Virtual Integration of Heterogeneous Data and Data Model Unification

Query Rewriting Using Views

Sergey Stupnikov

Institute of Informatics Problems, RAS

sstupnikov@ipiran.ru

Outline

- Preliminaries: tables, relations, predicates, queries
- Mediator approach and query rewriting using views
- Global-as-View mediation
- Local-as-View mediation

Tables and Relations

- Database is a collection of facts. Every row of a table corresponds to a fact.
  - "John is enrolled in Databases"
  - "John is enrolled in Data Mining"
  - "Jim is enrolled in Databases"

EnrolledInCourse

studentName	**code**
John	Databases
John	DataMining
Jim	Databases

- n-ary relation $R(a_1: T_1, \ldots, a_n: T_n)$
  - a_i is an attribute of type T_i (Boolean, integer, float, string, …)
  - R is a set of tuples like $t = \langle v_1, v_2, \ldots, v_n \rangle$, every v_i is of type T_i
- A relation corresponds to a table and vice versa
  - EnrolledInCourse(studentName: string, code: string)
  - EnrolledInCourse = { <John, Databases>, <John, DataMining>, <Jim, Databases> }

Relations and Predicates

- Predicate is an n-ary Boolean function
  - EnrolledInCourse(sn, c): Boolean
- A relation corresponds to a predicate and vice versa

EnrolledInCourse(John, Databases) = T [predicate]

<=>

<John, Databases> $\in$ EnrolledInCourse [relation]

First Order Predicate Logic

- FOPL is a language

- Alphabet
  - constants (5, “John”, true, 3.1405, …)
  - variables (x, y, z, …)
  - functional symbols (+, -, *, /, mod, …)
  - predicate symbols (=, ≠, <, >, EnrolledInCourse, …)
  - logical connectives (&, ∨, not, ⇒)
  - quantifiers (∀, ∃)
- Terms (words)
  - constants, variables
  - functional terms
    - 1*2
    - “John” + “Lennon”

FOPL (2)

- Formulae (sentences)
  - atom predicate
    - $x = 5$
    - EnrolledInCourse(y, Databases)
  - compound predicates (connected by a logical connective)
    - EnrolledInCourse(y, Databases) & $x = 5$
  - quantified formulae
    - $\forall$ x (EnrolledInCourse(Jim, x) $\Rightarrow$ x = Databases)
    - $\exists$ y (MasterStudent(y) & EnrolledInCourse(y, Databases))

Valid Closed Formulae

- Closed formula contains no unbounded variables (not bounded by quantifiers)
- Validity
  - [atomic predicate]
    - EnrolledInCourse(v, w) **is valid** $\Leftrightarrow$ <v, w> $\in$ EnrolledInCourse
  - [compound predicates]
    - conjunction
    - implication

A	B	A & B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

  - [quantified formulae]
    - $\forall$ x (F(x)) **is valid** $\Leftrightarrow$ for all constants v, F(v) **is valid**
    - $\exists$ x (F(x)) **is valid** $\Leftrightarrow$ exists constant v, such that F(v) **is valid**

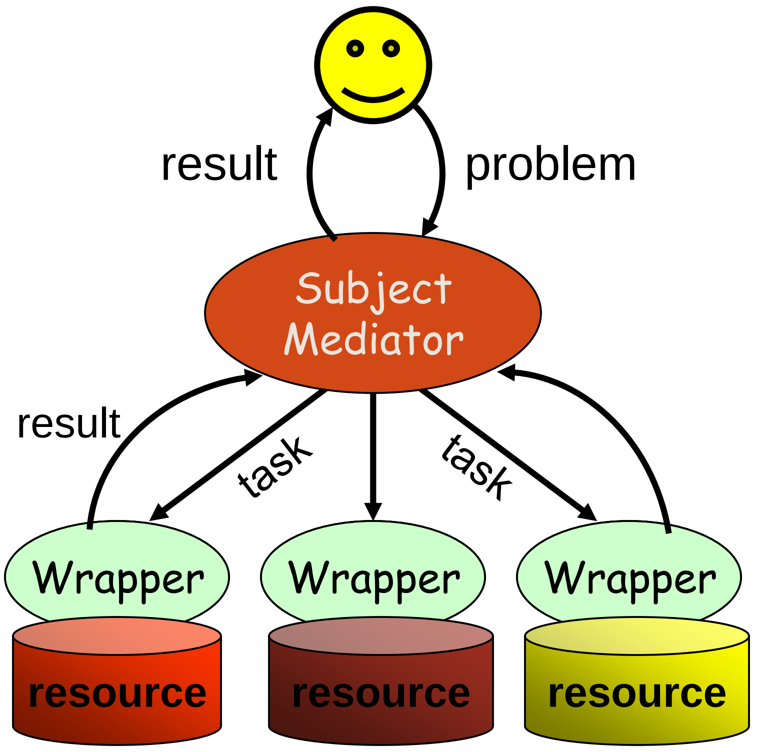
Validity Set for Formulae with Unbounded Variables

- A formula with unbounded variables is actually a predicate
  - $Q(x, y) \Leftrightarrow MasterStudent(x)\ \&\ EnrolledInCourse(x, y)\ \&\ y \neq Databases$
- Validity set for a formulae is a set of all tuples that turns the formula into true
  - $VS(Q(x, y)) =$
  - $\{ \langle v, w \rangle \mid MasterStudent(v)\ \&\ EnrolledInCourse(v, w)\ \&\ w \neq Databases \} =$
  - $\{ \langle John, DataMining \rangle \}$
- Bounding of new predicate with a formula is called a logical query
  - $Q(x, y)$:- $MasterStudent(x), EnrolledInCourse(x, y), y \neq Databases$.
  - Conjunction is usually replaced by comma
- A logical query corresponds to an SQL query
  - SELECT studentName AS x, code AS y INTO Q
  - FROM MasterStudent, EnrolledInCourse
  - WHERE y ≠ Databases

Mediator approach and query rewriting using views

Виртуальная интеграция в предметных посредниках

- Задача формулируется в терминах схемы посредника, затем
- трансформируется в набор подзадач (запросов) к ресурсам, зарегистрированным в посреднике;
- подзадачи исполняются на ресурсах, результаты возвращаются в посредник;
- результаты объединяются и представляются пользователю.



Mediator Approach

- Starts by designing a *global schema* (also called *mediated* schema) that serves as a unique entry point on which global queries are posed by users
- A main issue is then to specify the relationships, namely *semantic mappings*, between the schemas of the data sources and the global schema
  - Based on these mappings, one can answer queries over the global schema using queries over the data sources

Semantic Mappings

- $S_1, \ldots, S_n$ - local schemas of n pre-existing data sources
  - assume each S_i is made of a single relation that we denote also S_i
- Global schema $G = \{ G_1, \ldots, G_m \}$ - *global* relations
- Goal is to specify semantic relations between the S_i and G_j
- Semantic relations examples
  - $G_1 = S_1$
  - $G_2 = S_1 \cup S_2$ [union]
  - $G_3 = S_1 \bowtie S_3$ [join]
    - more flexible $G_3 \supseteq S_1 \bowtie S_3$
    - using containment instead of equality leaves open the possibility for other sources of providing data about G_3
    - $G_3 \supseteq \sigma_{A=\text{"yes"}}(S_4)$ [selection]
  - *global-as-view:* global relations are constrained by views of the local relations

  - $S_4 \subseteq G_1 \bowtie G_3$
  - *local-as-view:* each data source can be specified independently (by owner) of the other sources of the system

Semantic Mappings (2)

- General form: $v(S_1,\ldots,S_n) \subseteq v'(G_1,\ldots,G_m)$
  - v and v' are *views* (query expressions)

- Global-As-View (GAV)
  - $G_i \supseteq V_i(S_1, \ldots , S_n)$
  - each V_i is a view over the local schemas

- Local-As-View (LAV)
  - $S_i \subseteq V_i(G_1, \ldots , G_n)$
  - each V_i is a view over the global schema

Conjunctive queries

- A conjunctive query $q(x_1, ... x_n) \text{ :- } A_1(\vec{u_1}), ..., A_k(\vec{u_k})$
  - A_i is an relation, $u_1,...,u_k$ are tuples of constants and variables
  - each x_i occurs in some u_i
  - $q(x_1,..., x_n)$ is the head of the query
  - $A_1(u_1)$,..., $A_k(u_k)$ is the body of the query
  - x_i are *distinguished variables*
  - other variables are *existential*
- *Answer* of q over I is a tuple $\langle v(x_1),...,v(x_n)\rangle$
  - I is an instance of A_i
  - v is a valuation of x_i, such that for each i, $A_i(v(u_i))$ holds in I
  - $q(I)$ is the set of answers
- Interpretation of a conjunctive query in logical terms

$$\{x_1, ..., x_n \mid \exists y_1, ..., \exists y_m (A_1(\vec{u_1}) \wedge ... \wedge A_k(\vec{u_k}))\}$$

- *Query containment:* $q' \subseteq q \Leftrightarrow \forall I\, (\, q'(I) \subseteq q(I)\,)$

Query Answering using Mediation

- Suppose an instance I of data sources $\{S_1,\ldots,S_n\}$ is given alongside with constraints $v(S_1,\ldots, S_n) \subseteq v'(G_1,\ldots,G_m)$
- The instance J of the global schema $\{G_1,\ldots,G_m\}$is not known
- But is known $v(I(S_1),\ldots,I(S_n)) \subseteq v'(J(G_1),\ldots,J(G_m))$

❑ Given I, an *answer* to a global query q is a fact $q(a)$ that is true in any instance J that together with I satisfies the mapping constraint

Query Answering using Mediation – Example (I)

Global Schema:

- *MasterStudent(studentName)*

Local Schemas:

- S2.Erasmus(student, course, univ) - Erasmus exchange program
- S4.Mundus(program, course) - international master programs

MasterStudent(N) $\supseteq$ *S2.Erasmus(N,C,U), S4.Mundus(P,C)*

- ***I*** Erasmus

student	course	univ
Jim	Databases	Roehampton
John	Data Mining	TU Varna
Jane	Architecture	Turku

Mundus

program	course
Communication Technologies	Databases
Neuroscience	Data Mining
Security	Cyber Security

- J_1

studentName
Jim

J_2

studentName
Jim
John

J_3

studentName
Jim
John
Kate

Query Answering using Mediation – Example (II)

- $q(x)$:- $MasterStudent(x)$

- <Jim> $\in q(I)$
- <John> $\in q(I)$
- <Jane> $\notin q(I)$
- <Kate> $\notin q(I)$

Query Rewriting

- **Problem**: rewrite a user query expressed in the mediated schema into a query expressed in the source schema

- Given a query Q in terms of the mediator schema relations, and descriptions of information sources
- Find a query Q' that uses only the source relations, such that
  - Q' $\subseteq$ Q (Q' is contained in Q), and
  - Q' provides all possible answers to Q given the sources

Conjunctive Query Homomorphism

- $q_1(x_1,\ldots,x_n)$, $q_2(y_1,\ldots,y_n)$ - conjunctive queries
- *homomorphism from q_2 to q_1* is a mapping ψ from the variables of q_2 to the variables and constants of q_1 such that:
  - for each i, $\psi(y_i) = x_i$
  - for each atom $R(u_i)$ in the body of q_2, $R(\psi(u_i))$ is in the body of q_1
- *Homomorphism theorem*. Let q_1 and q_2 be two conjunctive queries. Then $q_1 \subseteq q_2$ *iff* there exists a homomorphism from q_2 to q_1

- Example
  - $q_1(x_1,x'_1) : -A_1(x_1,x_2,x_3), A_2(x'_1,x_2,x_3)$
  - $q_2(y_1,y'_1) : -A_1(y_1,y_2,y_3), A_2(y'_1,y_2,y'_3)$
  - homomorphism ψ: $\psi(y_i) = x_i$ for each i, $\psi(y'_1) = x'_1$ *and* $\psi(y'_3) = x_3$

Query Containment Algorithm

Input: Two conjunctive queries:

$q_1(x)$:- $g_1(x_1),\ldots,g_n(x_n)$

$q_2(y)$:- $h_1(y_1),\ldots,h_m(y_m)$

Output: Yes if $q_1 \subseteq q_2$; no otherwise

freeze q_1**:** construct a canonical instance $D_{can} = \{g_i(v(x_i)) \mid 1 \leq i \leq n\}$

for some valuation v mapping each variable in q_1 to a distinct constant

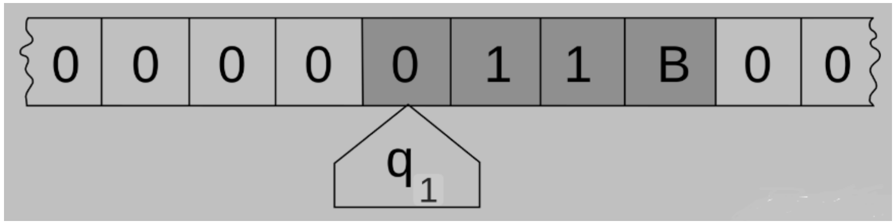
if $v(x) \in q_2(D_{can})$ **return** yes **else return** no

- Combined complexity (database + query) of conjunctive query containment (and conjunctive query answering) is NP-complete
  - PTIME for some classes
- *Data complexity* (query is fixed) of conjunctive queries is AC0

Query Containment Example (I)

- Queries
  - $q_1(x_1, x'_1) :- A_1(x_1, x_2, x_3), A_2(x'_1, x_2, x_3)$
  - $q_2(y_1, y'_1) :- A_1(y_1, y_2, y_3), A_2(y'_1, y_2, y'_3)$
- $q_1 \subseteq q_2$???
- Intuitively, q_2 joins A_1 and A_2 on the second attribute, whereas q_1 also joins on the third one
- $D_{can} = \{ A_1(a, b, c), \; A_2(a', b, c) \}$
- $v(x_1) = a; v(x'_1) = a'; v(x_1, x'_1) = <a, a'>$
- $q_2(D_{can}) = <a, a'> = v(x_1, x'_1)$

Машина Тьюринга



- Машина Тьюринга
  - алфавит $a_1 \dots a_N$
  - состояния $q_1 \dots q_M$
  - движения каретки R, L, N
  - программа – конечное множество команд $q_i\, a_j \rightarrow q_{i1}\, a_{j1}$ (R | L | N)
- Детерминированная машина Тьюринга
  - $\forall$ i, j $\exists$ не больше одной команды, начинающейся с $q_i\, a_j$
  - Для каждого входа вычисления образуют цепь
- Недетерминированная машина Тьюринга
  - могут существовать несколько команд с одинаковой левой частью
  - Для каждого входа вычисления в общем случае образуют дерево
  - Если машина заканчивает работу хотя бы на одной ветви – успешное завершение работы

Basics of Complexity

- Задача p класса сложности P (PTIME)
  - время работы полиномиально зависит от размера входных данных
  - $\exists\, P^n(x)\ \forall\, d\ (\#p(d) \le P^n(d))$
    - P^n – полином степени n
    - d – входные данные задачи p
    - $\#\, p(d)$ – количество шагов исполнения задачи p на данных d на ДМТ
- P-полная задача p_c
  - Любая задача из класса P сводится к p_c
  - p_c – самая сложная задача из класса P (может быть несколько)
  - Пример: линейное программирование (нахождение минимума линейной функции с линейными неравенствами)
- NP-полная задача np_c
  - выполнима за полиномиальное время на НМТ
- Класс AC0 $\subseteq$ PTIME
  - Задачи, реализуемые схемами из функциональных элементов конечной глубины и полиномиальной сложности
  - Задачи выполнимы за константное время на полиномиальном числе процессоров

Global-as-View Mediation

GAV Mapping

- $R(x_1,\ldots,x_n) \supseteq A_1(u_1)\ ,\ \ldots\ ,\ A_k(u_k)$

  head body

  - u_i is a tuple of variables and constants
- Semantics of the mapping
  - FOL($R(x_1,\ldots,x_n) \supseteq A_1(u_1)\ ,\ \ldots\ ,\ A_k(u_k)$) =

    $\forall x_1,\ldots,x_n(\exists y_1,\ldots,y_m(A_1(\vec{u}_1),\ldots,A_k(\vec{u}_k) \Rightarrow R(\vec{u})))$
  - $y_1,\ldots,y_m$ are of variables occurring in the body of the rule and not its head, called *existential*
  - $x_1,\ldots,x_n$ are called *distinguished*

Data Resources (Local Schemas)

- *S1.Catalogue(nomUniv, programme)*
  - a catalog of teaching programs offered in different French universities with master programs
- *S2.Erasmus(student, course, univ)*
  - names of European students enrolled in courses at some university within the Erasmus exchange program
- *S3.CampusFr(student, program, university)*
  - names of foreign students enrolled in programs of some French university
- *S4.Mundus(program, course)*
  - course contents of international master programs

Global Schema

- *MasterStudent(studentName)*
- *University(uniName)*
- *MasterProgram(title)*
- *MasterCourse(code)*
- *EnrolledIn(studentName,title)*
- *RegisteredTo(studentName, uniName)*

GAV Mappings

- $MasterStudent(N) \supseteq S2.Erasmus(N,C,U),\ S4.Mundus(P,C)$
- $MasterStudent(N) \supseteq S3.CampusFr(N,P,U),\ S4.Mundus(P,C)$
- $University(U) \supseteq S1.Catalogue(U,P)$
- $University(U) \supseteq S2.Erasmus(N,C,U)$
- $University(U) \supseteq S3.CampusFr(N,P,U)$
- $MasterProgram(T) \supseteq S4.Mundus(T,C)$
- $MasterCourse(C) \supseteq S4.Mundus(T,C)$
- $EnrolledIn(N,T) \supseteq S2.Erasmus(N,C,U),\ S4.Mundus(T,C)$
- $EnrolledIn(N,T) \supseteq S3.CampusFr(N,T,U),\ S4.Mundus(T,C)$
- $RegisteredTo(N,U) \supseteq S3.CampusFr(N,T,U)$

A Query to Global Schema

- $q(x)$:- $RegisteredTo(s, x)$, $MasterStudent(s)$
  - universities with registered master students
- Rewriting of this query into source queries is obtained by *unfolding* (развертывание) - replacing each atom which can be matched with the head of some view, by the body of the corresponding view
  - variable renaming during matching !
- Let $q(x) :- G_1(z_1), \ldots, G_n(z_n)$ be a query and for each i, $G_i(x_i) \supseteq q_i(x_i, y_i)$ be a GAV mapping. An *unfolding* of q is the query obtained from q by replacing, for each i, each conjunct $G_i(z_i)$ by $q_i(\psi_i(x_i, y_i))$ where ψ_i is a function that maps x_i to z_i, and the existential variables y_i to new fresh variables
  - query $q(x)$:- $F(x,y)$, $G(y)$
  - mappings $F(x,y) \supseteq S(x,z), S(y,z) \qquad G(x) \supseteq S(x,y)$
  - unfolding $q(x) :- S(x,v_1), S(y,v_1), S(y,v_2)$

Unfolding Example

- $q(x)$:- $RegisteredTo(s, x), MasterStudent(s)$

- Mapping whose head can be matched with *RegisteredTo(s,x)*
  - $RegisteredTo(N,U) \supseteq S3.CampusFr(N,T,U)$
- Mappings that match *MasterStudent(s)*
  - $MasterStudent(N) \supseteq S2.Erasmus(N,C,U), S4.Mundus(P,C)$
  - $MasterStudent(N) \supseteq S3.CampusFr(N,P,U), S4.Mundus(P,C)$
- Unfoldings
  - $q1(x)$:- $S3.CampusFr(s,v1,x), S2.Erasmus(s,v2,v3), S4.Mundus(v4,v2)$
  - $q2(x)$:- $S3.CampusFr(s,v5,x), S3.CampusFr(s,v6,v7), S4.Mundus(v6,v8)$

Rewritten Query and its Execution

- *q2(x) :- S3.CampusFr(s,v5,x), S3.CampusFr(s,v6,v7), S4.Mundus(v6,v8)* can be simplified
  - Replacing *S3.CampusFr(s,v5,x), S3.CampusFr(s,v6,v7)* by *S3.CampusFr(s,v6,x)* leads to an equivalent query
  - simplification relies on checking conjunctive query containment
  - simplification is done until the query is "minimal"
  - the resulting query may be much less expensive to evaluate that the initial one
- Final GAV rewritings
  - *r1(x) :- S3.CampusFr(s,v1,x), S2.Erasmus(s,v2,v3), S4.Mundus(v4,v2)*
  - *r2(x) :- S3.CampusFr(s,v6,x), S4.Mundus(v6,v8)*
  - Result - $r1(x) \cup r2(x)$

- Result can be optimized using standard query optimization
- Physical query plan that depends on the statistics that are available and the capabilities of the sources
  - *r2*: querying *S3* and then for each value *a* of *v6* (a particular university program), asking the query *q(X) :- S4.Mundus(a, X)* to *S4*

Correctness of GAV Rewriting

- **Proposition**. *Let S be a set of source relations and G a set of global relations defined by a set of GAV mappings over S. Consider the query* $q(z)$:- $G_{i1}(z_{i1}),\ldots,G_{in}(z_{in})$ *over G and the set* $\{r_\ell\}$ *of unfoldings of q given . Then for each database instance I over* $S_1, \ldots, S_n$, *the answer of q is given by* $\cup r_\ell(I)$.

Local-as-View mediation

LAV Mapping

- $S(x_1,\ldots,x_n) \subseteq A_1(u_1),\ldots,A_k(u_k)$

  head body

- Semantics of the mapping

  $$\forall x_1,\ldots,x_n [S(x_1,\ldots,x_n) \Rightarrow (\exists y_1,\ldots,y_m\, A_1(\vec{u_1}),\ldots,A_k(\vec{u_k}))]$$

Query Rewriting Using Views

- **Query Containment**: $q' \subseteq q \Leftrightarrow \forall D\ q'(D) \subseteq q(D)$
  - D is an instance of the database (a set of instances of all relations contained in the database schema)
- **Query Equivalence**: $q' = q \Leftrightarrow q' \subseteq q\ \&\ q \subseteq q'$

Given query q and view definitions $V = \{v_1, \ldots, v_n\}$

- q' is an **Equivalent Rewriting** of q using V if
  - q' refers only to views in V, and
  - $q' = q$ [$Exp_V(q') = q$]
- q' is an **Maximally-Contained Rewriting** of q using V if
  - q' refers only to views in V
  - $q' \subseteq q$ [$Exp_V(q') \subseteq q$]
  - There is no rewriting q_1, such that $q' \subseteq q_1 \subseteq q$ and $q_1 \neq q'$

- $Exp_V(q)$ – expansion of q w.r.t. V, obtained from q replacing views with their bodies

Rewriting Examples (I)

- Schema
  - *Student(sid, name, dept)*
  - *Course(cid, title, quarter)*
  - *Take(sid, cid, grade)*
- *q(T, G) :- Student(S, N, ee), Take(S, C, G), Course(C, T, Q).*
  - *ee* – constant (electrical engineering)

- Equivalent rewriting
  - $V_1(S, N, D, C, G)$:- *Student(S, N, D), Take(S, C, G).*
  - $V_2(S, C, T)$:- *Take(S, C, G), Course(C, T, Q).*
  - *rewriting(T, G)* :- $V_1(S, N, ee, C, G)$, $V_2(S, C, T)$.
  - $Exp_{V1,V2}$*(rewriting(T, G))* =

    Student(S, N, ee), Take(S, C, G), Take(S, C, G'), Course(C, T, Q').

Rewriting Examples (II)

- Contained rewriting
  - $V_1(S, N, D, C, G)$:- $Student(S, N, D), Take(S, C, G)$.
  - $V_2'(S, C, T)$:- $Take(S, C, G), Course(C, T, fall2006)$.
  - $rewriting(T, G)$:- $V1(S, N, ee, C, G), V2'(S, C, T)$.
  - $Exp_{V1,V2'}(rewriting(T, G)) =$

    $Student(S, N, ee), Take(S, C, G), Take(S, C, G'), Course(C, T, fall2006)$.

- No rewriting
  - $V_1'(S, N, D, C)$:- $Student(S, N, D), Take(S, C, G)$.
    - no grade information
  - $V_2(S, C, T)$:- $Take(S, C, G), Course(C, T, Q)$.

Global Schema

Student(studentName)

EuropeanStudent(studentName)

NonEuropeanStudent(studentName)

EnrolledInProgram(studentName, title)

EnrolledInCourse(studentName, code),

RegisteredTo(studentName, uniName)

University(uniName)

FrenchUniversity(uniName)

EuropeanUniversity(uniName)

NonEuropeanUniversity(uniName)

Program(title)

MasterProgram(title)

Course(code)

PartOf(code, title)

OfferedBy(title, uniName)

LAV Mappings [descriptions of the contents of data sources]

- *m1: S1.Catalogue(U,P)*$\subseteq$ *FrenchUniversity(U), Program(P), OfferedBy(P,U), OfferedBy(P',U), MasterProgram(P')*

- *m2: S2.Erasmus(S,C,U)*$\subseteq$ *Student(S), EnrolledInCourse(S,C), PartOf(C,P), OfferedBy(P,U), EuropeanUniversity(U), EuropeanUniversity(U'), RegisteredTo(S,U'), U$\neq$ U'*

  Students form the Erasmus source:
  European students enrolled in courses
  of a given (European) university
  that is different from
  their home (European) University
  in which they remain registered

- *m3: S3.CampusFr(S,P,U)*$\subseteq$ *NonEuropeanStudent(S), Program(P), EnrolledInProgram(S,P), OfferedBy(P,U), FrenchUniversity(U), RegisteredTo(S,U)*

- *m4: S4.Mundus(P,C)*$\subseteq$ *MasterProgram(P), OfferedBy(P,U), OfferedBy(P,U'), EuropeanUniversity(U), NonEuropeanUniversity(U'), PartOf(C,P)*

Global Query

MasterStudent(E) :-

Student(E), EnrolledInProgram(E, M), MasterProgram(M).

Idea of Rewriting (Bucket, Minicon)

- determine the local relations that are relevant to the query
- consider their combinations as candidate rewritings
- verify whether they are indeed correct

The Bucket Algorithm

1. construct for each atom *g* of the global query body its *bucket*, which groups the view atoms from which *g* can be inferred
2. build a set of candidate rewritings that are obtained by combining the view atoms of each bucket
3. check whether each candidate rewriting is valid

Bucket Creation

- *g* – an atom of the global query
- atoms in *bucket*(*g*) are the heads of mappings having in their body an atom from which *g* can be inferred
  - data comes from source relations, and a (global) query atom is satisfied by (local) data only if it can be matched to a (global) atom in the body of a mapping whose head can be matched to source facts
  - a match between *g* and an atom in the body of a mapping is an indication that the corresponding data source provides a relevant information for the query
- a view atom $v \in$ *bucket*(*g*) only if an atom in the body of *v* can be matched with *g* by a variable mapping such that the variables mapped to the *distinguished* variables of *g* are also *distinguished* variables in the view defining the mapping

Bucket Creation Example

- Global query
  - *q(x) :- RegisteredTo(s, x), EnrolledInProgram(s, p), MasterProgram(p)*
- Global query atom
  - *g = RegisteredTo(s, x)*
    - *x* is *distinguished*
- Mappings in which a body atom can be matched to *RegisteredTo(s, x): m2* and *m3*
  - *m3: S3.CampusFr(S, P, U)* $\subseteq$ *NonEuropeanStudent(S), Program(P), EnrolledInProgram(S, P), OfferedBy(P, U), FrenchUniversity(U), RegisteredTo(S, U)*
    - *RegisteredTo(s, x)* matches *RegisteredTo(S, U) with the variable mapping {S/s,U/x}*
    - *U is distinguished in the view*
  - *m2: S2.Erasmus(S,C,U)*$\subseteq$ *Student(S), EnrolledInCourse(S,C), PartOf(C,P), OfferedBy(P,U), EuropeanUniversity(U), EuropeanUniversity(U'), RegisteredTo(S,U'), U*$\neq$ *U'*
    - *RegisteredTo(s,x)* matches *RegisteredTo(S,U')* by the variable mapping *{S/s,U'/x}*
    - *U' is existentially quantified in the view*
    - *S2.Erasmus(S,C,U)* $\notin$ *bucket(g)*
- *bucket(RegisteredTo(s, x)) = {S3.CampusFr(s, v1, x)}*

Bucket Algorithm

Input: An atom $g = G(u_1,\ldots,u_m)$ of the query q and a set of LAV mappings

Output: The set of view atoms from which g can be inferred

(1) $Bucket(g) := \emptyset$

(2) **for each** LAV mapping $S(x) \subseteq p(x, y)$

(3) **if** there exists in $p(x,y)$ an atom $G(z_1, \ldots , z_m)$ such that

(4) z_i is distinguished for each i such that u_i is distinguished in q;

(5) **let** ψ the variable mapping $\{z_1/u_1, \ldots , z_m/u_m\}$

(6) extended by mapping the head variables in x not

(7) appearing in $\{z_1,\ldots,z_m\}$ to new fresh variables;

(8) **add** $S(\psi(x))$ to $Bucket(g)$

(9) **return** $Bucket(g)$

Logical Characterization of the View Atoms in the Buckets

Proposition. *Let $G(u_1,\ldots,u_m)$ be an atom of the global query. Let u be the (possibly empty) subset of existential variables in $\{u_1,\ldots,u_m\}$.*
Let m: $S(x) \subseteq q(x, y)$ be a LAV mapping. Then

$$S(v), FOL(m) \models \exists\, u\; G(u_1,\ldots,u_m)$$

iff there exists a view atom in Bucket(G) that is equal to S(v) (up to a renaming of the fresh variables).

Example.

S3.CampusFr(s, v1, x) &

$\forall$S, P, U (S3.CampusFr(S, P, U) $\Rightarrow$ NonEuropeanStudent(S) & Program(P) & EnrolledInProgram(S, P) & OfferedBy(P, U) & FrenchUniversity(U) & RegisteredTo(S, U))

$\models$ *$\exists$ s RegisteredTo(s, x)*

Candidate Rewritings

- Buckets

RegisteredTo(s,x)	EnrolledInProgram(s,p)	MasterProgram(p)
S3.CampusFr(s,v1,x)	S3.CampusFr(s,p,v2)	S1.Catalogue(v3,v4) S4.Mundus(p,v5)

- Candidate rewritings of the initial global query are then obtained by combining the view atoms of each bucket
  - *r1(x) :- S3.CampusFr(s,v1,x), S3.CampusFr(s,p,v2), S1.Catalogue(v3,v4)*
  - *r2(x) :- S3.CampusFr(s,v1,x), S3.CampusFr(s,p,v2), S4.Mundus(p,v5)*

Candidate Rewritings Validity

- Expanding *r1*, *r2* (replacing view heads by view bodies)
  - *Exp_r1(x) :- NonEuropeanStudent(s), Program(v1), EnrolledInProgram(s,v1), OfferedBy(v1,x), FrenchUniversity(x), RegisteredTo(s,x), Program(p), EnrolledInProgram(s,p), OfferedBy(p,v2), FrenchUniversity(v2), RegisteredTo(s,v2), FrenchUniversity(v3), Program(v4),OfferedBy(v4,v3), OfferedBy(v5,v3), MasterProgram(v5)*
  - new existential variables may be introduced by the expansion of some view atoms
    - *S1.Catalogue(v3,v4)* contains the existential variable *P'* in the LAV
  - such variables are renamed with new fresh variables to avoid unnecessary constraints between the variables
    - *v5*
- *ri* is *valid* if $Exp\_ri(x) \subseteq q(x)$ [query containment]
  - *r1* is not valid
  - *r2* is valid