Query Optimization in the Deep Web

Andrea Calì Davide Martinenghi

Oxford-Man Institute, University of Oxford
Department of Information Systems and Computing, Brunel University
Dipartimento di Elettronica e Informazione, Politecnico di Milano

Roma Tre University
Rome, 10 June 2010

Outline

The deep Web

The deep Web

The deep Web

The deep Web

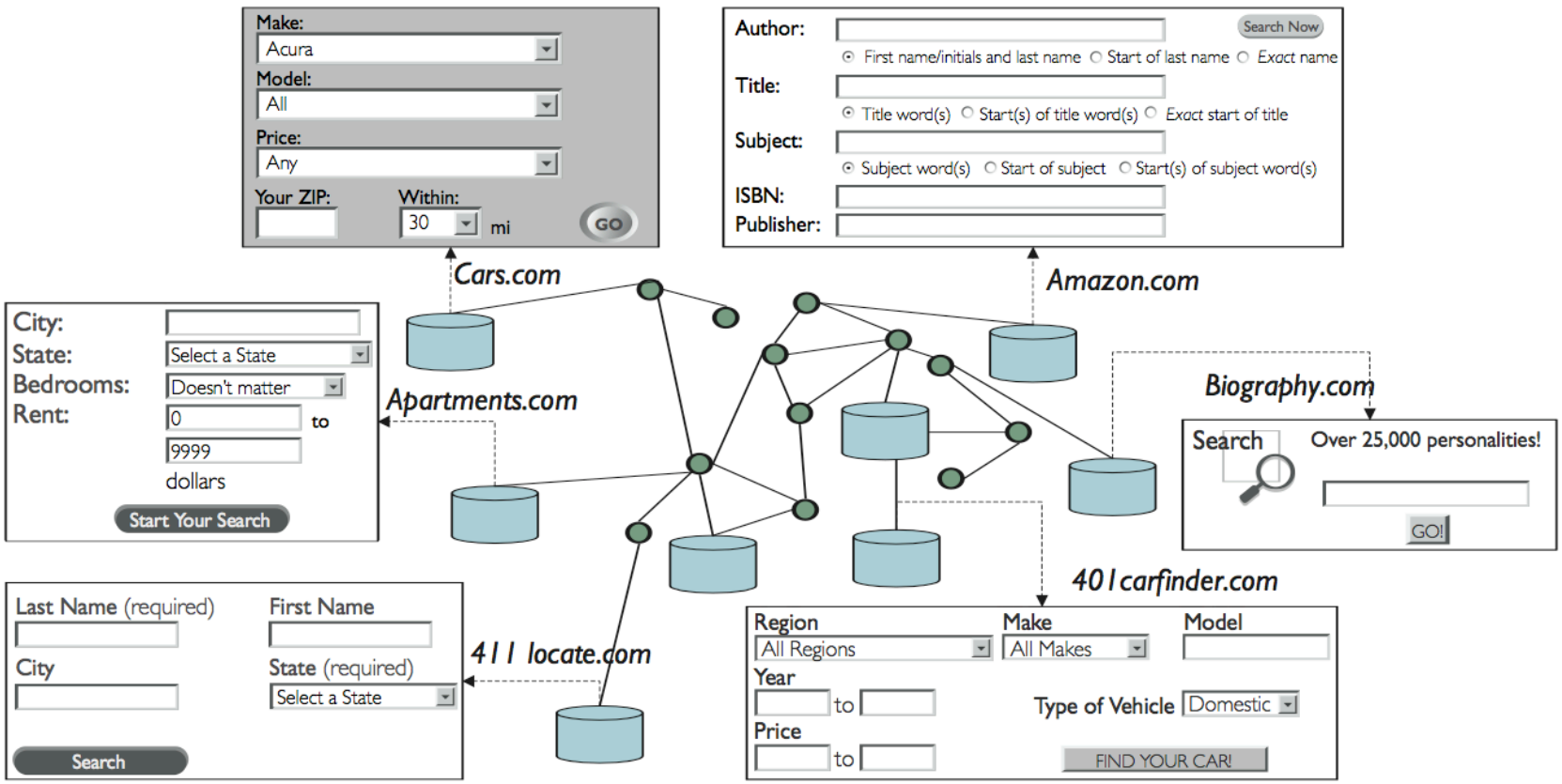
What is the Deep Web?

- Web pages (HTML mostly) have been indexed and searched for many years
- Such pages constitute the so-called Surface Web
  - huge, invaluable amount of information

What is the Deep Web?

- Web pages (HTML mostly) have been indexed and searched for many years
- Such pages constitute the so-called Surface Web
  - huge, invaluable amount of information
- The web has also continuously "deepened"
  - searchable databases, accessible usually through HTML forms
- The Deep Web (aka Hidden Web or Invisible Web) is not effectively crawlable nor indexeable
  - it is largely unexplored, apart from manual queries issued by users

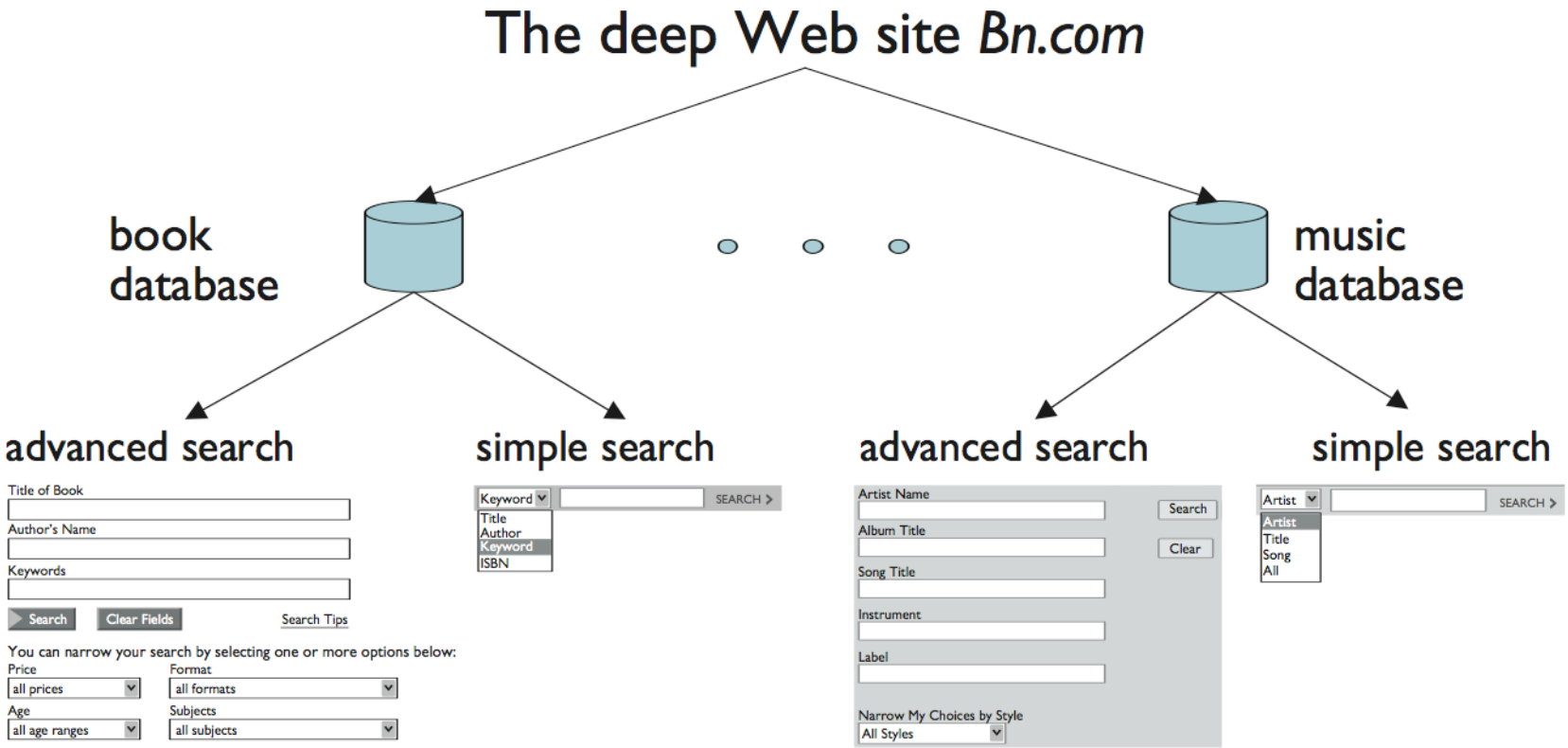
The conceptual view of the Deep Web [He et al. 2007]



A study of the existing Deep Web [He at al. 2007]

- Random sampling of 1M IP addresses
  - reserved and unused IPs removed
- HTML forms as query interfaces
  - non-query forms removed: site-search, login, subscription etc.
- duplicate query interfaces are removed

Duplicate query interfaces: example



Deep Web databases and sites

- Often, different query interfaces access the same database
- Two interfaces access the same data iff the objects retrieved from one can be found by accessing the other, and vice-versa
- Test: take five objects from one and check if they are found in the other
- Not always feasible!

Where to find query interfaces

- 100k IPs sampled
- 281 web servers found, crawled up to depth 10
  - 24 Deep Web sites
  - 129 query interfaces
  - 34 web databases
- 72% of interfaces within depth 3
- 94% of web databases appearing within depth 2
- 91.6% of Deep Web sites had their database within depth 3
- Deep Web not too deep

The scale of the Deep Web

- 1M IPs crawled up to depth 3
- Extrapolating from the 1M IPs to the entire IP space we get the following

	Sampling Results	*Total Estimate*	*99% Confidence Interval*
Deep Web sites	126	307,000	236,000 - 377,000
Web databases	190	450,000	366,000 - 535,000
–unstructured	*43*	*102,000*	62,000 - 142,000
–structured	*147*	*348,000*	275,000 - 423,000
Query interfaces	406	1,258,000	1,097,000 - 1,419,000

How structured is the Deep Web?

- Unstructured databases: objects as unstructured media (video, text, audio etc.)
- Structured databases: attribute-value pairs (i.e., relational tables)
- Analysis by manual inspection

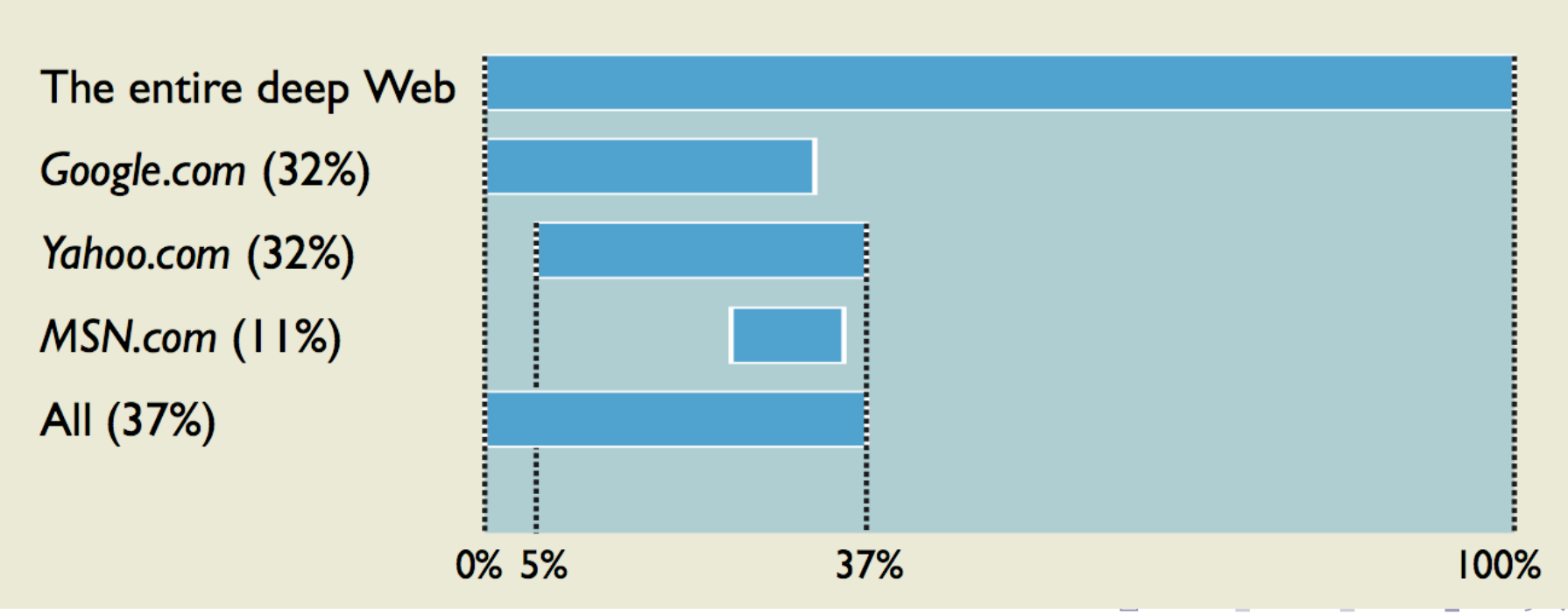
	Sampling Results	*Total Estimate*	*99% Confidence Interval*
Deep Web sites	126	307,000	236,000 - 377,000
Web databases	190	450,000	366,000 - 535,000
–unstructured	*43*	*102,000*	62,000 - 142,000
–structured	*147*	*348,000*	275,000 - 423,000
Query interfaces	406	1,258,000	1,097,000 - 1,419,000

Subject distribution of the Deep Web

- Manual inspection
- Great diversity
- E-commerce sites only 49%
- Deep Web sources emerging outside e-commerce

Coverage of the Deep Web by search engines

- Some Deep Web result pages can be indexed (reachable via URL)
- Test: on a sample subset of Deep Web sites:
  - manually access with five random inputs
  - checking if the result pages are indexed by executing suitable searches



Coverage of the Deep Web by search engines

- Quite large coverage (Google and Yahoo! 32%) but with significant overlap
- The Deep Web is not completely invisible
  - however, a large part of it is!
- Different situation on the Shallow Web
  - little coverage overlap
  - possibility of effectively combining different search engines

Coverage of Deep Web directories

- Web portals exist that classify Deep Web databases in taxonomies
- Small coverage (15.6% max; 0.2% in one case)
- Manual (at least apparently) classification does not scale to the large size of the Deep Web

Access limitations

Access limitations

Access limitations

Access limitations

Sources with access limitations

- Data sources in general may not always be queried freely:
  - Deep Web data accessible via forms
  - Legacy data wrapped in relational tables
  - Web services
  - ...
- Such sources are said to have access limitations
  - Some arguments may require to be bound to a constant
  - input vs output arguments
  - sometimes called bound vs free arguments

Access limitations - synonyms

- Access limitations are also commonly referred to as:
  - binding patterns
  - access patterns
  - access constraints
  - access hindrances

Access limitations - synonyms

- Access limitations are also commonly referred to as:
  - binding patterns
  - access patterns
  - access constraints
  - access hindrances

    used only, but not always, by us

Example of data with access limitations

Data behind forms

- Data accessible through Web forms
  - phone directories
  - auctions
  - stores
  - ...

Modeling access limitations

Example: white pages

- Impossible to ask for all entries (filling in no fields)
- At least one field must be filled in
- The result is a table

Modeling access limitations

Example: white pages

- Impossible to ask for all entries (filling in no fields)
- At least one field must be filled in
- The result is a table

Modeling

- We model each source as a relational table with access limitations
- Filling in a field in the form corresponds to querying with a selection only

Example: whitepages.com – query

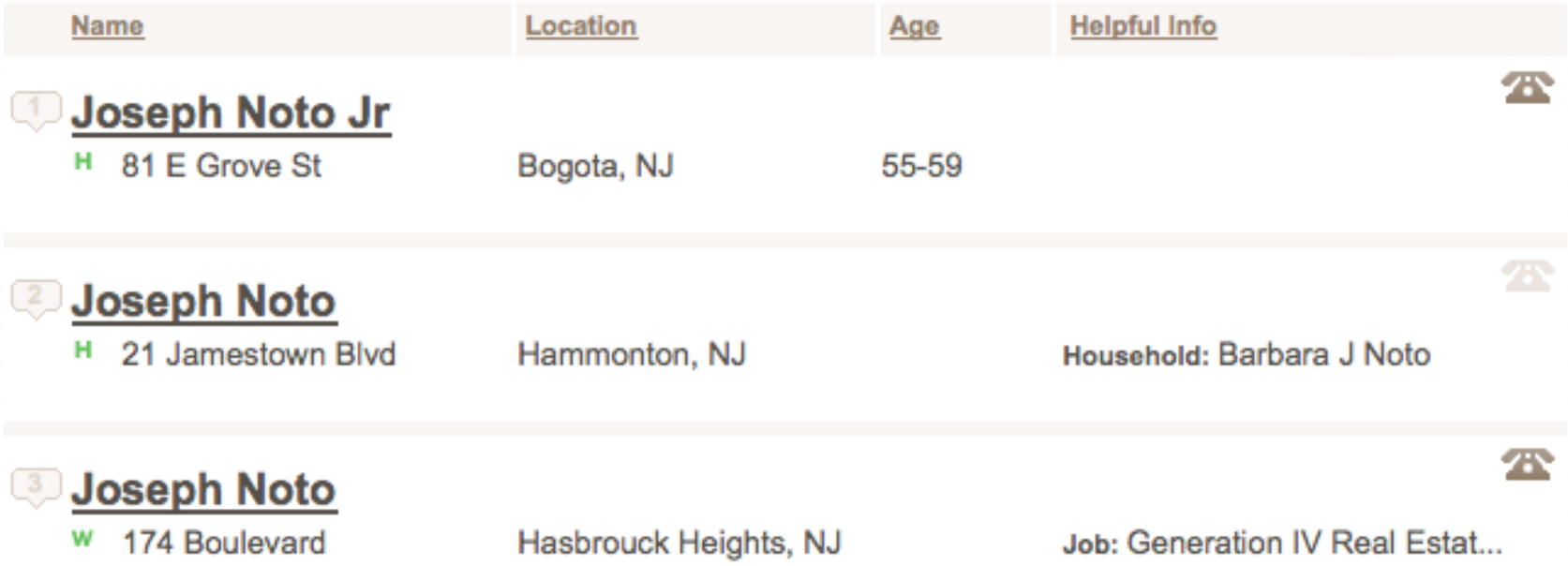
Example: whitepages.com – query


```
SELECT *
FROM whitepages
WHERE firstname='Joseph'
 AND lastname='Noto'
 AND stateprov='NJ'
```

Example: whitepages.com – results

	Name	Location	Age	Helpful Info
1	**Joseph Noto Jr** H 81 E Grove St	Bogota, NJ	55-59	
2	**Joseph Noto** H 21 Jamestown Blvd	Hammonton, NJ		**Household:** Barbara J Noto
3	**Joseph Noto** W 174 Boulevard	Hasbrouck Heights, NJ		**Job:** Generation IV Real Estat...

Example: whitepages.com – results



Name	Location	Age	Helpful Info
1 **Joseph Noto Jr** H 81 E Grove St	Bogota, NJ	55-59	
2 **Joseph Noto** H 21 Jamestown Blvd	Hammonton, NJ		**Household:** Barbara J Noto
3 **Joseph Noto** W 174 Boulevard	Hasbrouck Heights, NJ		**Job:** Generation IV Real Estat...

First	Last	House no.	···
Joseph Jr	*Noto*	*81*	···
Joseph	*Noto*	*21*	···
Joseph	*Noto*	*174*	···

Relevant scenarios in the deep Web

- Surfacing
  - Indexing of deep Web result pages for search engines
- Centralized data integration
  - Sources are registered and wrapped in advance
  - Semantic mappings between sources and mediated schema are manually built
- Large-scale on-the-fly integration
  - Sources are chosen at query processing time

Surfacing

Surfacing

Surfacing

Surfacing

Surfacing the Deep Web

- Surfacing precomputes relevant form submissions for all available HTML forms

  no specialization on a single domain
- The resulting URLs are indexed as any other web page
- The goal is a seamless inclusion of Deep Web pages into the web
- When a user clicks on a result, she is presented with fresh data contents by being redirected to the Deep Web source
- This allows for the inclusion of Deep Web pages into search engines

Scalability and main challenges

- The estimated number [Madhavan et al. 2007] of high-quality HTML forms is 10M approximately
- We need full automation

  Any human effort would be useless
- Main challenges of surfacing:
  1. deciding which forms to fill
  2. finding appropriate inputs for the forms

Considerations

- There are different kinds of inputs:
  1. selection inputs (e.g., pull-down)
  2. text inputs
  3. presentation inputs (e.g., sort-by)
- The fundamental problem is finding a good set of for submissions

Correlation among inputs

- Some inputs might be correlated (e.g., CITY and STATE, MINSALARY and MAXSALARY)
- Therefore, the set of values for an input to be instantiated should in principle different depending on the query template (binding pattern)
- However, this complicates things significantly
  - never considered in the literature

More considerations

- In surfacing, we aim at maximizing the coverage on the underlying data, while limiting the number of form submissions
- We want to cover as many Deep Web sites as possible
- Better to cover relevant information from many sites than deeply covering a few ones
- We do not want to cover the full contents of a Deep Web site: it is instead important to provide the search engine with seeds to diverse information on the site.

Selecting query templates: a tradeoff

- Selecting query templates with many input attributes retrieves in general more data, but it increases crawling traffic (generates all possible queries, or a large fraction of them)
- On the other hand, choosing too few input attributes retrieves too many tuples, often split into pages.
- Tradeoff!

Informative query templates

- A query template is informative if it generates several distinct pages from its form submissions
- Generated pages are assigned a signature; the higher the ratio signatures/submissions, the more informative the query template

Looking for informative templates

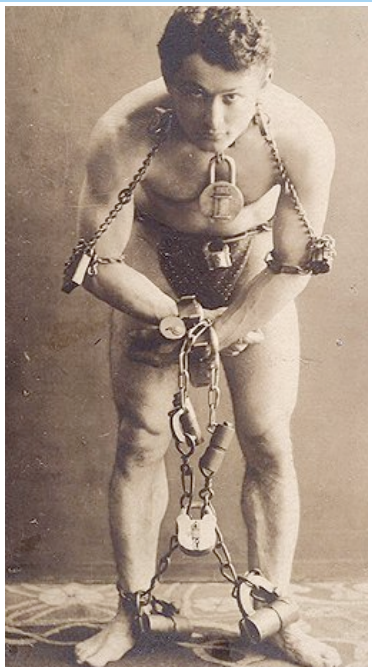
- Bottom-up method: if a candidate template of dimension (no. of input attributes) k is informative, we look for another of dimension $k+1$ that has as input attributes a superset of those of the former.
- This prunes the search significantly

Query Answering under Access Limitations

Query Answering under Access Limitations

Query Answering under Access Limitations

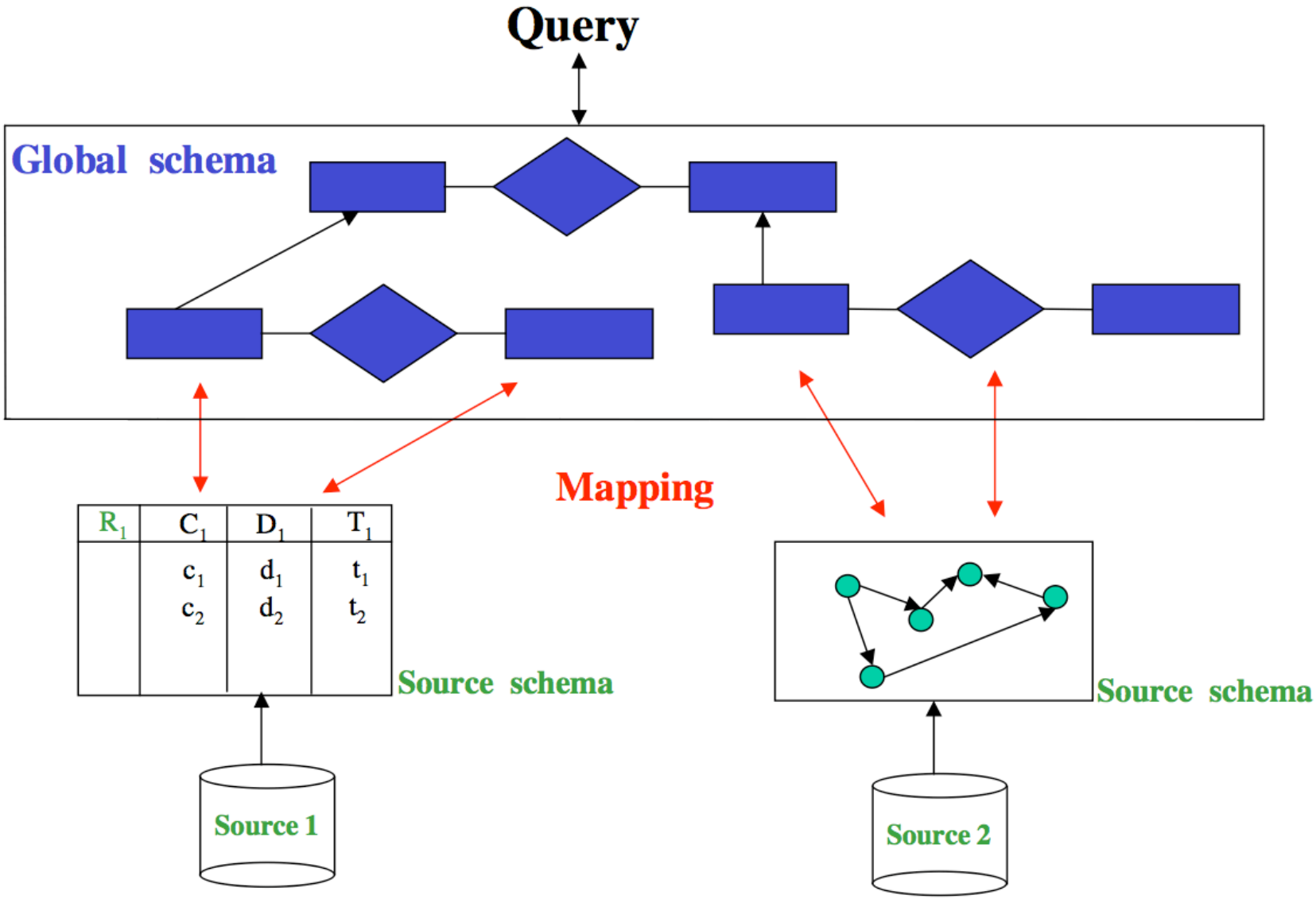
Query Answering under Access Limitations



Traditional data integration setting

- Sources are wrapped as relational tables
  - with access limitations
- They are registered to a central system (mediator) with a reconciled schema
- Mappings between the wrapped sources and the mediator are manually (or semi-automatically) built
- Queries are posed over the reconciled schema
- We will focus on the problems posed by access limitations

Traditional data integration setting [Lenzerini IJCAI'03]



Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Query

$q(A) \leftarrow r_2(A, italy, C), r_1(T, ny, A)$

Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Query

$$q(A) \leftarrow r_2(A, italy, C), r_1(T, ny, A)$$

Answering strategy: q cannot be executed from left to right

- the body atoms of q can be reordered:

$$q'(A) \leftarrow r_1(T, ny, A), r_2(A, italy, C)$$

- q and q' are equivalent (q' is executable and q is orderable)
- its answer can be obtained as with no access limitations

Query answering under access limitations

Finding the exact answer

Can a CQ always be answered with all the tuples that would be found without access limitations?

Query answering under access limitations

Finding the exact answer

Can a CQ always be answered with all the tuples that would be found without access limitations?

- Is CQ q executable *as is* from left to right?

Query answering under access limitations

Finding the exact answer

Can a CQ always be answered with all the tuples that would be found without access limitations?

- Is CQ q executable *as is* from left to right?
- It CQ q orderable into an executable query?

Query answering under access limitations

Finding the exact answer

Can a CQ always be answered with all the tuples that would be found without access limitations?

- Is CQ q executable *as is* from left to right?
- It CQ q orderable into an executable query?
  - Orderability sometimes called *feasibility* (ambiguous) and *executability* (ambiguous)
  - Can be considered as a syntactic version of executability
  - Reordering efficiently done in [Yang et al. PODS'06]
- Is CQ q stable, i.e., equivalent to an executable query
  - Stability sometimes called *feasibility* (ambiguous)
  - Can be considered as a semantic version of executability

Query answering under access limitations

Examples

Schema: $r_1^{oio}(Title, City, Artist)$,
$r_2^{ioo}(Artist, Nation, City)$

Query (executable): $q_e(A) \leftarrow r_1(T, ny, A), r_2(A, italy, C)$
Query (orderable): $q_f(A) \leftarrow r_2(A, italy, C), r_1(T, ny, A)$
Query (stable): $q_s(A) \leftarrow r_2(A, italy, C), r_1(T, ny, A), r_1(T, C', A)$

Query answering under access limitations

Results on stability

Theorem ([Li & Chang ICDT'01])

A CQ is stable iff its minimal equivalent is orderable.

Theorem ([Nash & Ludäscher PODS'04])

Stability is exactly as hard as query containment

Corollary ([Li & Chang ICDT'01])

Stability of conjunctive queries is NP-complete
Stability of Datalog queries in undecidable

Query answering under access limitations

Results on orderability

Theorem ([Yang et al. PODS'06])

Orderability is in P for conjunctive queries

Theorem ([Nash & Ludäscher PODS'04])

Orderability is NP-complete for FO queries

Theorem ([Yang et al. PODS'06])

Orderability is PSPACE-complete for non-recursive Datalog queries

Query answering under access limitations

Observations

- Limitations generally restrict the answers we can retrieve
  - Queries are not always stable
- We are interested in the best approximation of the query answers
- This might even require accessing off-query relations

Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Query

$q(A) \leftarrow r_2(A, italy, modena)$

Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Query

$q(A) \leftarrow r_2(A, italy, modena)$

Best effort answering: no executable reordering of q exists

- Starting from the constant *modena*, we can access r_1
- then we can obtain tuples with new *Artist* constants
- with such values we can access r_2 and start over

Example

Superscripts denote input and output attributes

Schema: concerts and artists

$r_1^{oio}(Title, City, Artist)$
$r_2^{ioo}(Artist, Nation, City)$

Query

$q(A) \leftarrow r_2(A, italy, modena)$

Best effort answering: no executable reordering of q exists

- Starting from the constant *modena*, we can access r_1
- then we can obtain tuples with new *Artist* constants
- with such values we can access r_2 and start over
- We consider abstract domains (*Year*, *Artist* etc.)

⋆ We assume never to enumerate domain values

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*

Relation r_2

Artist	Nation	City

Answer tuples

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*

Relation r_2

Artist	Nation	City
conte	*italy*	*genoa*

Answer tuples

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*
volare	*genoa*	*modugno*
K551	*genoa*	*kissin*

Relation r_2

Artist	Nation	City
conte	*italy*	*genoa*

Answer tuples

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*
volare	*genoa*	*modugno*
K551	*genoa*	*kissin*

Relation r_2

Artist	Nation	City
conte	*italy*	*genoa*
kissin	*russia*	*moscow*
modugno	*italy*	*bari*

Answer tuples

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*
volare	*genoa*	*modugno*
K551	*genoa*	*kissin*
sole_mio	*moscow*	*pavarotti*

Relation r_2

Artist	Nation	City
conte	*italy*	*genoa*
kissin	*russia*	*moscow*
modugno	*italy*	*bari*

Answer tuples

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*
volare	*genoa*	*modugno*
K551	*genoa*	*kissin*
sole_mio	*moscow*	*pavarotti*

Relation r_2

Artist	Nation	City
conte	*italy*	*genoa*
kissin	*russia*	*moscow*
modugno	*italy*	*bari*
pavarotti	*italy*	*modena*

Answer tuples

Example (cont'd)

Relation r_1

Title	City	Artist
azzurro	*modena*	*conte*
volare	*genoa*	*modugno*
K551	*genoa*	*kissin*
sole_mio	*moscow*	*pavarotti*

Relation r_2

Artist	Nation	City
conte	*italy*	*genoa*
kissin	*russia*	*moscow*
modugno	*italy*	*bari*
pavarotti	*italy*	*modena*

Answer tuples

$\langle pavarotti \rangle$

Query answering under access limitations

Possible approximations

- Maximal answer a.k.a. maximally contained answer or reachable certain answer or obtainable answer
  - largest sound set of answer tuples that can be obtained by a query plan that respects the access limitations
- Minimally containing answer
  - smallest complete set of answer tuples that can be obtained by a query plan that respects the access limitations

Query answering under access limitations

- Schema $\mathcal{S}$ with access limitations
- Database instance D over $\mathcal{S}$
- Query q over D
- Set of initially known constants I
- Maximal answer denoted $\text{ans}(q, \mathcal{S}, D, I)$

Finding the maximal answer

- Basic technique in [Millstein et al. 2000]
- Answering is inherently recursive
- Need for a set of initial constants (usually those in the query)
- Notion of abstract domain associated to an attribute
- Encoding in positive Datalog

Query answering under access limitations

Finding the maximal answer: execution strategy

- To get all obtainable tuples, we need all possible constants for input attributes
- Enumerating all values of a domain: generally unfeasible
- Use all constants in the query and all constants from tuples retrieved from other relations
  - even those that are not mentioned in the query

Query answering under access limitations

Naive algorithm to compute the maximal answer

1. B = set of constants initially in the query
2. C = set of caches (one per relation)
3. while new valid accesses are possible
   - Make all accesses you can with constants in B
   - Put the obtained tuples in the corresponding cache in C
   - Put the obtained constants in B
4. Evaluate the query over the cache

Query answering under access limitations

Naive algorithm to compute the maximal answer

1. B = set of constants initially in the query
2. C = set of caches (one per relation)
3. while new valid accesses are possible
   - Make all accesses you can with constants in B
   - Put the obtained tuples in the corresponding cache in C
   - Put the obtained constants in B
4. Evaluate the query over the cache

Answerability

A query q is answerable if there exists at least an instance D such that the maximal answer to q in D is non-empty (unanswerable otherwise)

Properties of queries under access limitations

Examples

Schema: $r_1^{oio}(Title, City, Artist), r_2^{ioo}(Artist, Nation, City)$

executable: $q_e(A) \leftarrow r_1(T, ny, A), r_2(A, italy, C)$

orderable: $q_f(A) \leftarrow r_2(A, italy, C), r_1(T, ny, A)$

stable: $q_s(A) \leftarrow r_2(A, italy, C), r_1(T, ny, A), r_2(T, C', A)$

answerable: $q_a(A) \leftarrow r_2(A, italy, modena)$

unanswerable: $q_u(A) \leftarrow r_2(A, italy, C)$

query	executable	orderable	stable	answerable
q_e	yes	yes	yes	yes
q_f		yes	yes	yes
q_s			yes	yes
q_a				yes
q_u	no	no	no	no

Naive program for previous example

$$
\begin{array}{lrcl}
\rho_1: & q(A) & \leftarrow & \hat{r}_2(A, italy, modena) \\
\rho_2: & \hat{r}_1(T, C, A) & \leftarrow & dom_C(C), r_1(T, C, A) \\
\rho_3: & \hat{r}_2(A, N, C) & \leftarrow & dom_A(A), r_2(A, N, C) \\
\rho_4: & dom_T(T) & \leftarrow & \hat{r}_1(T, C, A) \\
\rho_5: & dom_C(C) & \leftarrow & \hat{r}_1(T, C, A) \\
\rho_6: & dom_A(A) & \leftarrow & \hat{r}_1(T, C, A) \\
\rho_7: & dom_A(A) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_8: & dom_N(N) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_9: & dom_C(C) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_{10}: & dom_N(italy) & & \\
\rho_{11}: & dom_C(modena) & &
\end{array}
$$

Naive program for previous example

$$
\begin{array}{rrcl}
\rho_1: & q(A) & \leftarrow & \hat{r}_2(A, italy, modena) \\
\rho_2: & \hat{r}_1(T, C, A) & \leftarrow & dom_C(C), r_1(T, C, A) \\
\rho_3: & \hat{r}_2(A, N, C) & \leftarrow & dom_A(A), r_2(A, N, C) \\
\rho_4: & dom_T(T) & \leftarrow & \hat{r}_1(T, C, A) \\
\rho_5: & dom_C(C) & \leftarrow & \hat{r}_1(T, C, A) \\
\rho_6: & dom_A(A) & \leftarrow & \hat{r}_1(T, C, A) \\
\rho_7: & dom_A(A) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_8: & dom_N(N) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_9: & dom_C(C) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_{10}: & dom_N(italy) & & \\
\rho_{11}: & dom_C(modena) & &
\end{array}
$$

rewritten query : ρ_1

cache rules: ρ_2, ρ_3

domain rules: $\rho_4 - \rho_{11}$

initial constants: *italy*, *modena*

Naive program for previous example

$$
\begin{array}{lrcl}
\rho_1: & q(A) & \leftarrow & \hat{r}_2(A, italy, modena) \\
\rho_2: & \hat{r}_1(T, C, A) & \leftarrow & dom_C(C), r_1(T, C, A) \\
\rho_3: & \hat{r}_2(A, N, C) & \leftarrow & dom_A(A), r_2(A, N, C) \\
\rho_4: & dom_T(T) & \leftarrow & \hat{r}_1(T, C, A) \\
\cancel{\rho_5:} & \cancel{dom_C(C)} & \cancel{\leftarrow} & \cancel{\hat{r}_1(T, C, A)} \\
\rho_6: & dom_A(A) & \leftarrow & \hat{r}_1(T, C, A) \\
\cancel{\rho_7:} & \cancel{dom_A(A)} & \cancel{\leftarrow} & \cancel{\hat{r}_2(A, N, C)} \\
\rho_8: & dom_N(N) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_9: & dom_C(C) & \leftarrow & \hat{r}_2(A, N, C) \\
\rho_{10}: & dom_N(italy) & & \\
\rho_{11}: & dom_C(modena) & &
\end{array}
$$

rewritten query : ρ_1

cache rules: ρ_2, ρ_3

domain rules: $\rho_4 - \rho_{11}$

initial constants: *italy*, *modena*

Finding the relevant relations

Finding the relevant relations

Finding the relevant relations

Finding the relevant relations

Optimizing the execution strategy

The naive algorithm

- Inefficient bottom-up approach:
  - All relations in the schema are always accessed in all possible ways
  - Accesses are costly (sources on the Web)

Optimizing the execution strategy

The naive algorithm

- Inefficient bottom-up approach:
  - All relations in the schema are always accessed in all possible ways
  - Accesses are costly (sources on the Web)

Idea: avoiding irrelevant accesses

- Some of the relations may be irrelevant to the query, i.e., they cannot help discovering tuples in the maximal answer
  - This depends on the query and the schema
  - Not only the relations mentioned in the query, but also the joins between them
- Accesses to such relations should be avoided

Relevance

Definition: relevance

A relation r is relevant for a query q if there are two instances D_1, D_2 that differ only on the tuples of R, and such that $\text{ans}(q, \mathcal{S}, D_1, I) \neq \text{ans}(q, \mathcal{S}, D_2, I)$.

$\text{ans}(q, \mathcal{S}, D, I)$: maximal answer to q over schema $\mathcal{S}$ (with limitations Λ), evaluated over database D using initial constants I (superset of those in q), as with the naive algorithm

Relevance

Definition: relevance

A relation r is relevant for a query q if there are two instances D_1, D_2 that differ only on the tuples of R, and such that $\text{ans}(q, \mathcal{S}, D_1, I) \neq \text{ans}(q, \mathcal{S}, D_2, I)$.

$\text{ans}(q, \mathcal{S}, D, I)$: maximal answer to q over schema $\mathcal{S}$ (with limitations Λ), evaluated over database D using initial constants I (superset of those in q), as with the naive algorithm

Determining relevance

- solved for connection queries in [Li & Chang TODS'01]
- solved for CQs with no projections in [C&M ICDE'08]

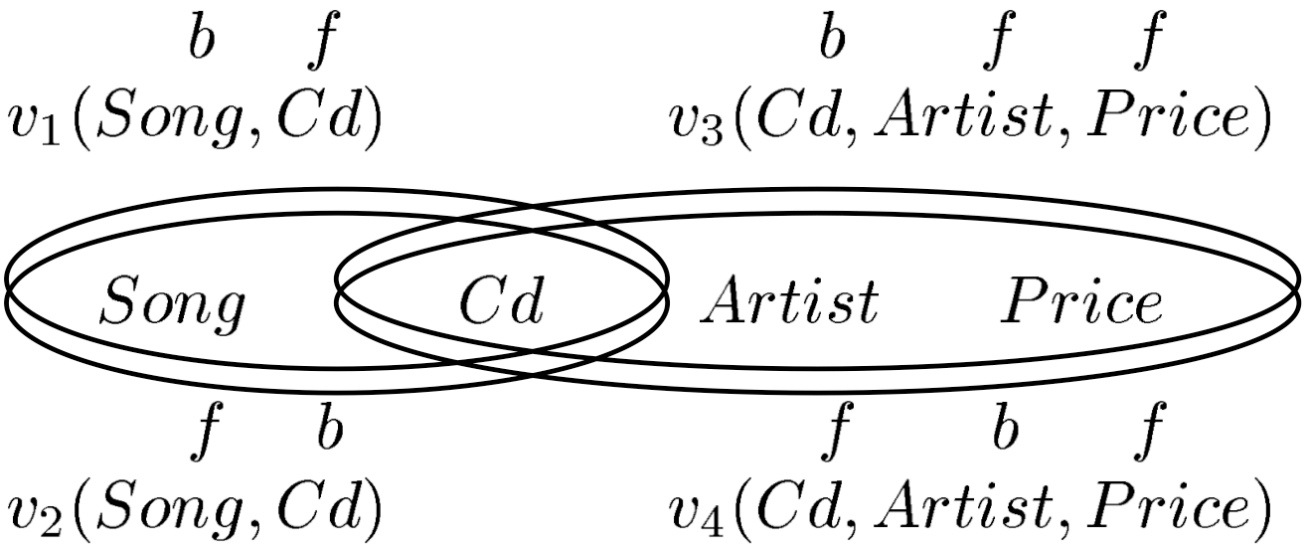
Connection queries [Li &Chang TODS'01]

Definition

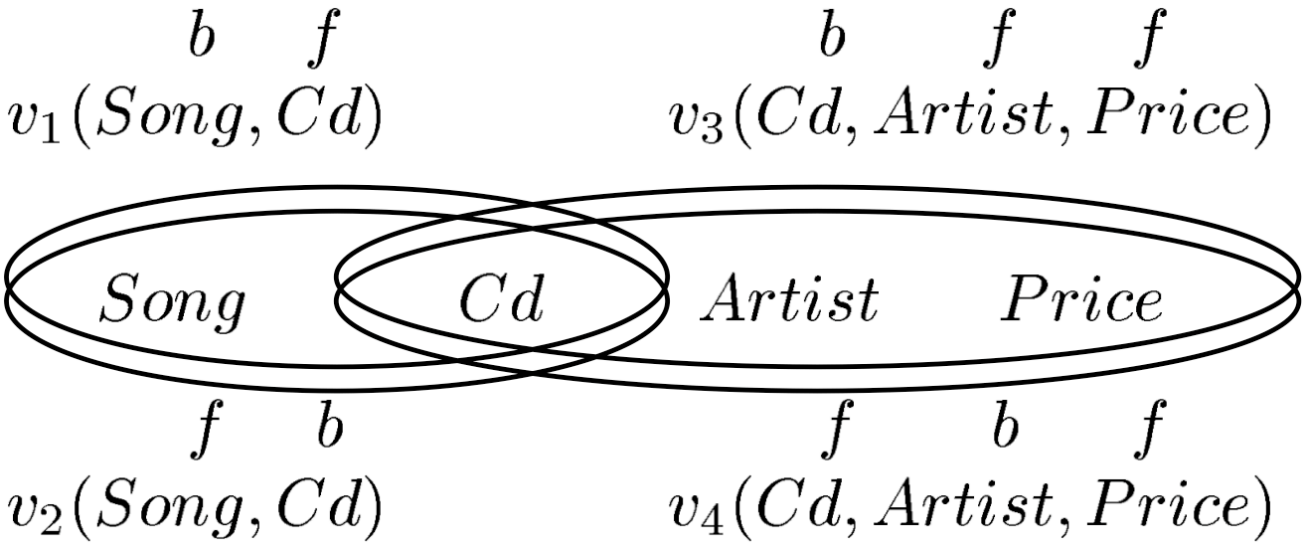
A connection query is a UCQ such that

- in each CQ, all attributes with the same domain are in a join

- Connection queries are useful for relations with disparate domains



Connection queries [Li &Chang TODS'01]



- Queries specified as $\langle I, O, C \rangle$
  - I is a list of input assignments *attribute=constant*
  - O is a list of output attributes the user is interested in
  - C is a list of connections, i.e., natural joins between relations, each join defining a CQ
  - example: $Q = \langle \{Song = t_1\}, \{Price\}, \{C_1, C_2, C_3, C_4\} \rangle$, where
  - $C_1 = \{v_1, v_3\}$, $C_2 = \{v_1, v_4\}$, $C_3 = \{v_2, v_3\}$, $C_4 = \{v_2, v_4\}$

Connection queries [Li &Chang TODS'01]

- A rewriting as a Datalog query can be provided that does not use any non-relevant relation
- ⋆ Connection queries do not cover CQs

Example: not a connection query

Schema: $supervisor^{oi}(Person, Person)$

- Can only ask who is supervisor of him/herself
- $q(X) \leftarrow supervisor(X, X)$ is a connection query
- $q(X, Y) \leftarrow supervisor(X, Y)$ is not a connection query

Relevance for CQs with no projections

- Relevance defined w.r.t. the body of a CQ in [C&M ICDE'08]
- This is equivalent to considering CQs with no projections
- Albeit incomplete, this notion of relevance allows pruning (some, perhaps not all) irrelevant relations from a query plan

Relevance for CQs with no projections

- Given a query and the schema, we represent dependencies among relations with a dependency graph (d-graph):
  - nodes are attributes
  - arcs tell which attributes provide values to feed attributes
- Sketchily represents how to extracts answers with the naive approach
- We prune non-relevant relations and accesses by deleting edges

Dependency graph

Nodes in the d-graph of query q:

- One black node for each argument of each atom occurring in q
- One white node for each argument of each relation not occurring in q
- Also marked with the access mode and the abstract domain

Arcs $u \longrightarrow v$ (v can receive values from u) if

- u and v have the same domain
- u is an output argument
- v is an input argument

Example

Schema

$showing^{io}(Location, Movie)$ $actor^{io}(Movie, Person)$
$married^{io}(Person, Person)$ $moviestar^{o}(Person)$
$production^{io}(Studio, Movie)$

Example

Schema

$showing^{io}(Location, Movie)$ $\quad$ $actor^{io}(Movie, Person)$
$married^{io}(Person, Person)$ $\quad$ $moviestar^{o}(Person)$
$production^{io}(Studio, Movie)$

Query

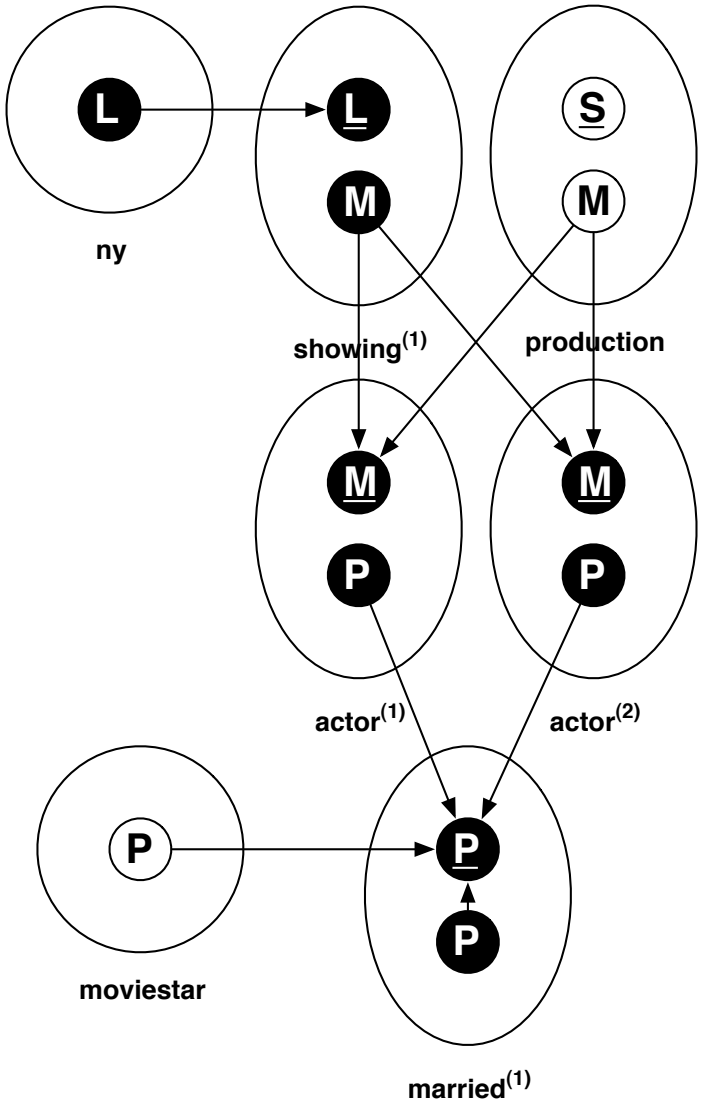
"actors in movies showing in New-York whose spouse has also played in some movie"

$$q(A) \leftarrow showing(ny, M), actor(M, A), married(A, A'), actor(M', A')$$

rewritten as

$$q(A) \leftarrow ny(L), showing(L, M), actor(M, A), married(A, A'), actor(M', A')$$

Example: d-graph



Marking the d-graph

Strong and weak arcs: intuition

- A strong arc $u \longrightarrow v$ denotes that v is "fed" through a join with values from u
- A weak arc $u \longrightarrow v$ denotes that v is "fed" with values from u but there is no join
- In the presence of a strong arc incoming on v, all weak arcs arcs incoming on v are unnecessary
  - unless they are needed to provide values to other nodes
- Strong arcs represent conditions that must hold all at the same time (intersection)
- Weak arcs represent conditions that are in a union

Pruning the d-graph

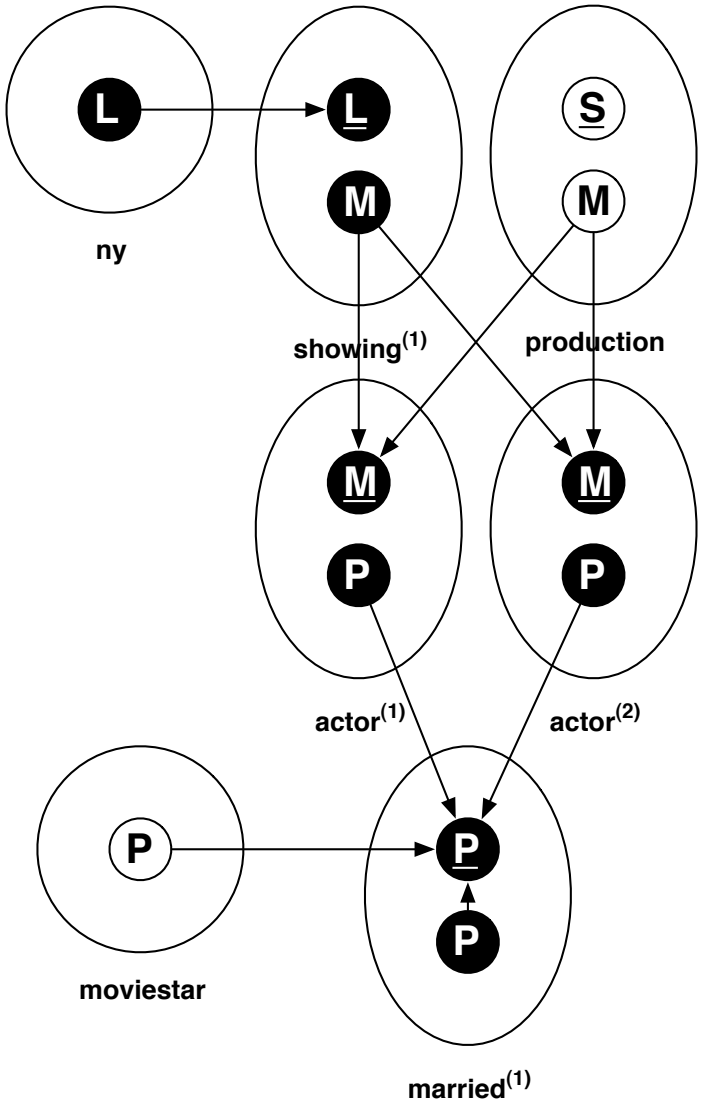
More terminology

- Candidate: arc whose corresponding nodes are in a join
- Cyclic: candidate that is in a cyclic path of candidates
  - Acyclicity enforced on strong arcs to preserve the maximal answer

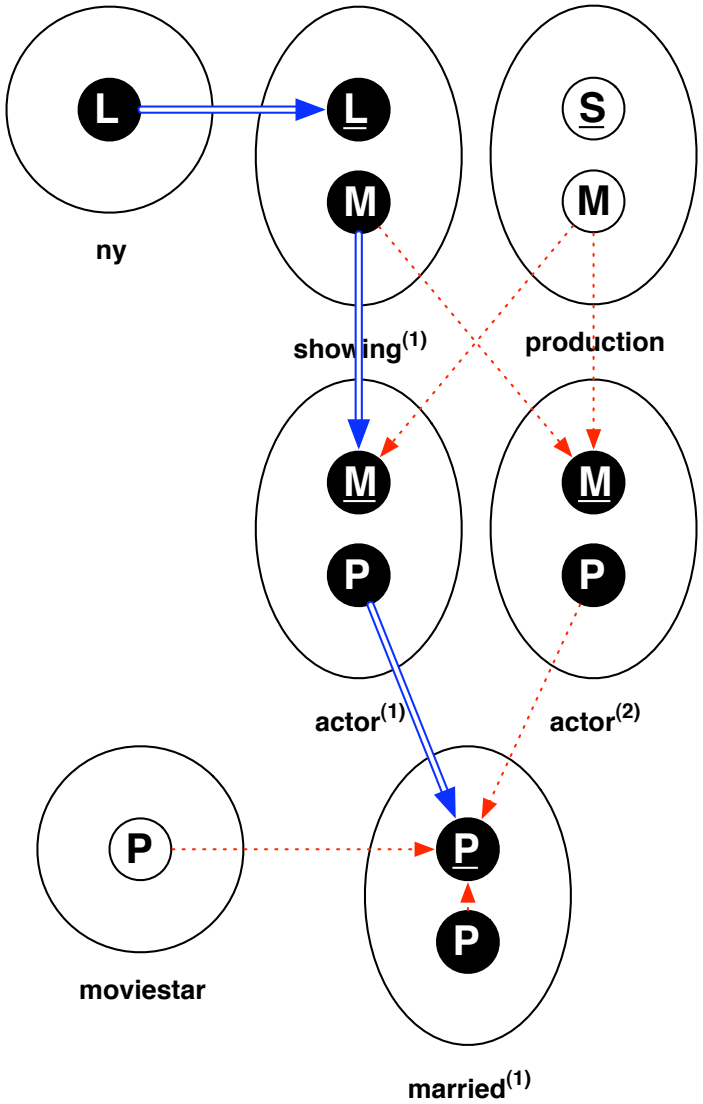
Determining strong and weak arcs

- Initial markings (as big as possible):
  - Strong arcs S_0 = all acyclic candidates
  - Deleted arcs D_0 = all arcs that are not candidates
  - A fixpoint algorithm unmarks the arcs until a consistent configuration is found

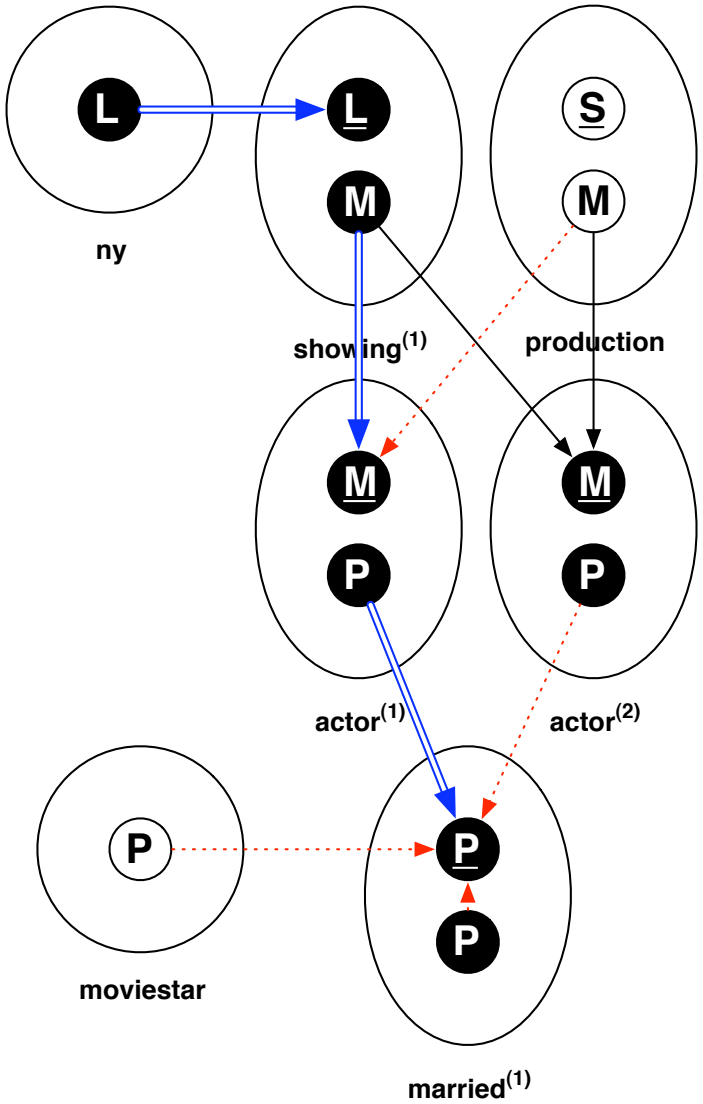
Example: d-graph (not marked)



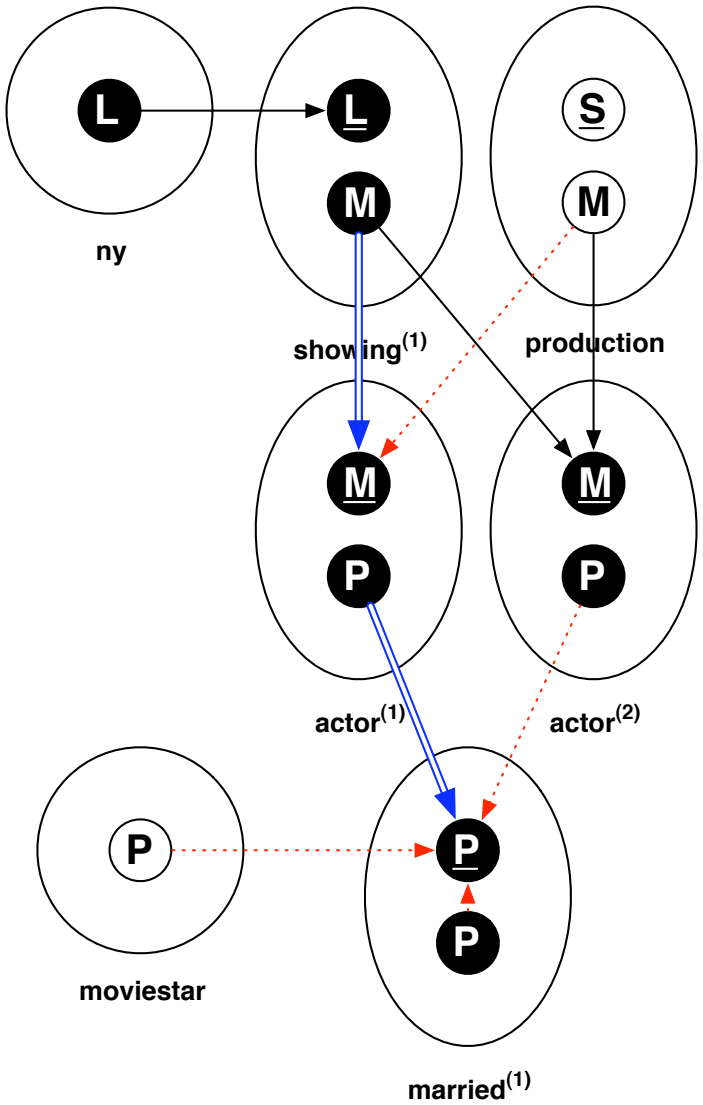
Example (cont'd): initial markings



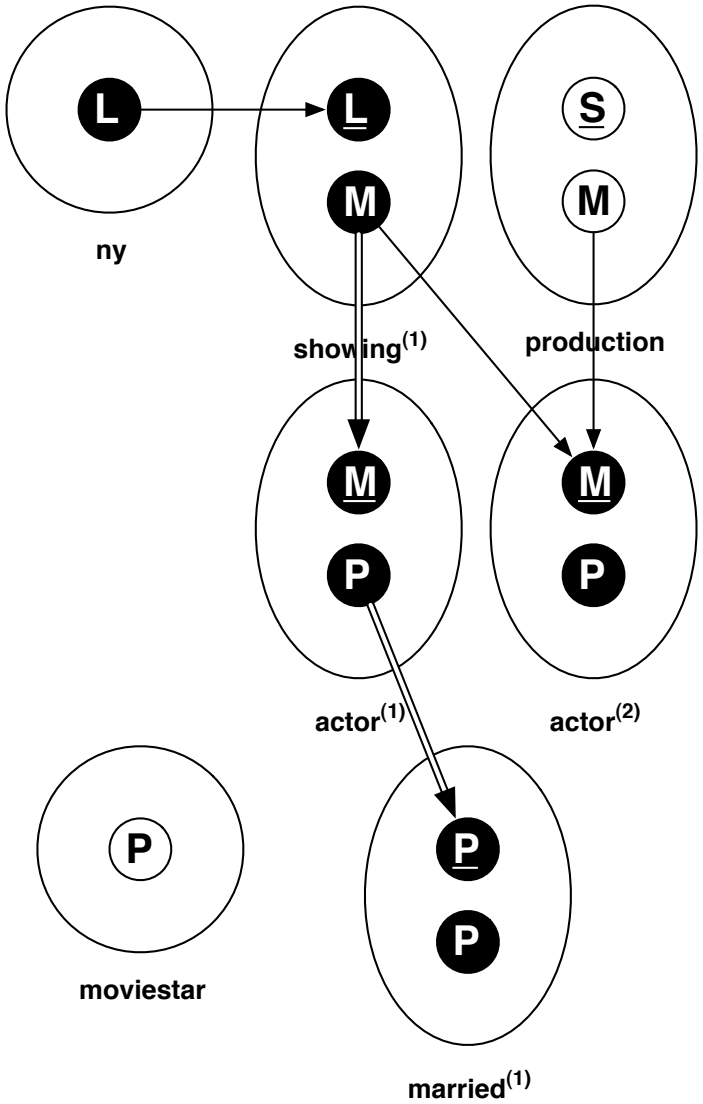
Unmarking deleted arcs not dominated by a strong arc



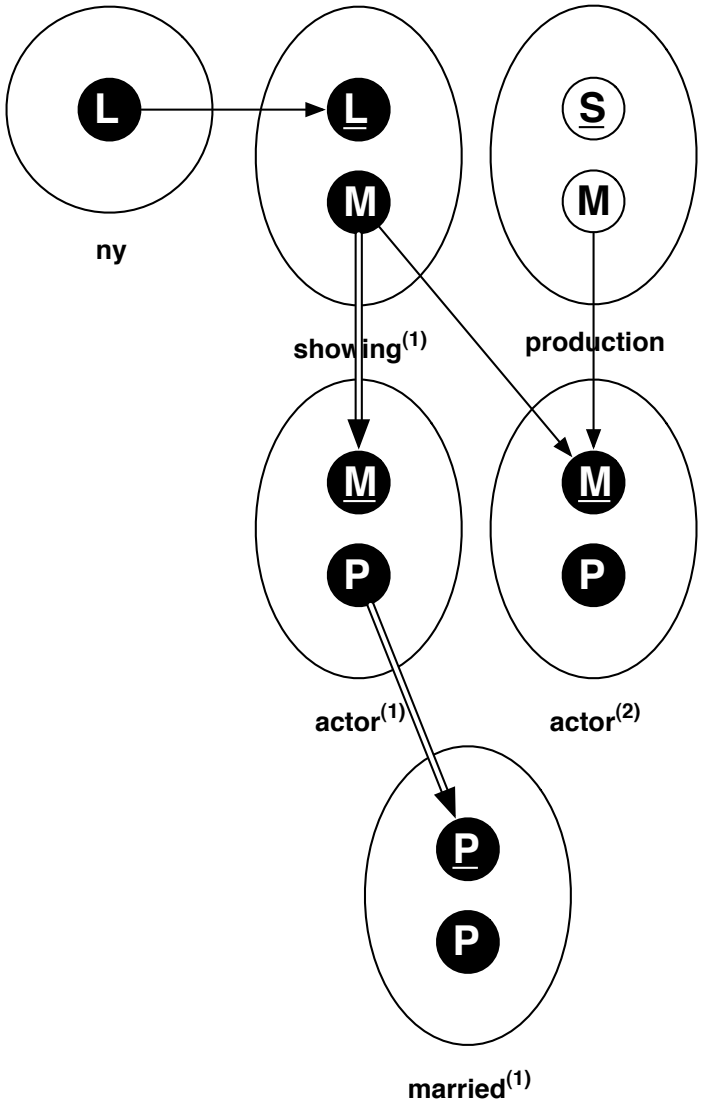
Unmarking strong arc followed by a weak arc



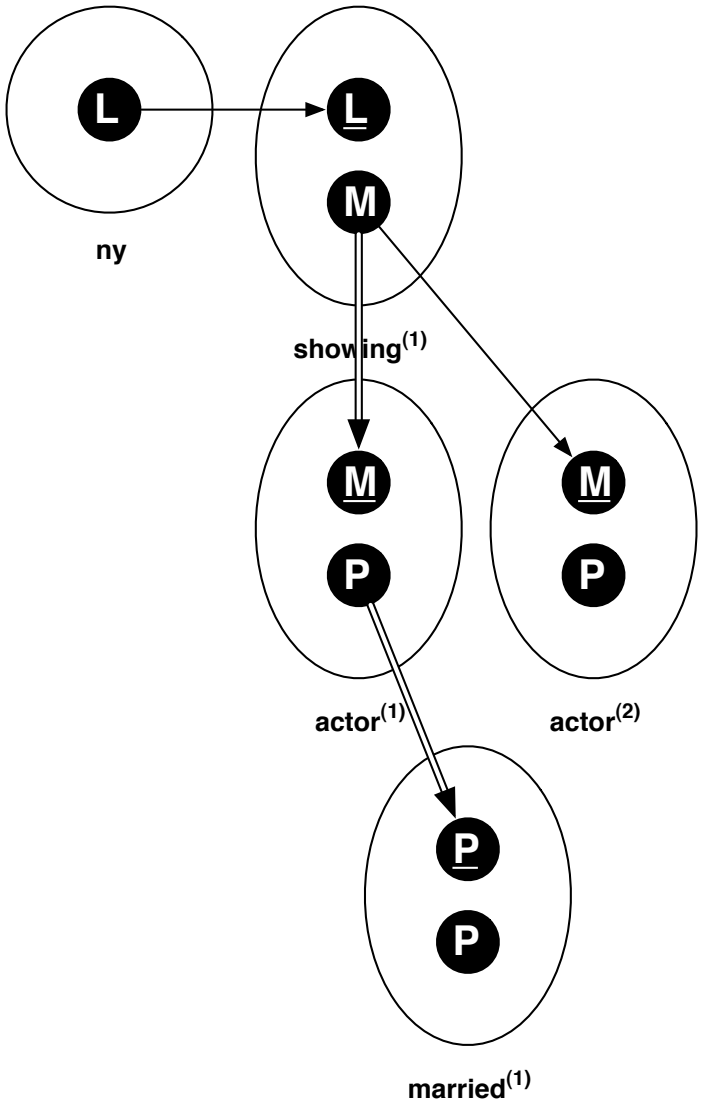
Fixpoint reached



Removing isolated white relations



Removing unaccessible white relations



Optimized query plan from the pruned d-graph

$$
\begin{aligned}
\rho_1 &: \quad q(A) \leftarrow \hat{showing}(ny, M), \hat{actor}_1(M, A), \\
&\qquad\qquad \hat{married}(A, A'), \hat{actor}_2(M', A') \\
\rho_2 &: \quad \hat{actor}_1(M, P) \leftarrow actor(M, P), dom_{Movie_1}(M) \\
\rho_3 &: \quad \hat{actor}_2(M, P) \leftarrow actor(M, P), dom_{Movie_2}(M) \\
\rho_4 &: \quad \hat{married}(P_1, P_2) \leftarrow married(P_1, P_2), dom_{Person}(P_1) \\
\rho_5 &: \quad \hat{showing}(L, M) \leftarrow showing(L, M), dom_{Location}(L) \\
\rho_6 &: \quad dom_{Movie_1}(M) \leftarrow \hat{showing}(\_, M) \\
\rho_7 &: \quad dom_{Movie_2}(M) \leftarrow \hat{showing}(\_, M) \\
\rho_8 &: \quad dom_{Person}(P) \leftarrow \hat{actor}(\_, P) \\
\rho_9 &: \quad dom_{Location}(ny)
\end{aligned}
$$

Optimized query plan from the pruned d-graph

$$
\begin{aligned}
\rho_1 : \quad & q(A) \leftarrow \hat{showing}(ny, M), \hat{actor}_1(M, A), \\
& \hat{married}(A, A'), \hat{actor}_2(M', A') \\
\rho_2 : \quad & \hat{actor}_1(M, P) \leftarrow actor(M, P), dom_{Movie_1}(M) \\
\rho_3 : \quad & \hat{actor}_2(M, P) \leftarrow actor(M, P), dom_{Movie_2}(M) \\
\rho_4 : \quad & \hat{married}(P_1, P_2) \leftarrow married(P_1, P_2), dom_{Person}(P_1) \\
\rho_5 : \quad & \hat{showing}(L, M) \leftarrow showing(L, M), dom_{Location}(L) \\
\rho_6 : \quad & dom_{Movie_1}(M) \leftarrow \hat{showing}(\_, M) \\
\rho_7 : \quad & dom_{Movie_2}(M) \leftarrow \hat{showing}(\_, M) \\
\rho_8 : \quad & dom_{Person}(P) \leftarrow \hat{actor}(\_, P) \\
\rho_9 : \quad & dom_{Location}(ny)
\end{aligned}
$$

rewritten query : ρ_1

cache rules: ρ_2, ρ_5

domain rules: $\rho_6 - \rho_9$

initial constants: ny

Optimized query plan in a more compact form

$$
\begin{array}{lrcl}
\rho_1: & q(A) & \leftarrow & \hat{showing}(ny, M), \hat{actor}(M, A), \\
 & & & \hat{married}(A, A'), \hat{actor}(M', A') \\
\rho_2: & \hat{actor}(M, P) & \leftarrow & actor(M, P), \hat{showing}(\_, M) \\
\rho_3: & \hat{married}(P_1, P_2) & \leftarrow & married(P_1, P_2), \hat{actor}(\_, P_1) \\
\rho_4: & \hat{showing}(ny, M) & \leftarrow & showing(ny, M)
\end{array}
$$

Optimized query plan in a more compact form

$$
\begin{array}{lrcl}
\rho_1: & q(A) & \leftarrow & \hat{showing}(ny, M), \hat{actor}(M, A), \\
 & & & \hat{married}(A, A'), \hat{actor}(M', A') \\
\rho_2: & \hat{actor}(M, P) & \leftarrow & actor(M, P), \hat{showing}(\_, M) \\
\rho_3: & \hat{married}(P_1, P_2) & \leftarrow & married(P_1, P_2), \hat{actor}(\_, P_1) \\
\rho_4: & \hat{showing}(ny, M) & \leftarrow & showing(ny, M)
\end{array}
$$

rewritten query : ρ_1

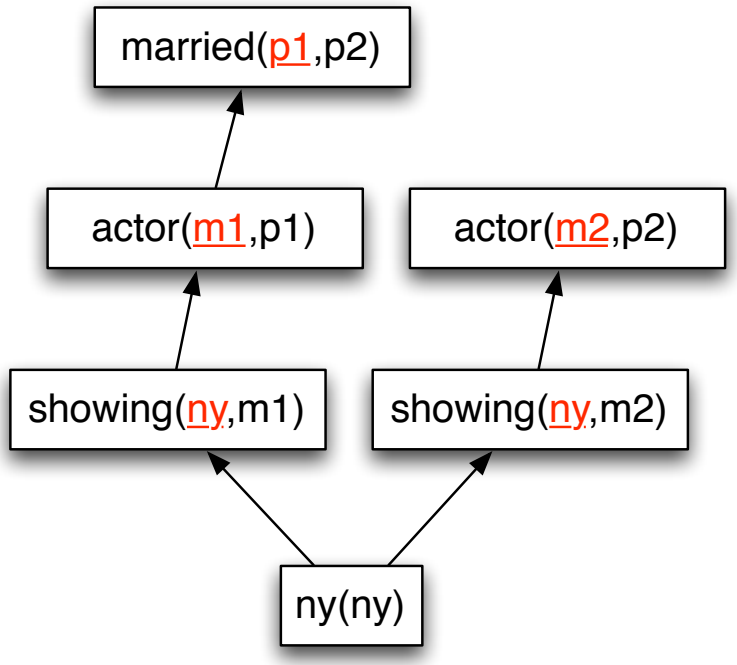
cache rules: ρ_2, ρ_4

initial constants: ny

Properties of the pruned d-graph

Extraction forest

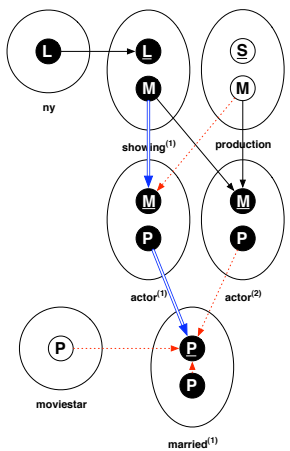
Roots: tuples in the input-free relations
Nodes: tuples in the DB
Arcs $u \curvearrowright v$: if output attribute in u is used in input attribute in v

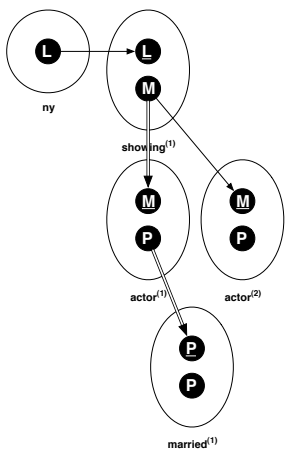


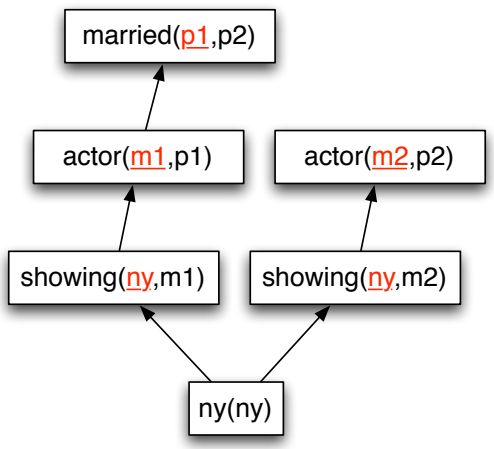
Properties of the pruned d-graph

Properties of deleted arcs in the pruned d-graph

- If $u \curvearrowright v$ is deleted, then each tuple in the answer has an extraction forest not containing $u \curvearrowright v$
- If $u \curvearrowright v$ is not deleted, then there is a database such that the answer contains a tuple whose extraction forests all contain $u \curvearrowright v$







Main results

Theorem: relevant sources

Relevant sources are exactly those appearing in the pruned d-graph

Main results

Theorem: relevant sources

Relevant sources are exactly those appearing in the pruned d-graph

Tractability result

- The algorithm performs a visit of the d-graph, visiting all edges plus some "neighbours" for every node
- ⋆ polynomial time complexity in the size of the d-graph

Extensions

General CQs

- The same technique cannot be extended to boolean CQs with cycles in the dependency graph
  - A relation r might be relevant to obtain all possible values (interesting for non-boolean queries) for some attributes
  - but maybe a value can be obtained without r
- A fortiori, this technique does not extend to general CQs

Extensions

General CQs

- The same technique cannot be extended to boolean CQs with cycles in the dependency graph
  - A relation r might be relevant to obtain all possible values (interesting for non-boolean queries) for some attributes
  - but maybe a value can be obtained without r
- A fortiori, this technique does not extend to general CQs

- Stability can be reduced to non-relevance
  - Testing relevance for CQs is coNP-hard

Extensions

"Natural" CQs

- The d-graph technique can be extended to CQs obtained as the Cartesian product of non-boolean connected queries
  - Connected: every atom has a join with at least another atom in the query
  - Non-boolean: there is at least one variable in the head
  - Very natural sub-class of CQs
  - Problem with boolean CQs isolated in each connected part

Extensions

"Natural" CQs

- The d-graph technique can be extended to CQs obtained as the Cartesian product of non-boolean connected queries
  - Connected: every atom has a join with at least another atom in the query
  - Non-boolean: there is at least one variable in the head
  - Very natural sub-class of CQs
  - Problem with boolean CQs isolated in each connected part

Adding safe negation

- Safe negation can be added with no trouble
  - Negated atoms are immaterial to access limitations

Extensions

Datalog

- Determining relevance for Datalog queries is undecidable
  - Query containment in Datalog is undecidable
  - Query containment in Datalog can be reduced to relevance

Minimality of query plans

Minimality of query plans

Minimality of query plans

Query plan

A deterministic program respecting the access limitations

Minimality of query plans

Query plan

A deterministic program respecting the access limitations

$\forall$-minimality (strong)

A query plan Π is $\forall$-*minimal* iff, for every database D for $\mathcal{S}$, $Acc(D, \Pi) \subseteq Acc(D, \Pi')$ for every query plan Π' of Q.

$Acc(D, \Pi)$ is the set of accesses to relations done by Π over D.

Minimality of query plans

Query plan

A deterministic program respecting the access limitations

$\forall$-minimality (strong)

A query plan Π is $\forall$-*minimal* iff, for every database D for $\mathcal{S}$, $Acc(D, \Pi) \subseteq Acc(D, \Pi')$ for every query plan Π' of Q.

$Acc(D, \Pi)$ is the set of accesses to relations done by Π over D.

Proposition

$\forall$-minimality does not always exist.

Example: $q(X) \leftarrow r_1(X), r_2(Y)$ on schema $\{r_1^o, r_2^o\}$

Weaker minimality of plans

Preliminary criterion

$\Pi' \prec \Pi$ whenever, for every database D, $Acc(D, \Pi') \subseteq Acc(D, \Pi)$ and there is a database D' such that $Acc(D', \Pi') \subset Acc(D', \Pi)$.

$\prec$-minimality

Π is $\prec$-*minimal* iff for no query plan Π'' for Q it holds $\Pi'' \prec \Pi$.

Observations

- A $\prec$-minimal plan always exists for every query
- It is unique iff a $\forall$-minimal plan exists
  - in this case they coincide

Results on $\prec$-minimality

- A $\prec$-minimal plan is always derivable from the pruned d-graph
  - For "natural" CQs
- Plans can be expressed in Datalog
  - with left-to-right execution of body atoms
  - $\star$ plus some ad-hoc strategies required for evaluation

Results on $\prec$-minimality

- A $\prec$-minimal plan is always derivable from the pruned d-graph
  - For "natural" CQs
- Plans can be expressed in Datalog
  - with left-to-right execution of body atoms
  - ⋆ plus some ad-hoc strategies required for evaluation

Fast-failing strategy

Stop evaluation as soon as the result is certainly empty:

- Some of the caches are known to be empty, or
- Some of the joins in the query are known to fail

Query rewriting with views and constraints

Query rewriting with views and constraints

Query rewriting with views and constraints

- Very general setting given in [Deutsch et al. TCS'07]:
  - UCQ$^{\neg}$ query q
  - Set of views V
  - Set of access limitations Λ
  - Set of integrity constraints Σ
- Problem $\{q, V, \Lambda, \Sigma\}$: rewriting queries using views in the presence of access patterns and integrity constraints

Query rewriting with views and constraints

- Very general setting given in [Deutsch et al. TCS'07]:
  - UCQ$^\neg$ query q
  - Set of views V
  - Set of access limitations Λ
  - Set of integrity constraints Σ
- Problem $\{q, V, \Lambda, \Sigma\}$: rewriting queries using views in the presence of access patterns and integrity constraints
- Interest in finding rewritings providing
  - the maximally contained answer
  - the minimally containing answer

Query rewriting with views and constraints

- Possible reductions:
  - Encoding the views into the constraints: $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, \Lambda, \Sigma'\}$

Query rewriting with views and constraints

- Possible reductions:
  - Encoding the views into the constraints:
    $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, \Lambda, \Sigma'\}$
  - Encoding the access limitations into the constraints:
    $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, V, \Sigma''\}$

Query rewriting with views and constraints

- Possible reductions:
  - Encoding the views into the constraints:
    $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, \Lambda, \Sigma'\}$
  - Encoding the access limitations into the constraints:
    $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, V, \Sigma''\}$
- Finally, only the query and the constraint remain:
  $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, \Lambda, \Sigma'\} \rightsquigarrow \{q, \Sigma'''\}$

Query rewriting with views and constraints

- Possible reductions:
  - Encoding the views into the constraints:
    $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, \Lambda, \Sigma'\}$
  - Encoding the access limitations into the constraints:
    $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, V, \Sigma''\}$
- Finally, only the query and the constraint remain:
  $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, \Lambda, \Sigma'\} \rightsquigarrow \{q, \Sigma'''\}$
  $\{q, V, \Lambda, \Sigma\} \rightsquigarrow \{q, V, \Sigma''\} \rightsquigarrow \{q, \Sigma''''\}$

Query rewriting with views and constraints

Theorem ([Deutsch et al. TCS'07])

Stability is NP-complete in the size of the query for fixed views and inclusion constraints over UCQs and Π_2^P-complete for UCQ$^\neg$s

- The above result uses a chasing technique and holds only for those cases in which the chase exists
- Checking whether the chase exists is undecidable
- Fairly general sufficient conditions exist, though

Query containment under access limitations

Query containment under access limitations

Query containment under access limitations

Query containment under access limitations

The containment problem

Notation

- Conjunctive queries q_1, q_2
- Relational schema $\mathcal{S}$
- $q(D)$: answer to q evaluated on database D

The containment problem

Notation

- Conjunctive queries q_1, q_2
- Relational schema $\mathcal{S}$
- $q(D)$: answer to q evaluated on database D

Containment

Containment $q_1 \subseteq q_2$ holds if for every database D for $\mathcal{S}$ we have

$$q_1(D) \subseteq q_2(D)$$

The containment problem (cont'd)

Containment is useful for:

- query minimization
  - used to decide stability of a query
- optimization of query execution
- in general, optimization of formulas
  - e.g., containment can be used to simplify the evaluation of integrity constraints during integrity checking
- ...

Conjunctive query containment: algorithm

1. freeze body(q_1) and head(q_1) by turning each variable into a distinct (fresh) constant
2. evaluate q_2 over the frozen body of q_1
3. $q_1 \subseteq q_2$ iff the evaluation returns the frozen head of q_1

Testing containment amounts to checking the existence of a query homomorphism from q_2 to q_1 [Chandra & Merlin 1977].

Example

From [Ullman 1997]

$$
\begin{aligned}
q_1 : \quad & p(X, Z) \quad \leftarrow \quad a(X, Y), a(Y, Z) \\
q_2 : \quad & p(X, Z) \quad \leftarrow \quad a(X, U), a(V, Z)
\end{aligned}
$$

Example

From [Ullman 1997]

$$\begin{aligned} q_1 : \quad & p(X, Z) \quad \leftarrow \quad a(X, Y), a(Y, Z) \\ q_2 : \quad & p(X, Z) \quad \leftarrow \quad a(X, U), a(V, Z) \end{aligned}$$

Frozen body(q_1):

$$\begin{aligned} a(0, 1) \leftarrow \\ a(1, 2) \leftarrow \end{aligned}$$

Example

From [Ullman 1997]

$$\begin{array}{lrcl} q_1: & p(X,Z) & \leftarrow & a(X,Y), a(Y,Z) \\ q_2: & p(X,Z) & \leftarrow & a(X,U), a(V,Z) \end{array}$$

Frozen body(q_1):

$$\begin{array}{l} a(0,1) \leftarrow \\ a(1,2) \leftarrow \end{array}$$

Frozen head(q_1): $p(0,2) \leftarrow$

Example (contd.)

Applying q_2 to the frozen body(q_1), we find an answer substitution:

$$X \to 0,\ U \to 1,\ V \to 1,\ Z \to 2$$

that yields $p(0, 2)$ which is the frozen head of q_1.

Example (contd.)

Applying q_2 to the frozen body(q_1), we find an answer substitution:

$$X \to 0,\ U \to 1,\ V \to 1,\ Z \to 2$$

that yields $p(0, 2)$ which is the frozen head of q_1.
Therefore $q_1 \subseteq q_2$.

Example (contd.)

Applying q_2 to the frozen body(q_1), we find an answer substitution:

$$X \to 0,\ U \to 1,\ V \to 1,\ Z \to 2$$

that yields $p(0, 2)$ which is the frozen head of q_1.
Therefore $q_1 \subseteq q_2$.

Note

The frozen body of q_1 is a representative of (a piece of) all databases that provide an answer to q_1

The containment problem with access limitations

Notation

- Conjunctive queries q_1, q_2
- Relational schema $\mathcal{S}$ with limitations Λ
- Initial constants $I \supseteq \text{const}(q_1) \cup \text{const}(q_2)$
- $\text{ans}(q, \mathcal{S}, D, I)$: maximal answer to q evaluated on a schema $\mathcal{S}$ under limitations Λ using initial constants I on database D.

The containment problem with access limitations

Notation

- Conjunctive queries q_1, q_2
- Relational schema $\mathcal{S}$ with limitations Λ
- Initial constants $I \supseteq \text{const}(q_1) \cup \text{const}(q_2)$
- $\text{ans}(q, \mathcal{S}, D, I)$: maximal answer to q evaluated on a schema $\mathcal{S}$ under limitations Λ using initial constants I on database D.

Containment

Containment $q_1 \subseteq_{\Lambda,I} q_2$ under limitations holds if for every database D for $\mathcal{S}$ we have

$$\text{ans}(q_1, \mathcal{S}, D, I) \subseteq \text{ans}(q_2, \mathcal{S}, D, I)$$

Comparing the two containments

$q_1 \subseteq q_2 \models q_1 \subseteq_{\Lambda,I} q_2$

Can be seen by applying homomorphisms:

- If there are extractible instances of the atoms in body(q_1)
- then there are extractible instances of the atoms in body(q_2)
  - that produce the same head

Comparing the two containments

$q_1 \subseteq q_2 \models q_1 \subseteq_{\Lambda,I} q_2$

Can be seen by applying homomorphisms:

- If there are extractible instances of the atoms in body(q_1)
- then there are extractible instances of the atoms in body(q_2)
  - that produce the same head

$q_1 \subseteq_{\Lambda,I} q_2 \not\models q_1 \subseteq q_2$

Example:

- Schema: $r_1^{io}(A, X), r_2^{oo}(B, X)$
- q_1: $q_1(X) \leftarrow r_1(A, X)$ $\quad$ q_2: $q_2(X) \leftarrow r_2(B, X)$
- We have $q_1 \subseteq_{\Lambda,I} q_2$, e.g., for $I = \emptyset$
- but clearly $q_1 \not\subseteq q_2$

The containment problem with access limitations (cont'd)

- A conjunctive query q under access limitations can be rewritten as an executable (recursive) Datalog program that retrieves the maximal answer to q
- Therefore, checking containment amounts to checking containment between two Datalog programs
- Not a good idea: Datalog query containment is undecidable
  - however, programs have a special form

Decidability of the problem

- Query containment is actually decidable under access limitations even for more expressive classes than CQs

Theorem (Millstein et al. JCSS'03)

Given

- *a (potentially recursive) datalog program* Q_1
- *a nonrecursive datalog program* Q_2
- *a set* Λ *of one access pattern adornment for each relation*

determining whether $Q_1 \subseteq_{\Lambda,I} Q_2$ *is decidable.*

Reducing containment to monadic datalog

- Conjunctive query containment under access limitation can be reduced to containment between monadic datalog programs
- A datalog program is monadic if its recursive predicates are monadic

Theorem (Li & Chang TODS'01)

Containment of connection queries under access limitations is decidable

- This easily extends to general conjunctive queries

Theorem (Li & Chang TODS'01)

Containment of CQs under access limitations is decidable

- Same complexity (2EXPTIME) as containment between monadic datalog programs

Direct approach to containment checking

- A direct approach to query containment under access limitations can be given
- This gives also a slightly improved complexity bound

The chase

How can we find a representative of (a piece of) all databases that provide an answer to q_1 in the case with access limitations?

The chase

How can we find a representative of (a piece of) all databases that provide an answer to q_1 in the case with access limitations?

The chase

Used for:

- query containment under relational dependencies [Johnson & Klug 1984]
- implication of relational dependencies
- querying incomplete data
- data integration and data exchange
- ...

The chase

How can we find a representative of (a piece of) all databases that provide an answer to q_1 in the case with access limitations?

The chase

Used for:

- query containment under relational dependencies [Johnson & Klug 1984]
- implication of relational dependencies
- querying incomplete data
- data integration and data exchange
- ...

We have a different version...

The crayfish-chase

The crayfish-chase

The crayfish-chase

Idea

- We construct an extraction tree producing exactly the frozen body of q_1
- We proceed backwards (moving as a crayfish)
  - We start from the frozen body of q_1
  - We add tuples that provide values used in the input fields of the tuples in the frozen body of q_1
  - Then we add tuples that provide values for the input fields of the previous tuples
  - and so on
  - we continue until we "close off" with tuples for relations that have no input field

The crayfish-chase: example

Schema

$r_1^{iio}(A, B, A)$
$r_2^{io}(A, B)$
$r_3^{o}(A)$

Query

$q(X_2) \leftarrow r_1(a, X_1, X_2)$

answerable but non-executable

- W.l.o.g., we use constant-free queries
- First the query is frozen, then "expanded" by chasing
  - ⋆ Special constants ζ_i denote unknown values (labelled nulls)

The crayfish-chase: example (cont'd)

Query with constants eliminated

$$q(X_2) \leftarrow r_1(X_a, X_1, X_2), \ell_a(X_a)$$

- ℓ_a is an aux. predicate with extension $\{\langle a \rangle\}$
- the chase starts from the frozen body

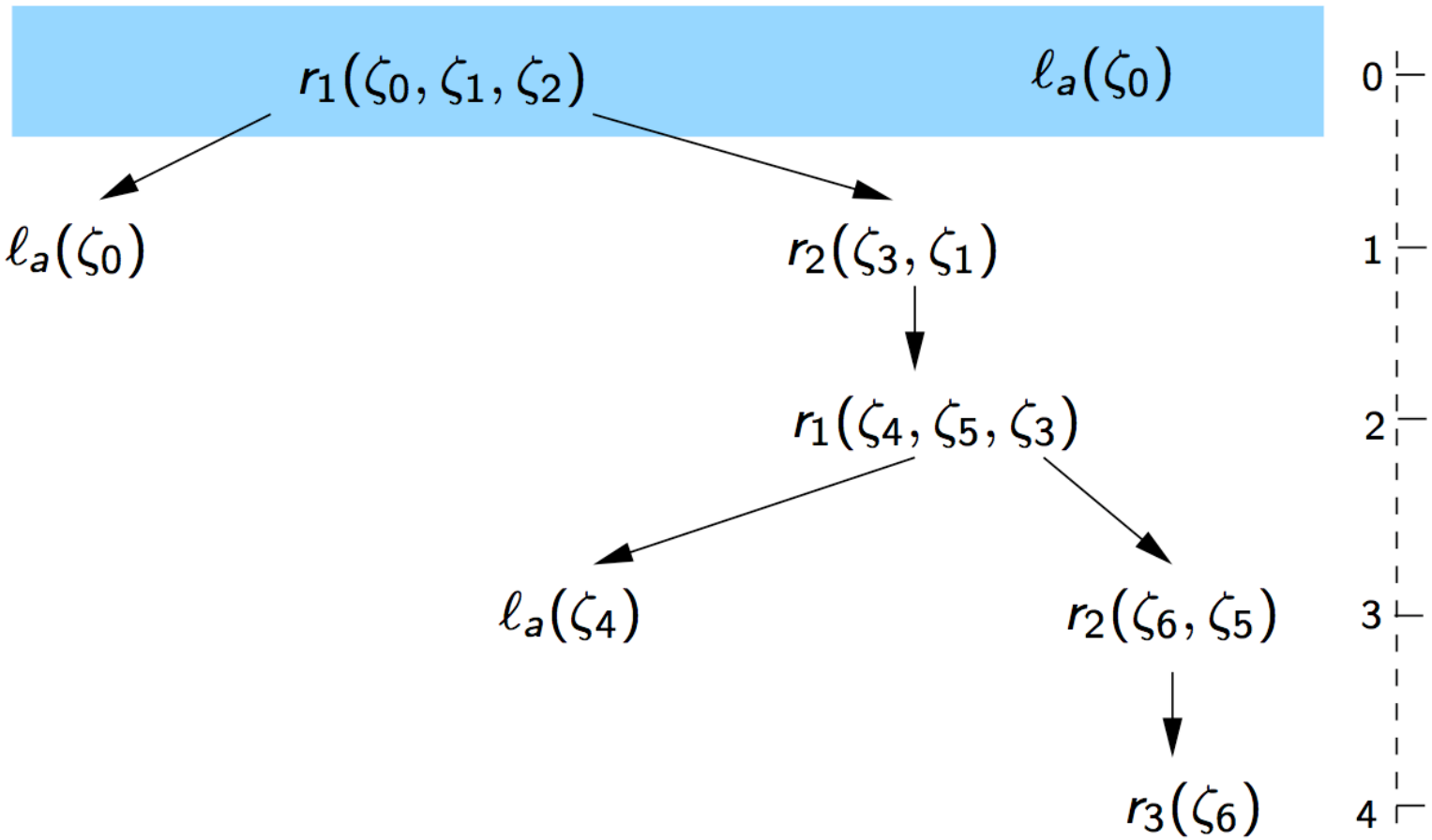
Frozen head

$$q(\zeta_2)$$

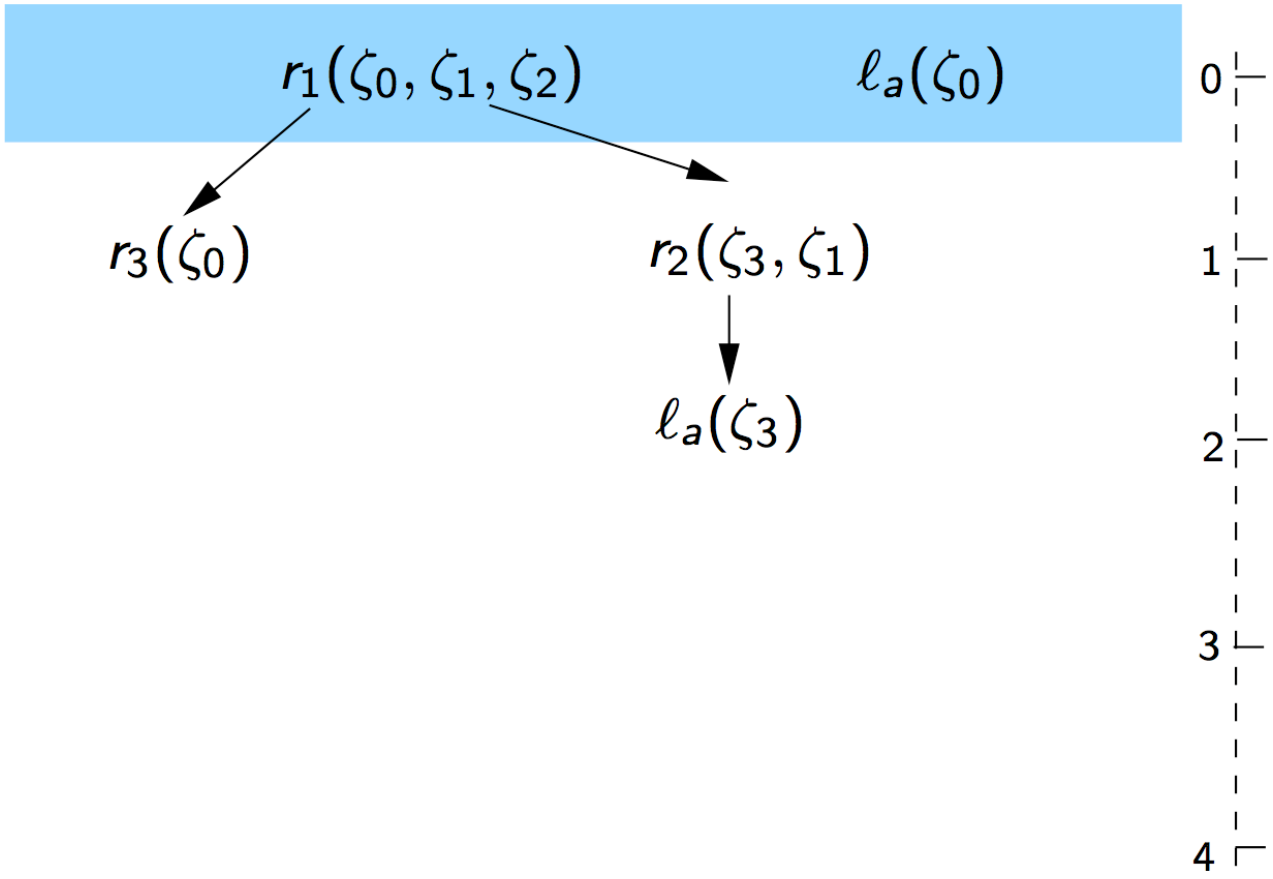
Frozen body

$$r_1(\zeta_0, \zeta_1, \zeta_2), \ell_a(\zeta_0)$$

Example of instance in the crayfish-chase



Example: another instance in the crayfish-chase



The crayfish-chase

Features

- Each constructible extraction tree producing the frozen body of q is a database of the crayfish chase
- The crayfish chase is
  - a set of databases, denoted cchase$(q, \mathcal{S}, I)$
  - a forest of extraction trees, layered in levels
- Every database represents one way of "extracting" an answer tuple
- Thus, the chase may serve as a tool for containment

Main property of the crayfish-chase

Theorem

$q_1 \subseteq_{\Lambda,I} q_2$ if and only if for every database $C \in \text{cchase}(q_1, \mathcal{S}, I)$

$$\text{frozen\_head}(q_1) \in q_2(C)$$

(frozen_head(q_1) is the same in every DB in the chase)

Main property of the crayfish-chase

Theorem

$q_1 \subseteq_{\Lambda,I} q_2$ if and only if for every database $C \in \text{cchase}(q_1, \mathcal{S}, I)$

$$\text{frozen\_head}(q_1) \in q_2(C)$$

(frozen_head(q_1) is the same in every DB in the chase)

C is an extraction tree: q_2 can be evaluated w/o access limitations

Main property of the crayfish-chase

Theorem

$q_1 \subseteq_{\Lambda,I} q_2$ if and only if for every database $C \in \text{cchase}(q_1, \mathcal{S}, I)$

$$\text{frozen\_head}(q_1) \in q_2(C)$$

(frozen_head(q_1) is the same in every DB in the chase)

C is an extraction tree: q_2 can be evaluated w/o access limitations

Warning

Not yet a strategy for deciding containment!

- there may be an infinite number of databases in a chase
- there is no bound on the size of databases in the chase

Decidability

Theorem

IF there exists a finite database $C \in \text{cchase}(q_1, \mathcal{S}, I)$ such that $q_1(C) \not\subseteq q_2(C)$,

THEN there exists another finite database $C' \in \text{cchase}(q_1, \mathcal{S}, I)$ such that

1. $q_1(C') \not\subseteq q_2(C')$, and
2. C' has maximum level $\delta = 2 \cdot |\mathcal{S}| + |q_2| - 3$

Decidability

Theorem

IF there exists a finite database $C \in \text{cchase}(q_1, \mathcal{S}, I)$ such that $q_1(C) \not\subseteq q_2(C)$,

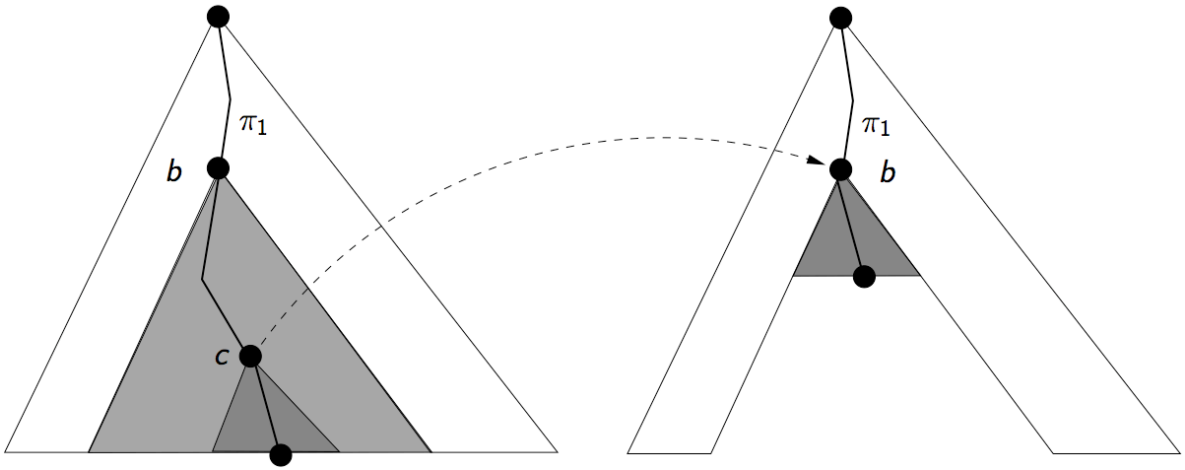
THEN there exists another finite database $C' \in \text{cchase}(q_1, \mathcal{S}, I)$ such that

1. $q_1(C') \not\subseteq q_2(C')$, and
2. C' has maximum level $\delta = 2 \cdot |\mathcal{S}| + |q_2| - 3$

Consequence

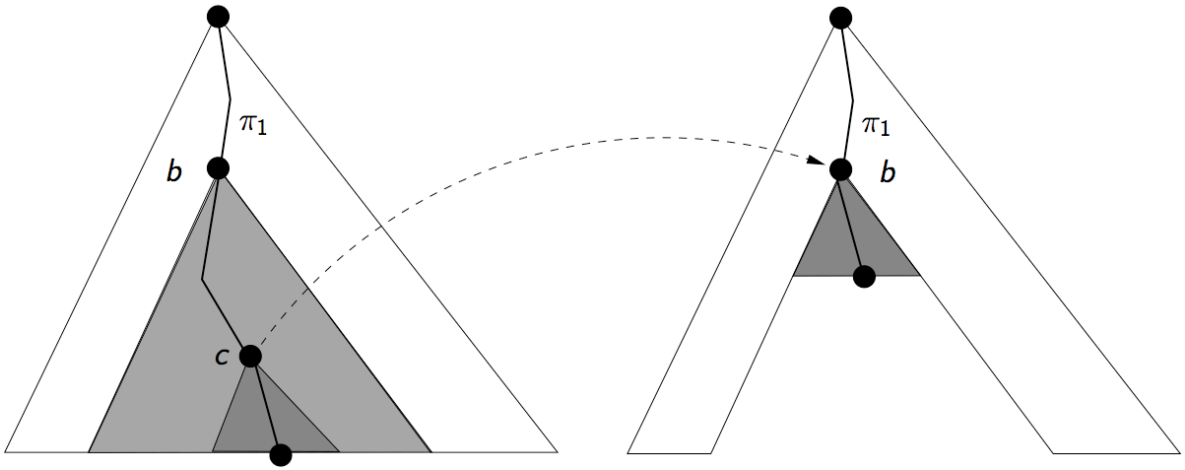
It is sufficient to check all databases in the chase up to a certain number of levels

Idea of the proof: iterative subtree replacement



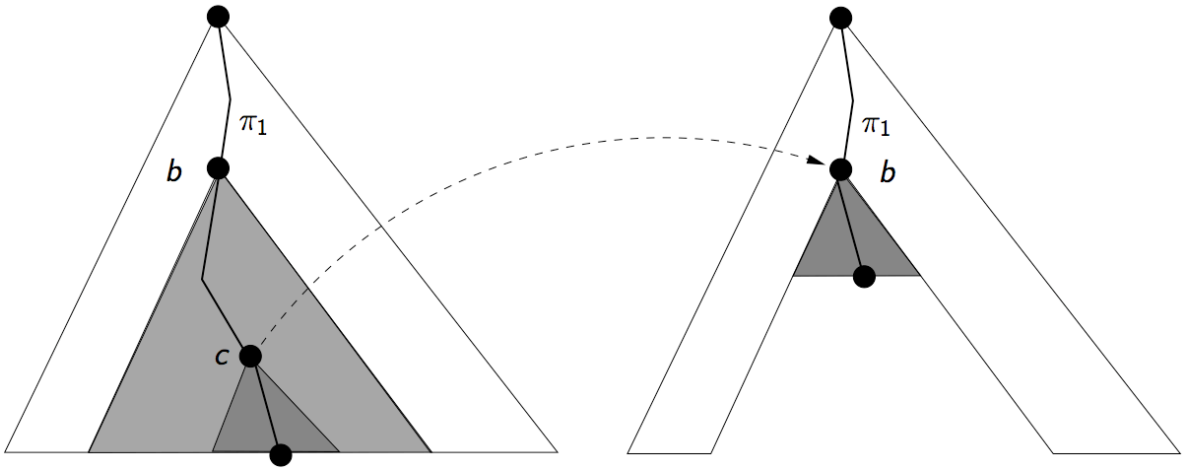
- Take a counterexample C in cchase$(q_1, \mathcal{S}, I)$

Idea of the proof: iterative subtree replacement



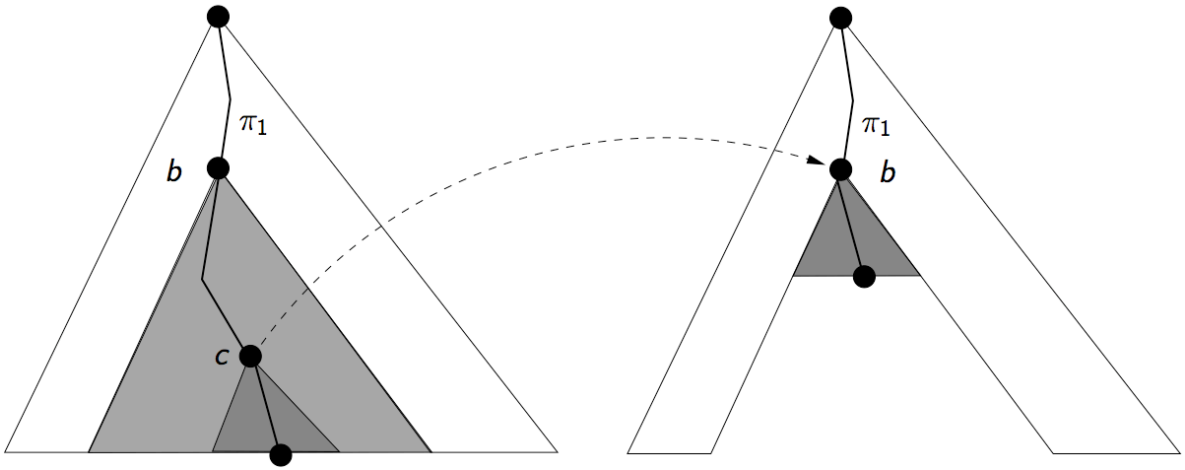
- Take a counterexample C in cchase$(q_1, \mathcal{S}, I)$
- if C exceeds the level δ, "shorten" it

Idea of the proof: iterative subtree replacement



- Take a counterexample C in cchase$(q_1, \mathcal{S}, I)$
- if C exceeds the level δ, "shorten" it
- C is "shortened" by subtree replacement

Idea of the proof: iterative subtree replacement



- Take a counterexample C in cchase$(q_1, \mathcal{S}, I)$
- if C exceeds the level δ, "shorten" it
- C is "shortened" by subtree replacement
- at each replacement, we get another counterexample

Complexity

Theorem

The complexity of checking containment of conjunctive queries under access limitations is in co-NEXPTIME.

Complexity

Theorem

The complexity of checking containment of conjunctive queries under access limitations is in co-NEXPTIME.

Proof sketch

1. guess $C \in \text{cchase}(q_1, \mathcal{S}, I)$ of depth less than the sufficient one; size $O(W^\delta)$ (W: max. arity)
2. evaluate q_2 over C: feasible in polynomial time in C and det. exp. time in q_2
3. if no counterexample to containment is found, then containment holds (otherwise containment does not hold)

Dynamic optimization

Dynamic optimization

Using constraints for dynamic optimization

Constraints considered in [C., Calvanese, M. JUCS'09]

- Functional dependencies (FD): $s : A \rightarrow B$
  - satisfied in D if, $\forall t_1, t_2 \in s^D$, $t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$
- Simple full-width inclusion dependencies (SFWID): $s_1 \subseteq s_2$
  - Satisfied in D if $s_1^D \subseteq s_2^D$
  - Special case of inclusion dependencies
  - Relevant to model sources with several access patterns

Using constraints for dynamic optimization

Constraints considered in [C., Calvanese, M. JUCS'09]

- Functional dependencies (FD): $s : A \rightarrow B$
  - satisfied in D if, $\forall t_1, t_2 \in s^D$, $t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$
- Simple full-width inclusion dependencies (SFWID): $s_1 \subseteq s_2$
  - Satisfied in D if $s_1^D \subseteq s_2^D$
  - Special case of inclusion dependencies
  - Relevant to model sources with several access patterns

Notation

$\mathcal{A}(s)$: set of all attributes of source s
$\mathcal{I}(s)$: set of input attributes of source s

Using constraints for dynamic optimization

Idea for dynamic optimization

- Store accesses and answer tuples in caches
- Avoid accesses recognized as irrelevant thanks to the constraints
  - The access may already conflict with the extracted tuples on some FD that holds in the database
  - Or it may coincide with an access already made
- Requires deciding implication of FDs in the presence of FDs and SFWIDs
  - Implication is undecidable for FDs and general inclusion dependencies [Chandra & Vardi, SIAM J.Comp.'85]

Implication of FDs and SFWIDs

Let Γ be a set of FDs and SFWIDs

Results

- Implication of SFWIDs from Γ only depends on SFWIDs
  - FDs have no impact on the implication of SFWIDs
  - The reflexivity and transitivity rules are sound and complete

Implication of FDs and SFWIDs

Let Γ be a set of FDs and SFWIDs

Results

- Implication of SFWIDs from Γ only depends on SFWIDs
  - FDs have no impact on the implication of SFWIDs
  - The reflexivity and transitivity rules are sound and complete
- Implication of a FD on a source s from Γ:
  - consider only sources s_i for which $s \subseteq s_i$ is implied
  - assert on s all FDs holding on those sources s_i
  - decide implication of FDs using only the FDs on s (plus the asserted ones) with Armstrong's rules

Inference rules for SFWIDs and FDs

1. For every source s and all sets of attributes $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s)$,
   if $\mathbf{A} \subseteq \mathbf{B}$, then $s : \mathbf{B} \rightarrow \mathbf{A}$.
2. For every source s and all sets of attributes $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathcal{A}(s)$,
   if $s : \mathbf{A} \rightarrow \mathbf{B}$, then $s : \mathbf{AC} \rightarrow \mathbf{BC}$.
3. For every source s and all sets of attributes $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathcal{A}(s)$,
   if $s : \mathbf{A} \rightarrow \mathbf{B}$ and $s : \mathbf{B} \rightarrow \mathbf{C}$, then $s : \mathbf{A} \rightarrow \mathbf{C}$.
4. For every source s,
   $s \subseteq s$.
5. For all sources s_1, s_2, s_3,
   if $s_1 \subseteq s_2$ and $s_2 \subseteq s_3$, then $s_1 \subseteq s_3$.
6. For all sources s_1, s_2 and sets of attr. $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s_1) = \mathcal{A}(s_2)$,
   if $s_1 \subseteq s_2$ and $s_2 : \mathbf{A} \rightarrow \mathbf{B}$, then $s_1 : \mathbf{A} \rightarrow \mathbf{B}$.

Results on implication

Theorem

The inference rules $1-6$ *are sound and complete for implication of FDs and SFWIDs.*

Results on implication

Theorem

The inference rules $1-6$ *are sound and complete for implication of FDs and SFWIDs.*

Theorem

Implication of SFWIDs and FDs can be decided in polynomial time.

Results on implication

Theorem

The inference rules $1-6$ *are sound and complete for implication of FDs and SFWIDs.*

Theorem

Implication of SFWIDs and FDs can be decided in polynomial time.

Theorem

Finite and unrestricted implication are equivalent for SFWIDs and FDs. (If there is a counterexample, then there is a finite one)

Example of dynamic optimization

Static info

Schema: $s^{iio}(A, B, C)$
FD: $s : A \rightarrow B, C$

Example of dynamic optimization

Static info

Schema: $s^{iio}(A, B, C)$
FD: $s : A \rightarrow B, C$

Dynamic context

D is the database state
t is a tuple that has been extracted from s^D such that $t[A] = a$
Suppose we have some value b for attribute B

- Accessing s^D with $A = a, B = b$ is useless:
  - it can either provide t alone, if $b = t[B]$
  - or no tuple, if $b \neq t[B]$
- because of the FD!

Dynamic relevance

Access

An access (to a source s) is a selection query on s
A binding (for s) is a tuple of constants used for the input attributes in an access to s

Dynamic relevance

Let $\mathcal{T}$ be a set of tuples for sources $s_1, \ldots, s_n$.
Let Γ be a set of FDs and SFWIDs. An access is dynamically relevant wrt. Γ and $\mathcal{T}$ if there is a database $D \models \Gamma$ such that:

- $\mathcal{T}$ is the set of tuples extracted from D with some accesses
- the access extracts from D at least
  - one tuple not in $\mathcal{T}$ (tuple relevance), or
  - one constant not in $\mathcal{T}$ (binding relevance)

$\mathcal{T}$ represents the set of already extracted tuples

Dynamic relevance and keys

Result

If a subset of the input attributes is a key, an access α is dynamically tuple-relevant iff no already extracted tuple coincides with α's binding on the key attributes.

- Let γ be the FD

$$s : \mathbf{K} \rightarrow \mathcal{A}(s)$$

  with $\mathbf{K} \subseteq \mathcal{I}(s)$.
- Let b be a binding for s and $\mathbf{T}_s$ a set of tuples that satisfies γ.

Accessing s using b is dynamically relevant wrt. γ and $\mathbf{T}_s$ iff there exists no tuple $t \in \mathbf{T}_s$ such that $b[\mathbf{K}] = t[\mathbf{K}]$.

Dynamic relevance and FDs

Result

An access is not dynamically tuple-relevant if:

- it has the same binding as some extracted tuple
- it violates a FD together with some extracted tuple

Let γ be the FD

$$s : \mathbf{A} \rightarrow \mathbf{B}$$

with $\mathbf{A} \subseteq \mathcal{I}(s)$. Let b be a binding for s and $\mathbf{T}_s$ a set of tuples. Then accessing s using b is dynamically relevant wrt γ and $\mathbf{T}_s$ iff

1. no tuple $t \in \mathbf{T}_s$ is such that $t[\mathcal{I}(s)] = b[\mathcal{I}(s)]$, and
2. no tuple $t \in \mathbf{T}_s$ is such that $t[\mathbf{A}] = b[\mathbf{A}]$ and $t[\mathbf{B} \cap \mathcal{I}(s)] \neq b[\mathbf{B} \cap \mathcal{I}(s)]$.

Example of dynamic optimization using SFWIDs

Static info

$s_1^{iio}(Code, Surname, City)$
$s_2^{ooi}(Code, Surname, City)$

SFWID: $s_1 \subseteq s_2$
FD: $s_2 : Code \rightarrow Surname, City$
Constants: *Rome* and *Kyoto*

Dynamic context

$s_1^D = s_2^D =$

Code	*Surname*	*City*
2	*brown*	*sidney*
5	*williams*	*london*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

Example of dynamic optimization using SFWIDs

Static info

$s_1^{iio}(Code, Surname, City)$
$s_2^{ooi}(Code, Surname, City)$

SFWID: $s_1 \subseteq s_2$
FD: $s_2 : Code \rightarrow Surname, City$
Constants: *Rome* and *Kyoto*

Dynamic context

$s_1^D = s_2^D =$

Code	*Surname*	*City*
2	*brown*	*sidney*
5	*williams*	*london*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

We can extract from s_2^D

Code	*Surname*	*City*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

Example of dynamic optimization using SFWIDs

Static info

$s_1^{iio}(Code, Surname, City)$
$s_2^{ooi}(Code, Surname, City)$

SFWID: $s_1 \subseteq s_2$
FD: $s_2 : Code \rightarrow Surname, City$
Constants: *Rome* and *Kyoto*

Dynamic context

$s_1^D = s_2^D =$

Code	*Surname*	*City*
2	*brown*	*sidney*
5	*williams*	*london*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

We can extract from s_2^D

Code	*Surname*	*City*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

Any access to s_1^D with the new extracted constants is useless to discover new constants

Dynamic relevance and SFWIDs

Interaction of FDs and SFWIDs

Let Γ be the following set of dependencies:

$$s_1 \subseteq s_2$$
$$s_2 : \mathbf{C} \rightarrow \mathbf{D}$$

with $\mathbf{C} \subseteq \mathcal{I}(s_1)$ and $\mathbf{D} \supseteq \mathcal{I}(s_2)$.
Let b be a binding for s_1 and $\mathbf{T_{s_2}}$ a set of tuples.
Accessing s_1 with b is dynamically binding-relevant with respect to Γ and $\mathbf{T_{s_2}}$ iff there exists no tuple $t \in \mathbf{T_{s_2}}$ such that $t[\mathbf{C}] = b[\mathbf{C}]$.

Main result

Theorem

Let $\mathcal{S}$ be a schema of sources with fixed maximum arity. Let Γ be a set of FDs and SFWIDs on sources in $\mathcal{S}$.
Dynamic relevance of an access with respect to Γ can be checked in polynomial time in the number of

- *dependencies in Γ*
- *attributes in $\mathcal{S}$*
- *tuples already extracted from sources in $\mathcal{S}$*

Conclusion

Conclusion

Conclusion

Conclusion

Conclusions

- Different approaches to the deep Web
  - Surfacing
  - Vertical integration
  - Large-scale on-the-fly data integration
- Different techniques for query answering under access limitations
  - finding the exact answer
  - finding best-effort approximations of the answer (maximally contained or minimally containing)
- Static optimization
  - exclusion of irrelevant sources
  - minimization of the accesses

Conclusions

- Conjunctive query containment under access limitations
  - reduction to decidable (limitation-free) cases of containment
  - direct analysis with the crayfish-chase
- Query rewriting with views under access limitations and integrity constraints
  - reduction to query answering under integrity constraints
- Dynamic optimization
  - excluding accesses deemed irrelevant during query answering
    - by dependencies on the schema (FDs and SFWIDs), and
    - knowledge of the already extracted tuples

Research directions

Taming the Web

- Improving on-the-fly data extraction
  - HTML parsing
  - Linguistic aspects
- Schema inference

Static optimization

- More expressive queries than "natural" CQs and connection queries
- Sources with multiple access patterns
- Query containment
  - Tight bounds, expressive query classes, constraints

Acknowledgments

Thanks to:

- Diego Calvanese
- Bertram Ludäscher
- The Search Computing (SeCo) project
- EPSRC "Schema mappings..." project

References


Andrea Calì, Diego Calvanese, and Davide Martinenghi.
Dynamic query optimization under access limitations and dependencies.
Journal of Universal Computer Science, 15(21):33–62, 2009.


Andrea Calì and Davide Martinenghi.
Conjunctive Query Containment under Access Limitations.
In *Proceedings of the Twentyseventh International Conference on Conceptual Modeling (ER 2008)*, pages 326–340, 2008.


Andrea Calì and Davide Martinenghi.
Querying data under access limitations.
In *Proceedings of the Twentyfourth IEEE International Conference on Data Engineering (ICDE 2008)*, pages 50–59. IEEE Computer Society Press, 2008.

References


Kevin Chen-Chuan Chang, Bin He and Zhen Zhang.
Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web.
CIDR 2005, pages 44–55.


Alin Deutsch, Bertram Ludäscher, and Alan Nash.
Rewriting queries using views with access patterns under integrity constraints.
Theoretical Computer Science, 371(3):200–226, 2007.


Oliver M. Duschka and Alon Y. Levy.
Recursive plans for information gathering.
In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 778–784, 1997.

References


Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu.
Query optimization in the presence of limited access patterns.
In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 311–322, 1999.


Bin He, Mitesh Patel, Zhen Zhang and Kevin Chen-Chuan Chang.
Accessing the deep web.
Communications of the ACM, 50(5): 94–01, 2007.


Chen Li.
Computing complete answers to queries in the presence of limited access patterns.
Very Large Database Journal, 12(3):211–227, 2003.

References


Chen Li and Edward Chang.
Answering queries with useful bindings.
ACM Transactions on Database Systems, 26(3):313–343, 2001.


Bertram Ludäscher and Alan Nash.
Processing union of conjunctive queries with negation under limited access patterns.
In *Proceedings of the Ninth International Conference on Extending Database Technology (EDBT 2004)*, pages 422–440, 2004.


Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen and Alon Y. Halevy.
Google's Deep Web crawl.
In *PVLDB* 1(2):1241–1252, 2008.

References


Todd D. Millstein, Alon Y. Halevy, and Marc Friedman.
Query containment for data integration systems.
Journal of Computer and System Sciences, 66(1):20–39, 2003.


Alan Nash and Bertram Ludäscher.
Processing first-order queries under limited access patterns.
In *Proceedings of the Twentythird ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2004)*, pages 307–318, 2004.


Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman.
Answering queries using templates with binding patterns.
In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'95)*, 1995.

References


Guizhen Yang, Michael Kifer, and Vinay K. Chaudhri.
Efficiently ordering subgoals with access constraints.
In *Proceedings of the Twentyfifth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 22–22, 2006.


Ramana Yerneni, Chen Li, Hector Garcia-Molina, and Jeffrey D. Ullman.
Computing capabilities of mediators.
In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 443–454, 1999.