Logic-based Techniques for Information Integration

Marie-Christine ROUSSET
University of Grenoble

Introduction

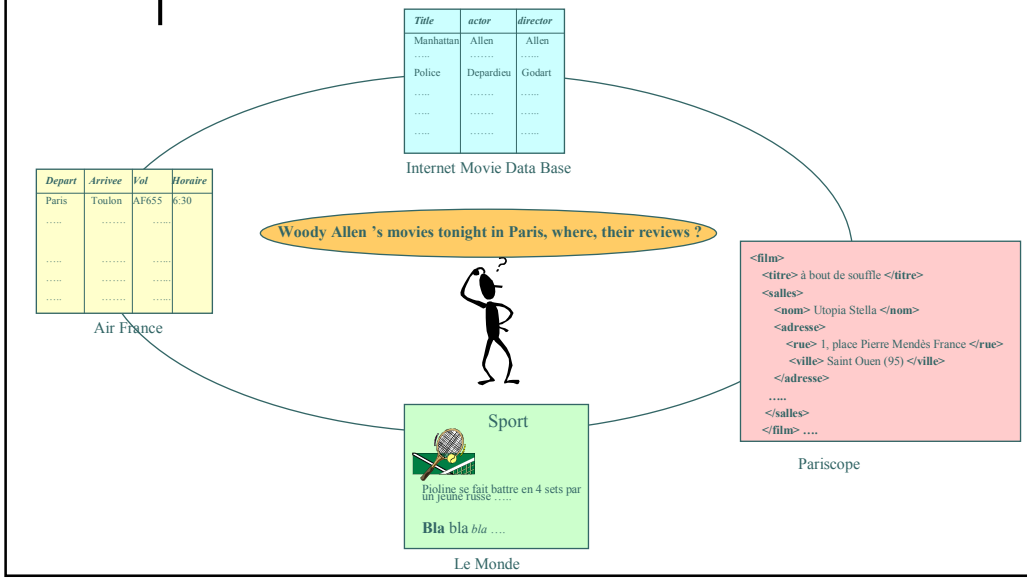
Information integration

- Core issue for the construction of modern information servers
  - **traditional DBMSs** : efficient management and querying of structured, centralized and reliable data
  - **data available through the Web** : distributed over autonomous and heterogeneous sources
- Requires innovative solutions
  - **mediation systems**

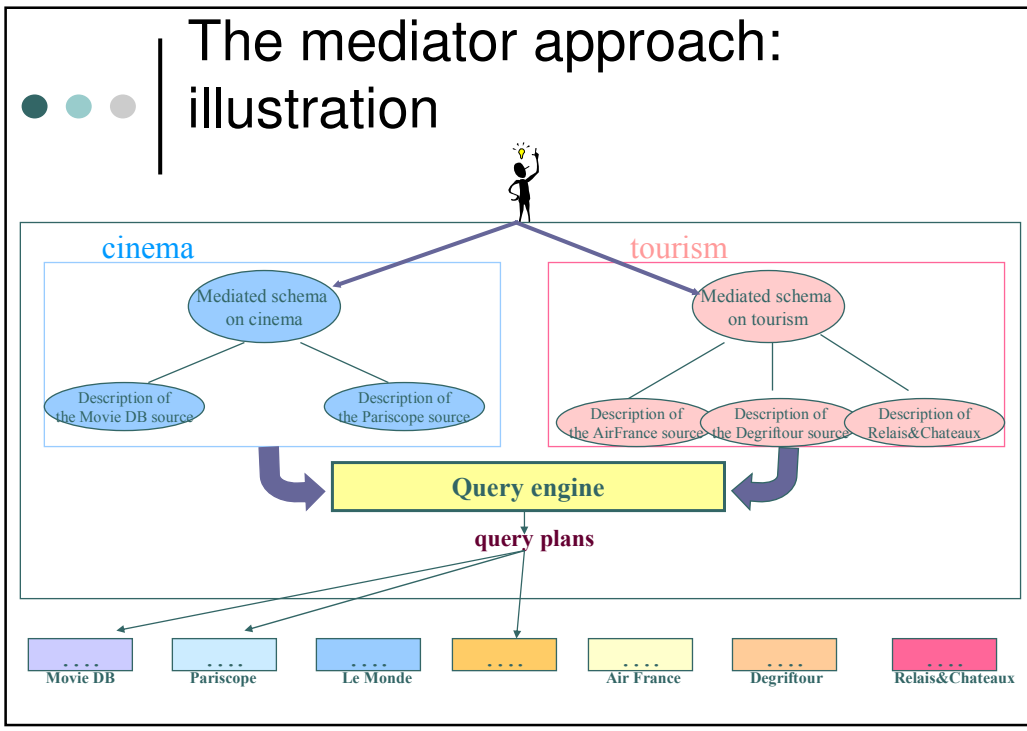
Key problems

- Define a mediated schema
  - A structured vocabulary of the domain of interest serving as query interface for users
- Describe the content of pre-existing data sources in terms of a mediated schema
- Reformulate and decompose user's queries into queries executable against the relevant data sources
- Combine the answers to the local queries in order to get (certain) answers to the original user's query

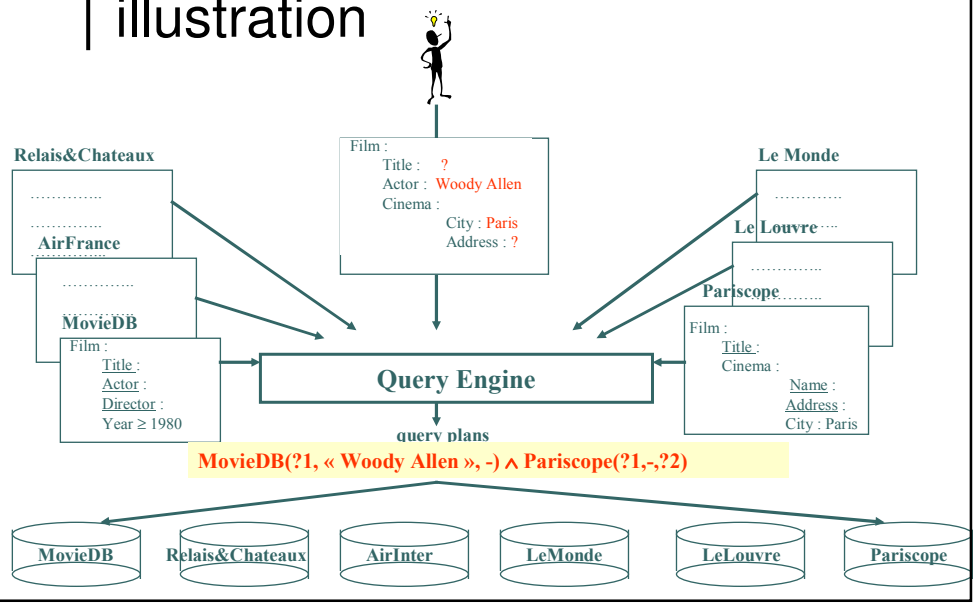
Illustration



The mediator approach: illustration



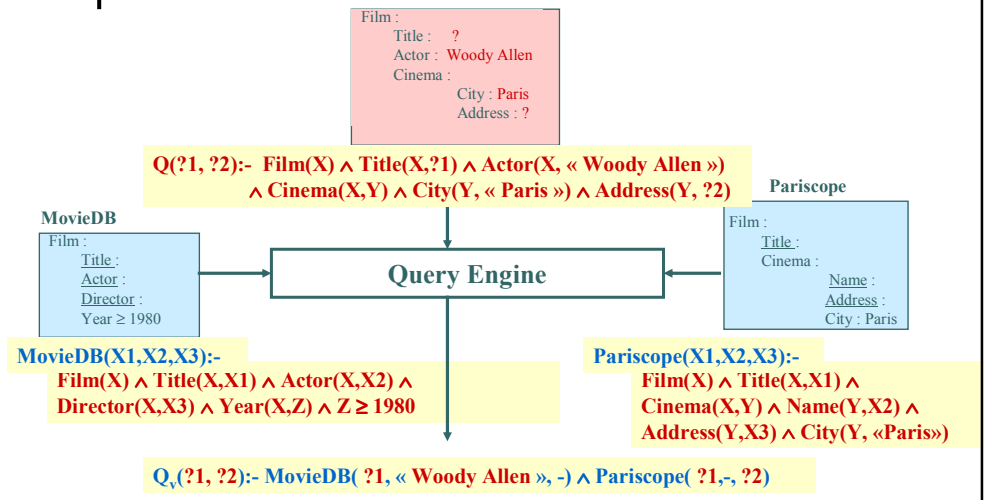
Query reformulation: illustration



Principles of a logical approach

- Logical description of schema, queries and views
  - the mediated schema :
    - a domain specific knowledge base (domain ontology)
  - the views :
    - an abstract description of the content of the data sources to integrate
- Reasoning for query reformulation
  - Query rewriting in terms of views
  - decidability and complexity depending on the logical formalism

Formal background: a logical semantics



Overview of the course

- Queries and views: logical semantics
- Query answering and rewriting: two reasoning problems
- Query rewriting using views in the simple relational model
  - Main algorithms
- Answering queries over ontologies by rewriting
  - First results on a complex problem

Queries

- Open formulae in first-order logic (FOL)

  $q(\underline{x}): \exists \underline{y}\ \Phi(\underline{x},\underline{y})$

  - $\underline{x}$ is a tuple of free (a.k.a distinguished) variables
  - $\Phi(\underline{x},\underline{y})$ is a FOL formula with as free variables those in $\underline{x}$ and $\underline{y}$
- Example

$q(X) : \exists A,C\ Flight(X) \wedge ArrivalAirport(X,A) \wedge Located(A,C) \wedge Capital(C)$

Datalog notation:

$q(X) \leftarrow Flight(X) \wedge ArrivalAirport(X,A) \wedge Located(A,C) \wedge Capital(C)$

Answers

- Answer set of a query q over a knowledge base K

  $Ans(q,K) = \{\underline{a} \mid K \models q(\underline{a})\}$

  - K is a set of closed FOL formulae
    - a database (a set of ground atoms of the form R(a1, …,an) where R is a relation of the schema) + possibly constraints on those relations
    - Abox $\cup$ Tbox (in Description Logic knowledge bases)
  - $\underline{a}$ is a tuple of constants appearing in K
  - $q(\underline{a})$: the formula $\exists \underline{y}\ \Phi(\underline{x},\underline{y})$ where the variables of $\underline{x}$ are replaced by the constants of $\underline{a}$

Views

- Named queries
  - for representing by a formula a set of data
    - cached answers of a given query to be re-used for optimization of answering new queries
    - the content of a distant data source
- Example

v1(X,Y1,Y2) : Flight(X) ∧ DepartureAirport(X,Y1) ∧ ArrivalAirport(X,Y2)
v2(X,Y) : Place(X) ∧ Located(X,Y) ∧ Capital(Y)

- can be materialized or virtual
  - Their extension can be stored (cached) or computed (by querying an external source)

Different semantics

of the mapping v(x) : def(x,y) between the view v and the query defining it

- exact semantics

  Ext(v) = Ans(def,K)

  new axiom : $\forall \underline{x}$ [v(x) $\Leftrightarrow \exists$y def(x,y)] added to K
- sound semantics

  Ext(v) $\subseteq$ Ans(def,K)

  new axiom : $\forall \underline{x}$ [v(x) $\Rightarrow \exists$y def(x,y)] added to K
- complete semantics

  Ans(def,K) $\subseteq$ Ext(v)

  new axiom : $\forall \underline{x}$ [$\exists$y def(x,y) $\Rightarrow$ v(x)] added to K

Answering queries using views

- Let q a query over a knowledge base K, V a set of views, Kv the set of axioms defining the semantics of the views w.r.t K, ext(V) the union of the view extensions,
- the set of **certain** answers of q over a K w.r.t V is

$$\text{CertainAns}(q,K,V) = \{\underline{a} \mid K \cup Kv \cup ext(V) \models q(\underline{a})\}$$

where $\underline{a}$ is a tuple of constants appearing in $K \cup ext(V)$

a reasoning problem on data

- More complex than answering queries in DBMS
  - The views extensions provide **incomplete** data on the extensions of the query predicates.

Views :
v1(X) : Reservation(X,X) ∧ Reservation(X,Y)
v2(X) : Reservation(X,Y)
v3(X) : Rservation(Y,X).

Views extensions :

v1	v2	v3
a	a	a
	b	c

Query :
q(X1,X2) ← Reservation(X1,X2).

The only **certain** answer to q using views: **(a,a)**

Reservation
(a,a)
(a,?)
(a,?)
(b,?)
(?,a)
(?,c)

Rewriting queries using views

- Given
  - a query $q(\underline{x})$ defined over K using a language L1
  - a set of views V defined on D in L2

    $V= \{v_1(\underline{x}_1) : Q_1(\underline{x}_1), \ldots, v_k(\underline{x}_k) : Q_k(\underline{x}_k)\}$
- q_v is a (maximal) rewriting of q using V if it is:
  - defined on $\{v_1, \ldots, v_k\}$ in L3
  - (maximally) contained in q

    for every $\underline{x}$: $q_v(\underline{x})$, $Kv \cup K \models q(\underline{x})$

    (if q'_v is a rewriting of q s.t for every $\underline{x}$ $q_v(\underline{x}) \models q'_v(\underline{x})$ then for every $\underline{x}$ $q_v(\underline{x}) \equiv q'_v(\underline{x})$)

Example

- Query :

$q(X) \leftarrow Flight(X) \wedge ArrivalAirport(X,A) \wedge Located(A,C) \wedge Capital(C)$

- Views (sound semantics) :

$v1(X,Y1,Y2) : Flight(X) \wedge DepartureAirport(X,Y1) \wedge ArrivalAirport(X,Y2)$

$v2(X,Y) : Place(X) \wedge Located(X,Y) \wedge Capital(Y)$

- Rewriting :

$q_v(X) \leftarrow v1(X,Y1,Y2) \wedge v2(Y2,Y)$

$Flight(X) \wedge DepartureAirport(X,Y1) \wedge ArrivalAirport(X,Y2)$

$\wedge Place(Y2) \wedge Located(Y2,Y) \wedge Capital(Y)$

Example with existential variables in the views definitions

- Query :

$q(X) \leftarrow$ Manager(X,Y) $\wedge$ Head(Y,Z)

- Views (exact semantics) :

v1(X) : Manager(X,Y)
v2(Y,Z) : Head(Y,Z)

$q_v(X) \leftarrow$ v1(X) $\wedge$ v2(Y,Z) rewriting of q ?

1) [$\exists$ Y,Z v1(X) $\wedge$ v2(Y,Z)], Kv $\equiv$ [$\exists$Y,Z,Y' Manager(X,Y') $\wedge$ Head(Y,Z)]

2) [$\exists$Y, Z,Y' Manager(X,Y') $\wedge$ Head(Y,Z) $\not\models \exists$ Y, Z Manager(X,Y) $\wedge$ Head(Y,Z)

q_v is not a rewriting of q

Answering queries by rewriting

For any q_v rewriting of q using views V :

$Ans(q_v,Ext(V)) = \{\underline{a} \mid Ext(V) \models q_v(\underline{a})\}$

$q_v(\underline{a})$, Kv $\cup$ K $\models q(\underline{a})$

$CertainAns(q,K,V) = \{\underline{a} \mid K \cup Kv \cup ext(V) \models q(\underline{a})\}$

Therefore: **$Ans(q_v,Ext(V)) \subseteq CertainAns(q,K,V)$**

- Rewriting the query using views:
  - A reasoning problem
    - on the query, the view definitions and the axioms
    - independent of the data
- Evaluating each rewriting against the views extensions
  - A DBMS problem, known to be polynomial in data complexity (the size of the data)

Key question

- Can we compute all the certain answers by computing and evaluating the rewritings ?
  - depends on the query, view and rewriting languages and semantics
  - necessary condition :
    - The instance checking problem is polynomial in data complexity
      - a bunch of negative results coming from the complexity of reasoning in Description Logics
  - Sufficient condition :
    - There exists a finite number of maximal conjonctive rewritings
      - Positive results in the simple setting of relational databases

Answering queries by rewriting using views in relational setting

Query rewriting using views in relational databases

- Extensively studied in DB theory
  - central for query optimization
  - closely related to the problem of query containment
- Results of decidability and complexity
  - depending on the (relational) languages used for the queries, rewritings and views
  - NP-complete when queries, rewritings and views are conjunctive queries

The «Bucket » algorithm

- Input
  - a conjunctive query (possibly with comparison predicates) over a relational schema
  - a set of sound views defined by conjunctive queries (possibly with comparison predicates)
- output : a set of conjunctive rewritings
  - maximally contained if the query is without comparison predicates (complete algorithm)
- Implemented in Information Manifold

A.Levy, A. Rajaraman, J.Ordille. Querying heterogeneous information sources using source descriptions. Proceedings of the Int.Conference on Very Large Data bases (VLDB 96)

Principles : two steps

- Create a « bucket» for each atom g in the query
  - store there all the view-atoms having a conjunct unifiable with g in its definition (without violating the comparison constraints)
- Build the set of candidate rewritings
  - pick one view-atom per bucket and take their conjunction
  - For each of them check whether their expansion is contained in the query
    - If yes : return it in the output
    - If no : add comparison atoms if it makes containment possible

Creation of bucket(g)

- let V be a view-atom such that one atom g' in its definition is unifiable with g
  - let ψ be the substitution of the variables of g' onto the variables of g
- let C(V) (resp. C(Q)) be the conjunction of comparison atoms in the definition of V (resp. Q)
  - If ψ(C(V)) and C(Q) are compatible

    add the atom ψ'(V) to bucket(g)

  where ψ' extends ψ to the variables in V and not in g' by substituting them with new variables

Example

views

V1(Student,Number,Year) : registered(Student,Course,Year) ∧ course(Course,Number)
∧ Number ≥ 500 ∧ Year ≥ 1992

V2(Student,Dept,Course) : registered(Student,Course,Year) ∧ enrolled(Student,Dept)

V3(Student,Course) : registered(Student,Course,Year) ∧ Year ≤ 1990

V4(Student,Course,Number) : registered(Student,Course,Year) ∧ course(Course,Number)
∧ enrolled(Student,Dept) ∧ Number ≤ 100

query : q(S,D) ← enrolled(S,D) ∧ registered(S,C,Y) ∧ course(C,N) ∧ N ≥ 300 ∧ Y ≥ 1995

Bucket(enrolled(S,D)) = {V2(S,D,C') , V4(S,C'',N')}

V3(S,C) ∉ Bucket(registered(S,C,Y))
because Y ≤ 1990 and Y ≥ 1995 are not compatible
V4(S,C,N''') ∈ Bucket(registered(S,C,Y))
because N''' ≤ 100 and N ≥ 300 are compatible

Example(ctd)

Bucket(enrolled(S,D)) = {V2(S,D,C') , V4(S,C'',N')}

Bucket(registered(S,C,Y)) = {V1(S,N'',Y) , V2(S,D',C) , V4(S,C,N''')}

Bucket(course(C,N)) = {V1(S',N,Y')}

Candidate rewriting number 1 :

Q1(S,D) ← V2(S,D,C') ∧ V1(S,N'',Y) ∧ V1(S',N,Y')

Example (ctd)

Verification of that candidate as a rewriting

$Q1(S,D) \leftarrow V2(S,D,C') \wedge V1(S,N'',Y) \wedge V1(S',N,Y')$

Containment checking between the query resulting from the candidate expansion :

$Exp(Q1)(S,D) \leftarrow$ registered(S,C',Y1) $\wedge$ enrolled(S,D) $\wedge$ registered(S,C1,Y) $\wedge$ course(C1,N'')

$\wedge N' \geq 500 \wedge Y \geq 1992 \wedge$ registered(S',C2,Y') $\wedge$ course(C2,N) $\wedge N \geq 500 \wedge Y' \geq 1992$

and the intial query :

$q(S,D) \leftarrow$ enrolled(S,D) $\wedge$ registered(S,C,Y) $\wedge$ course(C,N) $\wedge N \geq 300 \wedge Y \geq 1995$

- The variables of exp(Q1) are « frozen » :

$S \rightarrow a1$, $C' \rightarrow a2$, $Y1 \rightarrow a3$, $D \rightarrow a4$, $C1 \rightarrow a5$, $Y \rightarrow a6$, $N'' \rightarrow a7$, $S' \rightarrow a8$, $C2 \rightarrow a9$, $Y' \rightarrow a10$, $N \rightarrow a11$

- q(S,D) is evaluated against the resulting database :

FAILURE

SUCCESS if we add the comparison atoms : $N=N''$ **and** $Y \geq 1995$

Minicon : optimization of the bucket algorithm

- Query containment checking is avoided by a more elaborate verification of the atoms to add to the buckets.
  - When the definition of a view V contains an atom g' such that : $\sigma(g') = g$
    - If an existential variable Y of g also appears in other atoms g1, g2, …, gk of the query
    - if $Y' = \sigma(Y)$ is also existential in the view definition
      - $\sigma(V)$ is added to Bucket(g) only if g1, g2, …, gk are also covered by the defintion of σ (V)

Illustration by example

V4(X) :- cite(X,Y) ∧ cite(Y,X)

V5(X,Y) :- sameTopic(X,Y)

V6(X,Y) :- cite(X,Z) ∧ cite(Z,Y) ∧ sameTopic(X,Z)

Query : Q(U) :- cite(U,V) ∧ cite(V,U) ∧ sameTopic(U,V)

Bucket (cite(U,V)) ?

- **V4(U) is not added** because sameTopic(U,V) is not covered by the body of V4(U)

- **V6 ?**

σ(X)=U and σ(Z)=V

covering of cite(V,U) by the body of V6(U,Y) => σ(Y)=U

covering of sameTopic(U,V) by the body of σ (V6(X,Y)) ? yes

Bucket(cite(U,V)) = {V6(U,U)}

cover(V6(U,U)) = {cite(U,V), cite(V,U), sameTopic(U,V)}

Advantages of Minicon

- The rewritings are directly obtained by conjuncting the view-atoms of each bucket which have pairwise pairwise disjoint covering.
- Results
  - theoretical :
    - Same worst-case complexity as the Bucket algorithm (exponential in the size of the query)
  - experimental :
    - Scales up much better than the Bucket algorithm in presence of many views.

Summary

- When queries and views are (union of) conjunctive queries over a simple relational schema, the number of maximal conjunctive rewritings is finite and several algorithms exist for computing them.
- No guarantee that this property holds when some knowledge is added to the global schema or to the views
  - to express some binding access patterns to the sources
  - when queries are expressed w.r.t an ontology (schema with axioms or constraints)

Binding access patterns

- Some sources have some constraints to access to their content
  - to get movie titles from Internet Movie Database, we have to provide the name of an actor or of the director
  - for each view v(Att1,…,Attn) describing a source, for each attribute Atti we have to indicate if it must be instantiated for the source to provide answers
    - a simple way of modeling this knowledge: associate to each view a word of size n of *b* or *f*
      - a *b* in position i indicates that the attribute Atti must be instantiated for the source to return answers

Example

Source 1: retruns papers published in AAAI conferences

$PaperSource^{f}(X)$: AAAIpapers(X)

Source 2: returns papers cited by a given paper that must be provided as input

$CitationSource^{bf}(X,Y)$: Cites(X,Y)

Source3: for a given paper provided as input, answers if it has been awarded or not

$AwardSource^{b}(X)$: AwardPaper(X

Query : find all the awarded papers

Q(X) ← AwardPaper(X)

- **$Q_v(X) \leftarrow AwardSource(X)$** is not an executable rewriting
- **$Q'_v(X) \leftarrow PaperSource(X) \wedge AwardSource(X)$** is an executable rewriting
- **$Q''_v(X) \leftarrow PaperSource(V) \wedge CitationSource(V,X1) \wedge \ldots \wedge CitationSource(Xn,X) \wedge AwardSource(X)$**

is also an executable rewriting

The problem

- Infinite number of maximal conjunctive rewritings
- A maximal rewriting in Datalog :

$Papers(X) \leftarrow PaperSource(X)$

$Papers(X) \leftarrow Papers(Y) \wedge CitationSource(Y,X)$

$Q_v(X) \leftarrow Papers(X) \wedge AwardSource(X)$

Answering queries over ontologies by rewriting

Ontologies

- vocabulaires structurés
  - noms de concepts/classes
  - noms de propriétés
- définis à l'aide de langages formels
  - pour la définition et typage des concepts et des propriétés
  - permettant de faire des inférences fondées sur une sémantique logique
- OWL :Ontology Web Language
  - fondé sur les Logiques de Description

Logiques de Description

- Fragments décidables de la logique du premier ordre avec égalité
- Algorithmes de raisonnement avec une étude approfondie de leur complexité
- Des systèmes implémentés (RACER, PELLET)

Les constructeurs de classes

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1 \wedge \ldots \wedge C_n$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1 \vee \ldots \vee C_n$
complementOf	$\neg C$	$\neg$Male	$\neg C$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x_1 \vee \ldots \vee x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$[P]_{n+1}$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\langle P \rangle_n$

Les constructeurs de classes peuvent être imbriqués

- Person$\cap \exists$ haschild ($\forall$haschild.Doctor)

Les contraintes (axiomes)

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

FOL axiomatization

$(C1 \cap C2)(X) \equiv C1(X) \wedge C2(X)$

$(C1 \cup C2)(X) \equiv C1(X) \vee C2(X)$

$(\forall R.C)(X) \equiv \forall Y\ (R(X,Y) \Rightarrow C(Y))$

$(\exists R.C)(X) \equiv \exists Y\ (R(X,Y) \wedge C(Y))$

$(\geq n\ R)(X) \equiv \exists Y_1 \ldots Y_n\ (R(X, Y_1) \wedge \ldots \wedge R(X, Y_n) \wedge \bigwedge_{\{i,j/i \neq j\}} Y_i \neq Y_j$

$(\leq n\ R)(X) \equiv \exists Y_1 \ldots Y_{n+1}\ ((R(X, Y_1) \wedge \ldots \wedge R(X, Y_{n+1}) \Rightarrow \bigvee_{\{i,j/i \neq j\}} Y_i = Y_j$

Problèmes d'inférence étudiés

- Test de subsomption
  - Entre 2 expressions de concepts
  - Entre 2 expressions de concepts modulo un ensemble de contraintes définies dans une Tbox T:

    $T \models C1 \subseteq C2$?
- Test d'appartenance d'une instance à un concept
  - Étant donnés un concept C, une constante a, une Tbox T, une Abox A (un ensemble de faits de la forme A(b) et P(b,c))

    $T \cup A \models C(a)$?
- Nombreux résultats de décidabilité et de complexité en fonction des constructeurs et des axiomes autorisés dans T

Quelques résultats de complexité

Constructeurs	Complexité de la subsomption
ALN ($\cap$, forall, atleast, atmost)	P
ALE ($\cap$, forall, exists)	NP-complet
ALNE ($\cap$, forall, atleast, atmost, exists)	NP-complet
ALN + conjonction de rôles	co-NP-hard
ALC ($\cap$, forall, not)	Pspace-complet
ALCNR	Pspace-complet

La famille DL-Lite

- Restriction sur les constructeurs et les contraintes autorisées pour une complexité polynômiale du calcul des réponses à une requête conjonctive
- Approche suivie
  - Trouver des logiques de description (ou des langages logiques) pour lesquels le calcul de réponses est réductible à l'évaluation de requêtes sur une BD relationnelle simple
  - Condition nécessaire: la reconnaissance qu'un tuple est une réponse est LOGSPACE (et donc P)

DL-Lite : constructeurs et axiomes

- DL-Lite$_{core}$:
  un ensemble d'axiomes d'inclusion $B \subseteq C$ où B et C sont des expressions de concept sur des concepts atomiques A et des rôles atomiques P de la forme
  $B \rightarrow A \mid \exists R \qquad R \rightarrow P \mid P^{-} \qquad C \rightarrow B \mid \neg B$
- DL-Lite$_R$:
  DL-Lite$_{core}$ + axiomes d'inclusion de rôles $R \subseteq E$ où E est de la forme R ou $\neg R$
- DL-Lite$_F$:
  DL-Lite$_{core}$ + axiomes de fonctionnalité : (funct R)

Expressivité de DL-Lite

- Capture les principales contraintes utilisées en bases de données et génie logiciel
  - Relation ISA : $A1 \subseteq A2$
  - Disjonction : $A1 \subseteq \neg A2$
  - typage :
    - $\exists P \subseteq A1$ (le premier attribut de P est typé par A1)
    - $\exists P^- \subseteq A2$ (le second attribut de P est typé par A2)
  - Propriétés obligatoires ou interdites :
    - $A \subseteq \exists P$ $A \subseteq \exists P^-$ $A \subseteq \neg\exists P$ $A \subseteq \neg\exists P^-$
  - Restriction de fonctionnalité pour les relations binaires (les rôles)

Exemple

$Professor \subseteq \exists TeachesTo$
$Student \subseteq \exists HasTutor$
$\exists TeachesTo^- \subseteq Student$
$\exists HasTutor^- \subseteq Professor$
$Professor \subseteq \neg Student$

$HasTutor^- \subseteq TeachesTo$ $DL\text{-}Lite_R$
(func HasTutor) $DL\text{-}Lite_F$

Requêtes

- (Union de) requêtes conjonctives sur des concerpts et rôles atomiques

  $q0(x) \leftarrow TeachesTo(x,y) \wedge HasTutor(y,z)$

- Algorithme de récriture correct et complet

  $q1(x) \leftarrow TeachesTo(x,y) \wedge Student(y)$

  $q2(x) \leftarrow TeachesTo(x,y) \wedge TeachesTo(z',y)$

  $q3(x) \leftarrow TeachesTo(x,y')$

  $q4(x) \leftarrow Professor(x)$

  $q5(x) \leftarrow HasTutor(u,x)$

$$Ans(q0, T \cup A) = \cup_i \; Ans(qi, A)$$

Résumé

- Répondre (de façon complète) à des requêtes par récriture n'est « tractable » que pour très peu de Logiques de Description
- La reconnaissance d'instance (et donc le calcul des réponses à une requête) est coNP-hard en complexité des données pour des extensions mineures of DL-Lite$_{core}$ telles que :
  - DL-Lite$_{FR}$
  - $\forall$R.A ou $\neg$A apparaissent en partie gauche d'un axiome d'inclusion
  - $\forall$R.A ou $\neg$A apparaissent dans la requête
  - Union de concepts en partie gauche d'un axiome d'inclusion

References

- Querying heterogeneous information sources using source descriptions. A.Levy, A. Rajaraman, J.Ordille. Proceedings of the Int.Conference on Very Large Data bases (VLDB 96)
- Information Integration Using Logical Views, J. Ullman, Proceedings ICDT '97
- Logic-based techniques in data integration. A. Halevy, in Logic Based Artificial Intelligence, Ed. Jack Minker, Kluwer Publishers, 2000.
- Query Containment for Data Integration Systems, T. Millstein, A.Levy, M. Friedman, Proceedings PODS 2000
- Complexity of Answering Queries Using Materialized Views, S. Abiteboul, O. Duschka, Proceedings PODS 1998
- Theory of Answering Queries Using Views, A. Halevy, Proceedings of SIGMOD 2000
- The Use of CARIN Language and Algorithms for Information Integration: the PICSEL System, F. Goasdoué, V. Lattès, -C. Rousset. International Journal of Cooperative Systems, Vol 9, Number 4 (2000)
- Minicon: a scalable algorithm for answering queries using views. R. Pottinger, A. Halevy, VLDB Journal, Volume 10 (2-3), 2001.
- Query Rewriting and Answering under Constraints in Data Integration Systems, A. Cali, D. Lembo, and R. Rosati, Proceedings of IJCAI 2003
- D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, DL-Lite: Tractable Description Logics for Ontologies, Proceedings of AAAI 2005.
- D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Data Complexity of Query Answering in Description Logics, Proceedings of KR 2006

Logique des prédicats (rappels)

- Vocabulaire :
  - prédicats (ou relations), fonctions, constantes, variables
- Connecteurs logiques :
  - usuels : $\wedge, \vee, \neg, \Rightarrow, \ldots$
  - quantificateurs : $\forall, \exists$
- Règles de bonne formation de formules :
  - Termes
    - de base (variables, constantes)
    - fonctionnels (application de fonctions à des termes)
  - Formules atomiques : R(t1,t2,…, tn)
  - Quantification de variables :
    - Variables libres versus liées
  - Formules :
    - de base : formules atomiques
    - application des connecteurs logiques à des formules

Exemples

$\forall x\ R(x,a)$ R: prédicat, a: constante

$\exists y\ \forall x\ R(x,y)$ x,y: variables liées

$\forall x\ R(f(x),x)$ f: fonction

$\forall x\ \forall y\ \forall z\ (R(x,y) \wedge R(y,z) \Rightarrow R(x,z))$

$\forall x\ \exists y\ \neg R(x,y)$

$\exists y\ \neg R(y,x)$ x: variable libre

Interprétations et modèles de formules

- Une interprétation I de φ:
  - un domaine d'interprétation Δ^I
  - une fonction d'interprétation associant
    - aux constantes a de φ des éléments a^I de Δ^I
    - aux fonctions f (prédicats R) de φ des fonctions f^I de Δ^I dans Δ^I (relations R^I sur Δ^I).
  - des règles d'interprétations pour interpréter une formule à partir de l'interprétation de ses sous-formules
    - Interprétation d'une formule close (dont toutes les variables sont quantifiées) : valeur de vérité (vrai ou faux)
    - Interprétation d'une formule ouverte (n variables libres) : une relation n-aire sur Δ^I
- Un modèle d'une formule φ : une interprétation I
  - φ^I = vrai (si φ est close)
  - $\varphi^I \neq \emptyset$ (si φ est ouverte)

Règles d'interprétation des quantificateurs

- Soit φ une formule close de la forme $\forall x\ \psi$

  $[\forall x\ \psi(x)\ ]^I$ = vrai ssi pour tout $e \in \Delta^I$, $\psi^I(e)$ est vrai

- Soit φ une formule close de la forme $\exists x\ \psi$

  $[\exists\ x\ \psi(x)\ ]^I$ = vrai ssi il existe $e \in \Delta^I$, $\psi^I(e)$ est vrai

- Soit φ(x1,…,xn) une formule avec n variables libres

  $[\varphi(x1,\ldots,xn)]^I$ = {(e1,…,en) $\in \Delta^I$ x …x Δ^I / φ^I(e1,…,en) est vrai}

Exemples (suite)

Δ^I : N (ensemble des entiers naturels)

$a^I = 0$; f^I = succ ; $R^I = \geq$

$[\forall x\ R(x,a)]^I$ = vrai (car pour tout $e \in N$, $e \geq 0$)

$[\exists y\ \forall x\ R(x,y)]^I$ = vrai (car il existe $e \in N$ (e=0) tel que pour tout $e' \in N$ $e' \geq e$)

$[\forall x\ R(f(x),x)]^I$ = vrai (car pour tout $e \in N$, succ(e) $\geq$ e)

$[\forall x\ \forall y\ \forall z\ (R(x,y) \wedge R(y,z) \Rightarrow R(x,z))]^I$ = vrai car pour tout e1,e2,e3 $\in N$, si $e1 \geq e2$ et $e2 \geq e3$ alors $e1 \geq e3$)

$[\forall x\ \exists y\ \neg R(x,y)]^I$ = faux (car $[\exists y\ \forall x\ R(x,y)]^I$ = vrai)

$[\exists y\ \neg R(y,x)\ )]^I$ = {e $\in$ N/ il existe e' $\in$ N tel que e' < e}

= N^*

Problèmes d'inférence

- Satisfiabilité :
  - existence d'un modèle
- Déterminer si une formule φ est conséquence logique d'un ensemble de formules φ1, …, φn :
  - tout modèle de φ1, …, φn est modèle de φ
  - φ1, …, φn, ¬φ
- Problèmes semi-décidables :
  - on ne connaît pas d'algorithmes qui décident si une formule quelconque est satisfiable ou non satisfiable (est conséquence logique ou non d'un ensemble de formules)
- L'ensemble des interprétation d'une formule quelconque est infini

Fragments décidables

Des sous-ensembles de formules pour lesquels les problème d'inférence sont décidables

- Les règles logiques sans fonction :
  - Fondements logiques des systèmes experts avec variables
  - Datalog /BD déductives
- Les logiques de description :
  - Fondements logiques des langages à base de classes ou de concepts