy addition: easy to add new data sources.
- Reformulation: be able to reformulate a user query into a query over the sources efficiently and effectively.
- Nonlossy: be able to handle all queries that can be answered by directly accessing the sources

Languages for Schema Mapping



[based on slide by Doan, Halevy, Ives]

Schema Mappings

- Global-as-View (GAV): $\Phi s(X,Y) \rightarrow g(X)$
  - Mediator relation defined as a view over source relations
  - Ex: TSIMMIS (Stanford), HERMES (Maryland), SIMS, BIRN (USC)
- Local-as-View (LAV): $s(X) \rightarrow \exists Y\ \Phi g(X,Y)$
  - Source relation defined as view over mediator relations
  - Ex: Information Manifold (AT&T), Tukwila (UW), InfoMaster (Stanford), Prometheus, BIRN Mediator/GQR (USC)
- General (GLAV): $\Phi s(X,Y) \rightarrow \exists Z\ \Psi g(X,Z)$
  - BIRN Mediator (USC)

View ~ named query ~ logical formula

Global-as-View (GAV) Schema Mapping

Each mediator relation is defined as a view over source relations: $\Phi_s(X,Y) \rightarrow g(X)$

review(Title,Year,Review) $\leftarrow$ s1(ID,Title,Director,Year) ^
s3(ID,Review)

```
CREATE VIEW review AS
SELECT Title, Year, Review
FROM s1, s3
WHERE s1.ID = s3.ID
```

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Domain Query: *Find reviews after 1997*

q(T,R) ← review(T,Y,R), Y >= 1997

rewritten to

Source Query:

q'(T,R) ← s1(I,T,D,Y), s3(I,R), Y >= 1997

using GAV schema mapping:

review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Global-as-View (GAV)

Each mediator relation is defined as a view over source relations: $\Phi_s(X,Y) \rightarrow g(X)$

movie(Title,Year,Director) $\leftarrow$ s1(ID,Title,Director,Year)

movie(Title,Year,Director) $\leftarrow$ s2(Title,Director, Studio,Year)

review(Title,Year,Review) $\leftarrow$ s1(ID,Title,Director,Year) ^ s3(ID,Review)

Assume s1 and s3 are in the same database db1, and s2 is in a different source

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
   s1(I',T,D,Y), s3(I',R)

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
   s1(I,T,D,Y), s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I'

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
~~s1(I,T,D,Y)~~, s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I' and 2nd s1 is redundant

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
   s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I' and 2nd s1 is redundant

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
   s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I' and 2nd s1 is redundant

2. q'(T,R) ← s2(T,D,S,Y), Y >= 1997,
   s1(I',T,D,Y), s3(I',R)

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
   s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I' and 2nd s1 is redundant

2. q'(T,R) ← s2(T,D,S,Y), Y >= 1997,
   s1(I,T,D,Y), s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I'

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

1. q'(T,R) ← s1(I,T,'Scott',Y), Y >= 1997,
   s3(I,R)

Assume (T,Y) and (I) are keys of s1, then I=I' and 2nd s1 is redundant

2. ~~q'(T,R) ← s2(T,D,S,Y), Y >= 1997,
   s1(I,T,D,Y), s3(I,R)~~

Assume (T,Y) and (I) are keys of s1, then I=I'

If s1 and s3 have same movies, then q2 is redundant wrt q1

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

Query Reformulation in GAV

Query reformulation = rule unfolding + simplification

Query: *Find reviews for movies directed by Scott after 1997*

q(T,R) ← movie(T,Y, 'Scott'), review(T,Y,R), Y >= 1997

q'(T,R) ← s1(I,T,'Scott',Y), s3(I,R), Y >= 1997,

movie(Title,Year,Director) ← s1(ID,Title,Director,Year)
movie(Title,Year,Director) ← s2(Title,Director, Studio,Year)
review(Title,Year,Review) ← s1(ID,Title,Director,Year) ^ s3(ID,Review)

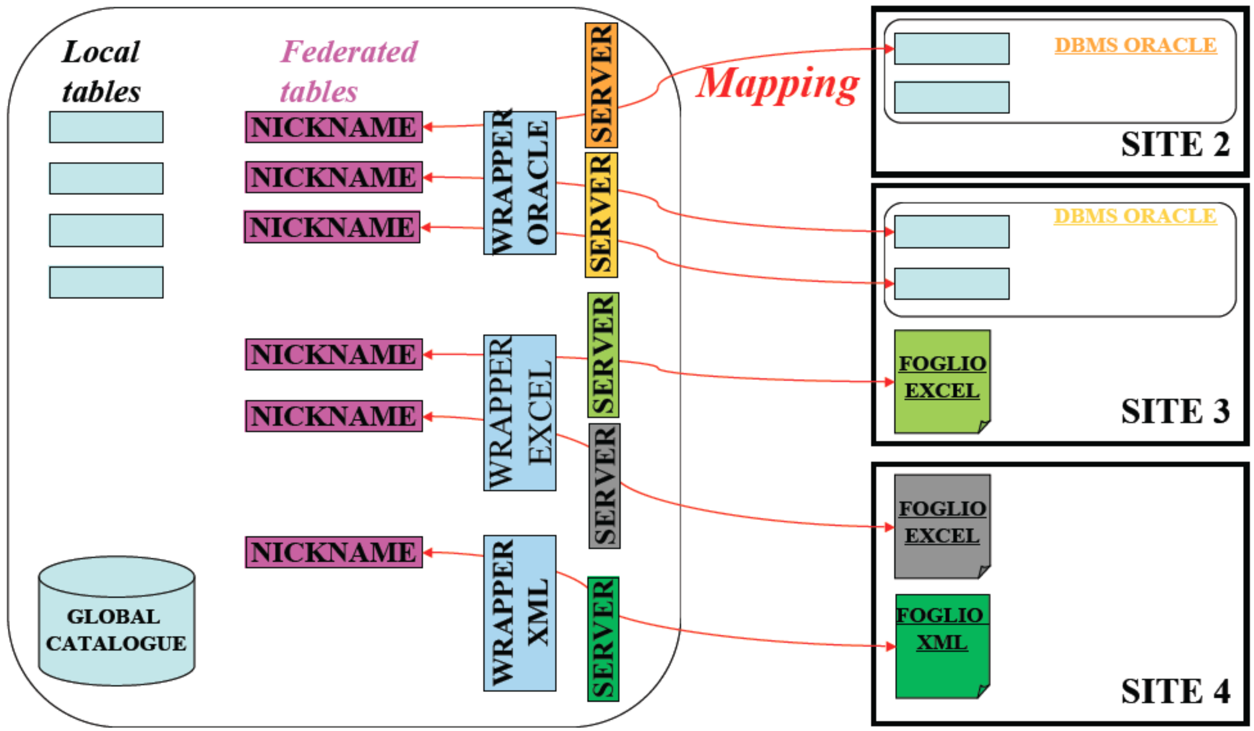
Distributed/Federated Queries vs. Integration

- Some systems allow to query remote databases
  - Ex:

IBM

Information

Integrator*



- However, If we just expose the source schemas, the user would quickly get overloaded
- For integration, we need to create global schema

(*) IBM marketing changes the name of the system quite often, today it is called InfoSphere Federation Server: http://www-01.ibm.com/software/data/infosphere/federation-server/

Local-as-View (LAV) Schema Mappings

- Each source relation is defined as a view over mediator relations: $s(X) \rightarrow \exists Y\ \Phi_g(X,Y)$

V1(title, year, director) $\overset{\subseteq}{\rightarrow}$ Movie(title,year,director,genre) ^ American(director) ^ year ≥1960 ^ genre = 'Comedy'

V2(title, review) $\overset{\subseteq}{\rightarrow}$ ∃year, director, genre Movie(title,year,director,genre) ^ year≥1990 ^ hasReview(title, review)

Open world! Tuples in sources guaranteed to satisfy the definitions, but sources don't have all possible tuples

Existentials! Invent values! Skolem functions, aka labelled nulls

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'), year ≥1950, hasReview(title,review)

s1(title, year, director) $\rightarrow$ Movie(title,year,director,genre) ^ American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) $\rightarrow$ Movie(title,year,director,genre) ^ year≥1990 ^ hasReview(title, review)

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'),
year ≥1950, hasReview(title,review)

Is s2 enough?

Only s2 provides hasReview

Reformulated query:

q'(title,review) :- s2(title,review)

s1(title, year, director) → Movie(title,year,director,genre) ^
American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) → Movie(title,year,director,genre) ^ year≥1990 ^
hasReview(title, review)

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'), year ≥1950, hasReview(title,review)

Is s2 enough? *No!*
Need s1 to ensure Genre='Comedy'

Reformulated query:

q'(title,review) :- s1(title,year,director), s2(title,review)

s1(title, year, director) → Movie(title,year,director,genre) ^ American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) → Movie(title,year,director,genre) ^ year≥1990 ^ hasReview(title, review)

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'),
year ≥1950, hasReview(title,review)

Reformulated query:

Are we done?
What about year ≥ 1950 and Genre = 'Comedy'?

q'(title,review) :- s1(title,year,director), s2(title,review)

s1(title, year, director) → Movie(title,year,director,genre) ^
American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) → Movie(title,year,director,genre) ^ year≥1990 ^
hasReview(title, review)

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'),
year ≥1950, hasReview(title,review)

Are we done? *Yes!*
What about year ≥ 1950 and Genre = 'Comedy'?
Constraints are implied

Reformulated query:

q'(title,review) :- s1(title,year,director), s2(title,review)

s1(title, year, director) → Movie(title,year,director,genre) ^
American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) → Movie(title,year,director,genre) ^ year≥1990 ^
hasReview(title, review)

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'),
    year ≥1950, hasReview(title,review)

Does q' answers what q asks?

Reformulated query:

q'(title,review) :- s1(title,year,director), s2(title,review)

s1(title, year, director) → Movie(title,year,director,genre) ^
    American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) → Movie(title,year,director,genre) ^ year≥1990 ^
    hasReview(title, review)

Query Reformulation in LAV

Query: *Reviews for comedies produced after 1950*

q(title,review) :- Movie(title,year,director,'Comedy'),
year ≥1950, hasReview(title,review)

Does q' answers what q asks?
Not quite, it gives a subset ...

$q' \subseteq q$

Reformulated query:

q'(title,review) :- s1(title,year,director), s2(title,review)

... but it's the best we can do given the available sources

s1(title, year, director) $\rightarrow$ Movie(title,year,director,genre) ^
American(director) ^ year ≥1960 ^ genre = 'Comedy'

s2 (title, review) $\rightarrow$ Movie(title,year,director,genre) ^ year≥1990 ^
hasReview(title, review)

Answering queries using views

Given query q and view definitions V = {V1…Vn}

- q' is a Maximally-Contained Rewriting of q using V if:
  - q' refers only to views in V, and
  - $q' \subseteq q$, and
  - there is no rewriting q1, such that $q' \subseteq q1 \subseteq q$ and $q1 \neq q$
- q' is an Equivalent Rewriting of q using V if:
  - q' refers only to views in V, and
  - $q' = q$

GAV vs. LAV

$\Phi s(X,Y) \rightarrow g(X)$

- Not modular
  - adding new sources may require changes to all integration rules
- Query reformulation is easy
  - reduces to view unfolding
  - (though query simplification not so easy)
- Best when
  - Few, stable, data sources
  - well-known to the mediator (e.g. corporate integration)

$s(X) \rightarrow \exists Y\ \Phi g(X,Y)$

- Modular
  - adding new sources is easier, does not affect previous rules
- Expressive
  - power of entire query language available to describe sources
- Query reformulation is hard
  - answering queries using views
- Best when
  - Many, relatively unknown data sources
  - possibility of addition/deletion of sources

[Levy+1996]

Query Reformulation in LAV: The Bucket Algorithm

- Given: user query q, source descriptions {Vi}
- Find relevant sources (fill buckets)
  - For each predicate g in query q
    - Find Vj that contains predicate g
    - Check that constraints in Vj are compatible with q
- Combine source relations {Vj} from each bucket into a conjunctive query q' and check for containment (q' $\subseteq$ q)

The Bucket Algorithm: Example

- Global Schema

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

  Actor(ID,actor) Revenues(ID, Amount)

- LAV Schema Mappings

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M

V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)

V3(I, A) → Revenues(I,A), A < $50M

V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

- User Query

q(ID,Dir) ← Movie(ID,Title,Year,Genre), Revenues(ID, Amount),
Director(ID,Dir), Amount > $100M

1. Filling the Buckets

A goal g_v in a view v is relevant to a goal g_q in a query q if:

1. The goals are on the same predicate
2. The interpreted predicates on variables of g_v, after appropriate variable substitutions, are satisfiable
3. If g_q includes a head variable of q, then the corresponding variable in g_v is also in the head of v

1. Filling the Buckets

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > \$200M

V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)

V3(I, A) → Revenues(I,A), A < \$50M

V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > \$100M

1. Filling the Buckets

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M

V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)

V3(I, A) → Revenues(I,A), A < $50M

V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)

V2(ID,A')

V4(ID,D',Year)

1. Filling the Buckets

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > \$200M

V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)

V3(I, A) → Revenues(I,A), A < \$50M

V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > \$100M

V1(ID,Year)	V1(ID,Y’)
V2(ID,A’)	V2(ID,Amt)
V4(ID,D’,Year)	

1. Filling the Buckets

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M

V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)

V3(I, A) → Revenues(I,A), A < $50M

V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y’)	V4(ID,Dir,Y’’)
V2(ID,A’)	V2(ID,Amt)	
V4(ID,D’,Year)		

1. Filling the Buckets

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M

V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)

V3(I, A) → Revenues(I,A), A < $50M

V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y’)	V4(ID,Dir,Y’’)
V2(ID,A’)	V2(ID,Amt)	
V4(ID,D’,Year)		

2. Checking Containment

- Construct a rewritten query q’ by choosing one view from each bucket
- Check for containment: $q' \subseteq q$
- Collect all such rewritten queries {q’}, then maximally-contained rewriting is the union

2. Checking Containment

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y')	V4(ID,Dir,Y'')
V2(ID,A')	V2(ID,Amt)	
V4(ID,D',Year)		

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M
V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)
V3(I, A) → Revenues(I,A), A < $50M
V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

2. Checking Containment

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y')	V4(ID,Dir,Y'')
V2(ID,A')	V2(ID,Amt)	
V4(ID,D',Year)		

V1(ID,Year), V1(ID,Y'), V4(ID,Dir,Y'') →

Movie(ID,T,Year,G), Revenues(ID,A), ID > 5000, A > $200M,

Movie(ID,T',Y',G'), Revenues(ID,A'), ID > 5000, A' > $200M,

Movie(ID,T'',Y'',G''), Director(ID,Dir), ID < 3000,

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M
V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)
V3(I, A) → Revenues(I,A), A < $50M
V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

2. Checking Containment

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y')	V4(ID,Dir,Y'')
V2(ID,A')	V2(ID,Amt)	
V4(ID,D',Year)		

V1(ID,Year), V1(ID,Y'), V4(ID,Dir,Y'') →

Movie(ID,T,Year,G), Revenues(ID,A), ID > 5000, A > $200M,

Movie(ID,T',Y',G'), Revenues(ID,A'), ID > 5000, A' > $200M,

Movie(ID,T'',Y'',G''), Director(ID,Dir), ID < 3000,

Inconsistent!

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M
V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)
V3(I, A) → Revenues(I,A), A < $50M
V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

2. Checking Containment

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y')	V4(ID,Dir,Y'')
V2(ID,A')	V2(ID,Amt)	
V4(ID,D',Year)		

V2(ID,A'), V2(ID,Amt), V4(ID,Dir,Y''), *Amt > $100M* →

V2(ID,Amt), V4(ID,Dir,Y''), *Amt > $100M* →

Movie(ID,T',Y',G'), Revenues(ID,Amt),

Movie(ID,T'',Y'',G''), Director(ID,Dir), ID < 3000, *Amt > $100M* →

Movie(ID,T',Y',G'), Revenues(ID,Amt), Director(ID,Dir), ID < 3000, *Amt > $100M* →

q(ID,Dir)

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M
V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)
V3(I, A) → Revenues(I,A), A < $50M
V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

2. Checking Containment

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year) V1(ID,Y') V4(ID,Dir,Y'')

V2(ID,A') V2(ID,Amt)

V4(ID,D',Year)

Only movies with ID < 3000

V2(ID,A'), V2(ID,Amt), V4(ID,Dir,Y''), *Amt > $100M* →

V2(ID,Amt), V4(ID,Dir,Y''), *Amt > $100M* →

Movie(ID,T',Y',G'), Revenues(ID,Amt),

Movie(ID,T'',Y'',G''), Director(ID,Dir), ID < 3000, *Amt > $100M* →

Movie(ID,T',Y',G'), Revenues(ID,Amt), Director(ID,Dir), ID < 3000, *Amt > $100M* →

$\subseteq$

q(ID,Dir)

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M
V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)
V3(I, A) → Revenues(I,A), A < $50M
V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

2. Checking Containment

q(ID,Dir) ←

Movie(ID,Title,Year,Genre), Revenues(ID, Amt), Director(ID,Dir), Amt > $100M

V1(ID,Year)	V1(ID,Y’)	V4(ID,Dir,Y’’)
V2(ID,A’)	V2(ID,Amt)	
V4(ID,D’,Year)		

V2(ID,A’), V2(ID,Amt), V4(ID,Dir,Y’’), *Amt > $100M* →

q’(ID, Amt) ← V2(ID,Amt), V4(ID,Dir,Y’’), *Amt > $100M* →

Movie(ID,T’,Y’,G’), Revenues(ID,Amt),

Movie(ID,T’’,Y’’,G’’), Director(ID,Dir), ID < 3000, *Amt > $100M* →

Movie(ID,T’,Y’,G’), Revenues(ID,Amt), Director(ID,Dir), ID < 3000, *Amt > $100M* →

$\subseteq$

q(ID,Dir)

V1(I, Y) → Movie(I,T,Y,G), Revenues(I,A), I > 5000, A > $200M
V2(I, A) → Movie(I,T,Y,G), Revenues(I,A)
V3(I, A) → Revenues(I,A), A < $50M
V4(I, D, Y) → Movie(I,T,Y,G), Director(I,D), I < 3000

[Duschka+1997]

Query Reformulation in LAV: Inverse-Rules Algorithm

- Idea: Construct an equivalent logic program which evaluation yields the answer to user query

- "Invert the rules": simply use standard logical manipulations (conversion to clausal form)

The Inverse-Rules Algorithm Conversion to Clausal Form

V1(dept,course) $\rightarrow$ Enrolled(student,dept) ^ Registered(student,course)

$$a \rightarrow b \equiv \neg a \vee b$$

$$\forall D,C\ [v1(D,C) \rightarrow \exists S\ [\ e(S,D) \wedge r(S,C)]]$$

$$\equiv \neg v1(D,C) \vee [e(f(D,C),D) \wedge r(f(D,C),C)]$$

$$\equiv [\neg v1(D,C) \vee e(f(D,C),D)] \wedge [\neg v1(D,C) \vee r(f(D,C),C)]$$

$$\equiv [v1(D,C) \rightarrow e(f(D,C),D)] \wedge [v1(D,C) \rightarrow r(f(D,C),C)]$$

$$\equiv$$

$$e(f(D,C),D) \leftarrow v1(D,C)$$

$$r(f(D,C),C) \leftarrow v1(D,C)$$

This is essentially converting the rules to clausal form. See for example:
http://logic.stanford.edu/classes/cs157/2010/lectures/lecture09.pdf

The Inverse-Rules Algorithm: Example

$q(D) \leftarrow$ Enrolled(S,D) ^ Registered(S,'DB')
$v1(D,C) \rightarrow$ Enrolled(S,D) ^ Registered(S,C)

$q(D) \leftarrow$ Enrolled(S,D) ^ Registered(S,'DB')
Enrolled(f(D,C),D) $\leftarrow$ v1(D,C)
Registered(f(D,C),C) $\leftarrow$ v1(D,C)

$q(D) \leftarrow$ v1(D,'DB')

Ext(v1) = {('CS', 'DB'), ('EE', 'DB'), ('CS', 'AI')}
Ext(q) = {('CS'), ('EE')}

GLAV mappings

$\square s(X,Y)\ \square\ \exists Z\ \square g(X,Z)$

Example:

s1(id, title, year, director) ^ s2(id, review) $\rightarrow$
Movie(title,year,director,genre) ^ American(director) ^ year ≥1960 ^ genre = 'Comedy'^ year≥1990 ^ hasReview(title, review)

Note we did not model the `id` attribute (i.e., it's not in the consequent of the rule), perhaps it's an internal id to the sources that we don't care about

GLAV = LAV + GAV

A GLAV mapping $\Phi s(X,Y) \rightarrow \exists Z\ \Psi g(X,Z)$

can be converted to a LAV and a GAV mapping

- Just invent intermediate predicate v:
  - $\Phi s(X,Y) \rightarrow v(X)$ [GAV]
  - $v(X) \rightarrow \exists Z\ \Psi g(X,Z)$ [LAV]

So we can process GLAV rules by a combination of LAV and GAV query rewriting algorithms

Modeling Source Capabilities

Negative capabilities:

- A web-site (HTML form) or a web service may require certain inputs in order to provide outputs.
  - Ex: to output current temperature, Yahoo weather needs ZIP as input
- Need to consider only valid query execution plans
  - Consistent orderings of source calls

Positive capabilities:

- A source may be database (understands SQL)
- Need to decide the placement of operations according to capabilities

Problem: how to describe and exploit source capabilities

Negative Capabilities: Binding Patterns

Sources:

$AAAIdb^{f}(X) \rightarrow AAAIPapers(X)$

$CitationDB^{bf}(X,Y) \rightarrow Cites(X,Y)$

$AwardDB^{b}(X) \rightarrow AwardPaper(X)$

Query: find all the award winning papers:

$q(X) \leftarrow AwardPaper(X)$

Recursive Rewritings

$q(X) \leftarrow AwardPaper(X)$

- Problem: *Unbounded* union of conjunctive queries

$q1(X) \leftarrow AAAIdb(X), AwardDB(X)$

$q1(X) \leftarrow AAAIdb(X1), CitationDB(X1,X), AwardDB(X)$

…

$q1(X) \leftarrow AAAIdb(X1), CitationDB(X1,X2), \ldots, CitationDB(Xn,X), AwardDB(X)$

- Solution: Recursive Rewriting

$papers(X) \leftarrow AAAIdb(X)$

$papers(X) \leftarrow papers(Y), CitationDB(Y,X)$

$q'(X) \leftarrow papers(X), AwardDB(X)$

$AAAIdb^{f}(X) \rightarrow AAAIPapers(X)$
$CitationDB^{bf}(X,Y) \rightarrow Cites(X,Y)$
$AwardDB^{b}(X) \rightarrow AwardPaper(X)$

LAV query rewriting with sources with binding restrictions: Inverse-Rules Algorithm

Sources:

$AAAIdb^{f}(X) \rightarrow AAAIPapers(X)$

$CitationDB^{bf}(X,Y) \rightarrow Cites(X,Y)$

$AwardDB^{b}(X) \rightarrow AwardPaper(X)$

Query: find all the award winning papers:

$q(X) \leftarrow AwardPaper(X)$

Inverse-Rules Algorithm under binding restrictions: Domain predicate and rules

Invent a new predicate, *dom(X)*, that represents all objects in our domain

Use *dom* predicate to satisfy binding patterns

Given source descriptions (SD):

$AAAIdb^{f}(X) \rightarrow AAAIPapers(X)$

$CitationDB^{bf}(X,Y) \rightarrow Cites(X,Y)$

$AwardDB^{b}(X) \rightarrow AwardPaper(X)$

Modify SD by introducing *dom*

$AAAIdb(X) \rightarrow AAAIPapers(X)$

[no restrictions on AAAIdb, just remove binding annotation]

$dom(X) \wedge CitationDB(X,Y) \rightarrow Cites(X,Y)$

[we need to input objects as 1st argument of CitationDB]

$dom(X) \wedge AwardDB(X) \rightarrow AwardPaper(X)$

[we need to input objects to AwardDB to test them]

… and add *dom rules,* which define the behavior of *dom*

$AAAIdb(X) \rightarrow dom(X)$

[we can get objects from AAAIdb,since it has no restrictions]

$dom(X) \wedge CitationDB(X,Y) \rightarrow dom(Y)$

[if we input an object to CitationDB, we get another object out]

Inverse-Rules Algorithm under binding restrictions

Evaluating this datalog program generates the answers to query q

$q(X) \leftarrow AwardPaper(X)$ — query

$AAAIPapers(X) \leftarrow AAAIdb(X)$
$Cites(X,Y) \leftarrow \mathit{dom}(X) \wedge CitationDB(X,Y)$
$AwardPaper(X) \leftarrow \mathit{dom}(X) \wedge AwardDB(X)$
— inverse rules with *dom*

$\mathit{dom}(Y) \leftarrow \mathit{dom}(X) \wedge CitationDB(X,Y)$
$\mathit{dom}(X) \leftarrow AAAIdb(X)$
— *dom* rules

Inverse-Rules Algorithm under binding restrictions

Simplyfing the program:

q(X) $\leftarrow$ dom(X) ^ AwardDB(X)

dom(Y) $\leftarrow$ dom(X) ^ CitationDB(X,Y)

dom(X) $\leftarrow$ AAAIdb(X)