SAPIENZA
UNIVERSITÀ DI ROMA

IMDB

The Internet Movie Database

DL-Lite$_A$ Ontology

Seminari di Ingegneria del Software

Prof. G. De Giacomo

SAPIENZA
UNIVERSITÀ DI ROMA

SOMMARIO

- Specifiche del progetto
- Ontology-Based Data Access
- Ontologie
- DL-Lite$_A$
- Mapping
- Reasoning in DL-Lite$_A$
- Reasoning in DL-Lite$_A$ con mappings
- Linguaggi di query epistemici
- Implementazione TBox IMDB
- Implementazione Mappings IMDB
- Test e problemi riscontrati

1. Specifiche del progetto

- Scrittura di un'ontologia in DL-Lite$_A$ per rappresentare il dominio di IMDB (The Internet Movie Database – www.imdb.com)
- Mapping dell'ontologia su un database relazionale preesistente
- Implementazione dell'ontologia in Mastro-I
- Implementazione dell'ontologia in Protégé (plugin OBDA)
- Interrogazione dell'ontologia (CQ e EQL)

2.1 Ontology-based Data Access

Lo scopo dell'OBDA è fornire ai client del sistema un punto d'accesso unificato a un insieme di sorgenti dati eterogenee, mediante la conoscenza del dominio descritto con un'ontologia.

Un sistema OBDA è composto da una tripla <G,S,M>:

- G: rappresentazione del Global Schema in logica
- S: sorgente dati relazionale
- M: linguaggio di mapping di tipo GAV (global as view), per mettere in relazione le sorgenti con il global schema

2.2 Ontology-based Data Access

Requisiti fondamentali di un sistema OBDA:

1. G è descritto in un linguaggio ontologico che permette un giusto trade-off tra potere espressivo e complessità computazionale
2. La logica adottata deve consentire di delegare il query answering al DBMS relazionale
3. Il sistema di mapping deve essere in grado di affrontare il problema del'impedence mismatch
4. I client devono poter esprimere queries esclusivamente basandosi sulla conoscenza del Global Schema G

3.1 Ontologie

Un’ontologia è una concettualizzazione del dominio di interesse di un sistema informativo espressa in un qualche linguaggio formale.

Schemi concettuali UML e ER sono ontologie graph-based.

Siamo interessati in questo contesto a ontologie descritte in logica.

Siamo particolarmente interessati alle Description Logic, che risultano adatte a descrivere e ragionare su conoscenza strutturata.

3.2 Ontologie in DL

Livello intensionale (TBox):

Concept: rappresenta una collezione di istanze

Attribute: qualifica un altro elemento (Role o Concept)

Role: esprime una relazione tra concepts

Assertion: condizione a livello intensionale che deve essere soddisfatta a livello estensionale

Livello estensionale (ABox):

Istance: rappresenta un'istanza di un Concept

Fact: rappresenta una relazione tra due o più istanze

4.1 DL-Lite$_A$ Syntax

Alfabeto: simboli per Atomic Concepts (A), Value Domains (T_i), Atomic Roles (P), Atomic Attributes(U_c) e costanti ($c_i \in \Gamma$).

Expression:

1. Concept expressions:

$B := A \mid \exists Q \mid \delta(U_c)$	Basic Concept
$C := T_C \mid B \mid \neg B \mid \exists Q.C$	General Concept

2. Value-domain expressions:

$E := \rho(U_c)$	Basic value-domain
$F := T_D \mid T_1 \mid \cdot \cdot \cdot \mid T_n$	Value-domain expression

3. Role expressions:

$Q := P \mid P -$	Basic Role
$R := Q \mid \neg Q$	General Role

4. Attribute expressions:

$V_c := U_c \mid \neg U_c$	General Attribute

4.2 DL-Lite$_A$ Semantic

La semantica delle espressioni DL-Lite$_A$ è definita in termini di interpretazioni della logica del prim'ordine.

Ogni value-domain T_i è interpretato come l'insieme val(T_i) di valori del corrispondente RDF data type.

Ogni costante C_i è interpretata come uno specifico valore denotato val(C_i) $\in$ val(T_i).

Importante notare che: $i \neq j$ implica che val(T_i)$\cap$val(T_j)=$\emptyset$
cioè DL-Lite$_A$ adotta la *unique name assumption*.

4.3 DL-Lite$_A$: objects vs values

DL-Lite$_A$ pone particolare enfasi sulla distinzione tra oggetti e valori, distinguendo:

- Concept da value-domains, dove un concept è un'astrazione di un insieme di oggetti, mentre un value-domains denota un insieme di valori concreti
- Attributes da roles, dove un role denota una relazione binaria tra oggetti, mentre un attribute denota una relazione binaria tra oggetti e valori

Dato un attributo U_c denotiamo con $\delta(U_c)$ il dominio di U_c, cioè l'insieme di oggetti che U_c mette in relazione con i valori; mentre denotiamo con $\rho(U_c)$ il range di U_c, ovvero l'insieme di valori che Uc mette in relazione con gli oggetti.

$\delta(U_c)$: il dominio di un attributo U_c è un Concept

$\rho(U_c)$: il range di un attributo U_c è un value-domain.

4.4 DL-Lite$_A$ Ontologies

Un'ontologia O in DL-Lite$_A$ è formata da una coppia <T,A>, dove T è una DL-Lite$_A$ TBox mentre A rappresenta una DL-Lite$_A$ ABox.

Una **Abox** in DL-Lite$_A$ è un insieme di asserzioni chiamate *membership assertions* della forma:
A(a), P(a, b), Uc(a, b) dove a e b sono costanti nell'alfabeto Γ.

Intensional assertions:

$B \sqsubseteq C$	(concept inclusion assertion)
$Q \sqsubseteq R$	(role inclusion assertion)
$E \sqsubseteq F$	(value-domain inclusion assertion)
$U_C \sqsubseteq V_C$	(attribute inclusion assertion)
(funct Q)	(role functionality assertion)
(funct U_C)	(attribute functionality assertion)

4.5 DL-Lite$_A$ TBox

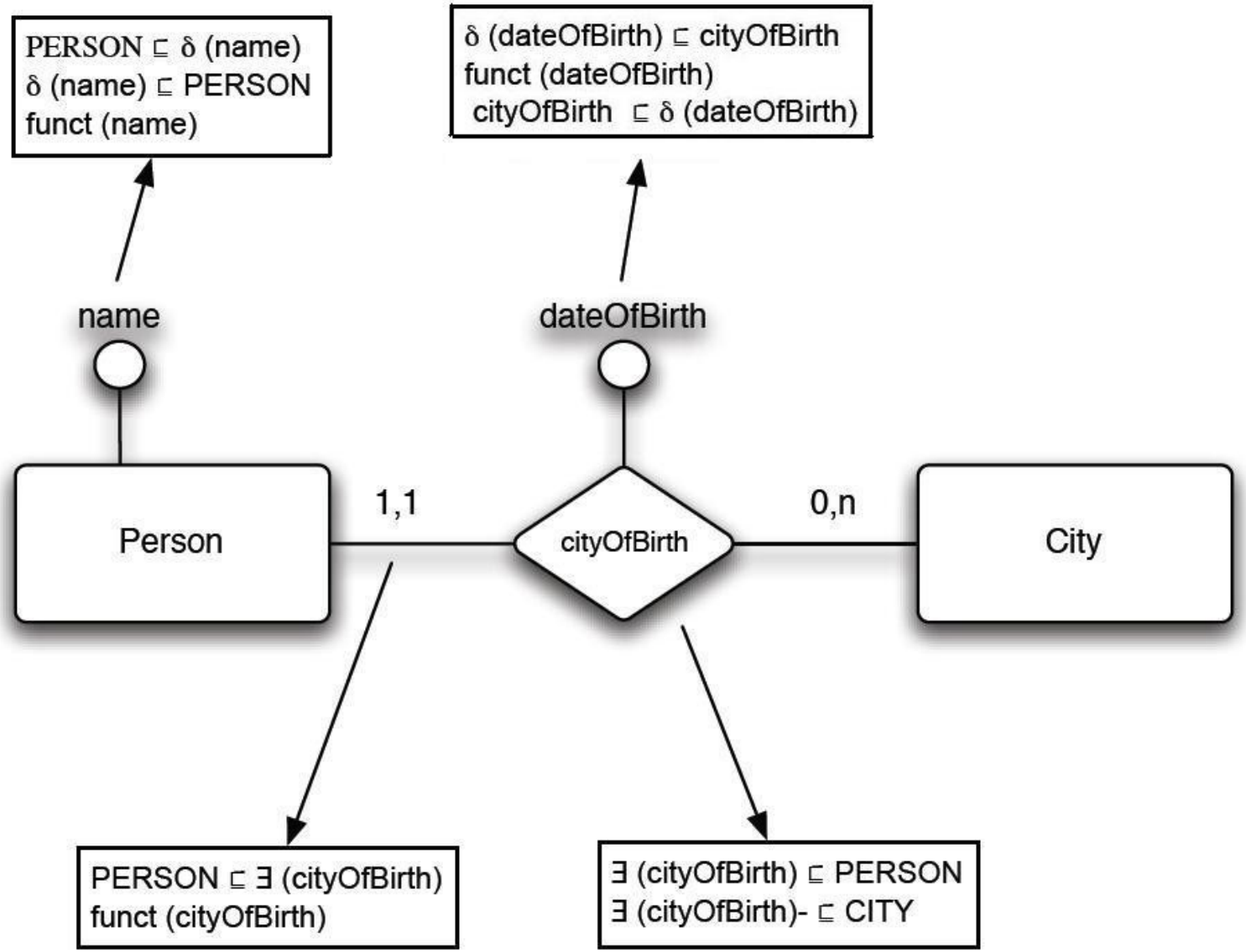
Una DL-Lite$_A$ TBox è un insieme finito T di intensional assertions che soddisfano la condizione che **ogni identifying property in T è primitiva in T.**

Un atomic attribute U_c (o basic role Q) è detto **indentifying property** se T contiene l'asserzione (funct U_c) (o rispettivamente (funct Q)).

Un atomic attribute U_c (o basic role Q) è detto **primitivo** se non appare positivamente nella parte destra di una inclusion assertions del tipo $Y \sqsubseteq U_c$ (o rispettivamente $Y \sqsubseteq Q$) in T, e non appare in una espressione della forma $\exists Q.C$ in T.

In parole povere questo significa che in DL-Lite$_A$ non è possibile specializzare le identifying properties.

4.6 DL-Lite$_A$ VS ER



4.7 DL-Lite$_A$ Complexity

OWL lo standard W3C per la rappresentazione di ontologie per il Web semantico, non è adatto nei sistemi OBDA.

OWL senza opportune restrizioni ha complessità computazionale esponenziale rispetto la dimensione dei dati.

DL-Lite$_A$ risulta avere complessità LOGSPACE rispetto alla dimensione dei dati della ABox.

Ancora più importante DL-Lite$_A$ permette di delegare il query answering al DBMS relazionale (Data Layer).

Per quanto riguarda il potere espressivo in DL-Lite$_A$ è possibile esprimere role attribute, cosa non possibile in OWL-DL.

5.1 Mapping

Un'ontologia con mapping in DL-Lite$_A$ è caratterizzata da una tripla Om=<T,M,DB> tale che:

- T è una TBox DL-Lite$_A$
- DB è un database relazionale
- M è un insieme di *mapping assertion*, partizionato in due insieme Mt e Ma dove:
  - Mt è un insieme di *typing mapping assertion* della forma $\Phi \rightarrow$ Ti dove Φ è una query di arità 1 sul DB (proiezione su una colonna) e Ti è un data type DL-Lite$_A$
  - Ma è un insieme di *data-to-object mapping assertions* della forma $\Phi \rightarrow \psi$ dove Φ è una query SQL arbitraria sul DB e ψ è una conjunctive query su T che coinvolge variable terms. Un variable term è del tipo f(z) con f simbolo di funzione di arità n e z ennupla di variabili.

5.2 Impedence Mismatch

Le sorgenti dati (relazionali) memorizzano valori.

Le istanze dei Concept sono invece oggetti.

Ogni oggetto è denotato un identificatore.

DL-Lite$_A$ tiene ben separati i valori dagli identificatori:
gli object identifier sono termini della logica costruiti a partire dai valori utilizzando di volta in volta un oportuno simbolo di funzione f.

Il sistema di mapping definito precedentemente risolve quindi in modo corretto il problema dell'impedence mismatch

5.3 CWA vs OWA

I DBMS relazionali adottano la Closed World Assumption:

Si assume di avere conoscenza completa sul mondo, cioè si considera vero quello che è asserito nella base di dati e falso tutto il resto

Nelle ontologie si adotta invece la Open World Assumption:

Si assume di avere a che fare con informazione incompleta, quindi si considera vero tutto quello che è asserito nell'ABox mentre sul resto non si ha conoscenza, potrebbe essere sia vero sia falso.

5.4 CWA vs OWA

Mapping assertion: $\Phi \rightarrow \psi$

Poiché Φ è una query SQL che viene valutata direttamente dal DBMS, la parte sinistra dell'asserzione rispetta CWA

L'implicazione materiale presente nell'asserzione invece rispetta OWA infatti impone che tutti i modelli di Φ siano anche modelli di ψ ma non dice nulla sul viceversa!

In altre parole, tutte le tuple di Φ soddisfano ψ ma ci potrebbero essere anche altre tuple che soddisfano ψ .

6.1 Query Answering

Cosa significa valutare una query su un'ontologia?

PERSON

Name
Claudio
Ilaria
Giuseppe

STUDENT

Name
Valerio
Ilaria

q(x): Person(x) $\wedge \neg$ Student(x)

Il risultato di questa query è diverso a seconda dell'assunzione

In CWA: {Claudio, Giuseppe}
In OWA: {}

6.2 Query Answering

In Open World Assumption il risultato di una query in FOL dipende dal modello.

Mentre in CWA la FOL risulta decidibile, in OWA è quindi indecidibile.

Per interrogare un'ontologia utilizziamo un frammeto della FOL che risulta decidibile: le Conjunctive Queries, e unione di esse (UCQ).

In questo modo però il potere espressivo risulta notevolmente limitato a causa della mancanza dell'operatore NOT.

6.3 Conjunctive Queries

Una *conjunctive query* in DL-Lite$_A$ è una formula del tipo:

$$q(\mathbf{x}) \leftarrow conj(\mathbf{x}, \mathbf{y})$$

dove:

- q(x) è l'head della query, conj(x,y) è il body
- l'arità di q coincide con l'arità di x
- x è una tupla di variabili non quantificate (libere)
- y è una tupla di variabili quantificate esistenzialmente
- conj è una congiunzione di atomi della forma:

$$A(x),\ P(x),\ D(x),\ Uc(x,y),\ x=y$$

6.4 Certain Answers

Data un'ontologia O=<T,A>, un'interpretazione I per O e una CQ q(x) si dice ***answer*** di q(x) su I, denotata con q^I l'insieme delle tuple t di costanti tale che la valutazione del body è vera nell'interpretazione I.

Rispondere a una query q su un'ontologia O significa restituire solo le ***certain answers***; cioè le tuple t tali che $t^I \in q^I$ per ogni modello I di O.

6.5 Reasoning in DL-Lite$_A$

Sia O=<T,A> un'ontologia in DL-Lite$_A$. Possiamo rappresentare l'ABox A con un database relazionale db(A). Db(A) contiene:

- una relazione unaria T_A per ogni atomic concept A in T.
  La tupla $t \in T_A$ se e solo se l'asserzione $A(t) \in A$.

- una relazione binaria T_P per ogni atomic role P in T.
  La tupla $t \in T_P$ se e solo se l'asserzione $P(t) \in A$

- una relazione binaria T_U per ogni atomic concept attribute U in T.
  La tupla $t \in T_U$ se e solo se l'asserzione $U(t) \in A$

6.6 Complessità Reasoning in DL-Lite$_A$

Il reasoning su un'ontologia in DL-Lite$_A$ può essere ridotto alla valutazioni di particolari queries su db(A).

Tali queries possono essere espresse in SQL, e valutate con un DBMS relazionale:

Per questo motivo il reasoning sull'ontologia ha la stessa complessità computazionale del SQL, ovvero **LOGSPACE** nella dimensione dei dati.

6.7 Soddisfacibilità di O=<T,A>

Soddisfacibilità ⇔ *Violates(T)*

Violates(T) è una FOL-query che chiede per ogni costante in A se sono violati:

- Vincoli espliciti corrispondenti ad asserzioni di funzionalità e asserzioni di disgiunzione in T

- Vincoli impliciti che seguono dalla semantica di T:
  - ogni concept è in disgiunzione da ogni dominio
  - per ogni coppia T_i,T_j di rdfDataType, T_i e T_j sono disgiunti.

Sostituendo ogni predicato X in Violates(T) con Tx si ottiene una query equivalente ***ViolatesDB(T)*** espressa in SQL su db(A).

6.8 Query Answering su O=<T,A>

Query Answering ⇔ *PerfectRef(Q,T)*

PerfectRef(Q,T) prende in input una Union of Conjunctive Query Q e una TBox T e riformula Q in una nuova query Q'

La risposta di Q' rispetto a <Ø,M,DB> coincide con la certain answer di Q rispetto a <T,M,DB>

Sostituendo ogni predicato X in *PerfectRef(Q,T)* con Tx si ottiene una query equivalente ***PerfectRefDB(Q,T)*** espressa in SQL su db(A).

7.1 Reasoning in DL-Lite$_A$ con mapping

Approccio Bottom-up

Utilizza i mapping per produrre una ABox fisica a partire dai dati delle sorgenti che vengono quindi duplicati.
Si applicano poi gli algoritmi di query answering all'ontologia O=<T,A(M,DB)>

Problema 1: Complessità PTIME nella dimensione del database

Problema 2: Il database diventa indipendente dall'ontologia che non riflette gli aggiornamenti nel tempo; questo implica la necessità di un meccanismo per tenere ABox e DB sincronizzati

7.2 Reasoning in DL-Lite$_A$ con mapping

Approccio Top-down

L'ABox non viene materializzata ma mantenuta virtuale.

Durante il reasoning i mapping vengono considerati on-the-fly.

In questo modo la complessità resta LOGSPACE.

7.3 Split version di un'ontologia

Data un'ontologia O_m= <T,M,DB> in DL-Lite$_A$ si dice versione split di O_m, denotata come Split(O_m)=<T,M',DB>, una nuova ontologia ottenuta da O_m costruendo un nuovo insieme di mappings M' tale che:

- ogni typing assertion di M sarà presente anche in M'

- per ogni mapping assertion $\Phi \rightarrow \psi$ in M e per ogni atomo $X \in \psi$, M' conterrà la mapping assertion $\Phi' \rightarrow \psi$ dove Φ' è la proiezione di Φ sulle variabili che occorrono in X.

Si dimostra che O_m è logicamente equivalente a Split(O_m).

Inoltre il processo di splitting è PTIME nella dimensione dei mappings.

7.4 Virtual ABox

Data un'ontologia O_m= <T,M,DB> in DL-Lite$_A$ la ABox virtuale è una ABox dove le asserzioni sono computate on-the-fly durante il reasoning applicando le mapping assertions ai dati presenti nel DB.

Presa una mapping assertion $m \in M$: $\Phi \rightarrow \psi$ si dice ABox virtuale generata da m

$$A(m,DB) = \{X[x/v] \mid v \in ans(\Phi, DB)\}$$

Dove X[x/v] denota il ground atom ottenuto da X(x) sostituendo la ennupla di variabili x con la ennupla di costanti v.

La ABox virtuale di O_m è un insieme di asserzioni del tipo:

$$A(M,DB) = \{A(m,DB) \mid m \in M\}$$

Si dimostra che O_m=<T,M,DB> è logicamente equivalente a O_m=<T,A(M,DB)>.

7.5 Unfolding

Data O_m e una CQ Q, la fase di Unfolding consiste nel computare una nuova query Q' che è una query SQL sulle sorgenti dati relazionali.

L'insieme delle tuple che verranno restituite valutando Q' sulle sorgenti coincide con l'insieme delle tuple ottenute valutando Q su db(A(M,DB)).

UnfoldDB(O_m,Q): prende in ingresso un'ontologia scritta in DL-Lite$_A$ e una query Q sull'ontologia e restituisce l'insieme di risultati che descrive le query da mandare al db e le sostituzioni da applicare in ordine al risultato per ottenere le risposte a Q.

7.6 Soddisfacibilità con mapping

Algorithm *Sat(O_m)*

Input: DL-Lite$_A$ ontology with mappings O_m = <T ,M,DB>
Output: true or false
Q^s ← Violates(T);
S' ← UnfoldDB(Q^s,O_m);
Q' ← false;
for each ansΘ ← q' ∈ S' **do**
    Q' ← Q' ∪ {q'} ;
return not(ans(Q',DB))

Data O_m si dimostra che Sat(O_m) termina e l'ontologia è soddisfacibile se e solo se l'algoritmo ritorna true.

7.7 Query Answering con mapping

Algorithm *Answer(Q,O_m)*

Input: DL-Lite$_A$ ontology with mappings O_m = <T ,M,DB>, UCQ Q over O_m

Output: set of tuples R^s

if O_m is not satisfiable
then return AllTup(Q,O_m)
else
$Q^p \leftarrow \cup_{qi \in Q}$ PerfectRef(q_i,T);
S' ← UnfoldDB(Q^p,O_m)
$R^s \leftarrow \emptyset$;
for each ansΘ ← q' ∈ S' **do**
$R^s \leftarrow R^s \cup$ ans(q',DB)Θ;
return R^s

7.8 Complessità computazionale

Le mapping assertions sono della forma $\Phi \rightarrow \psi$ dove Φ è una query SQL sul database relazionale sottostante. Si assume che tale query appartenga alla First Order Logic e in quanto tale è **LOGSPACE** rispetto alla dimensione dei dati nel DB.

Sia O_m=<T,M,DB> un'ontologia in DL-Lite$_A$ con mappings e Q una UCQ su O_m.
La funzione UnfoldDB(Q,O_m) impiega tempo esponenziale rispetto alla dimensione di Q e polinomiale rispetto alla dimensione di M.

7.9 Complessità computazionale

Data un'ontologia O_m=<T,M,DB> in DL-Lite$_A$ con mappings, l'algoritmo Sat(O_m) è **LOGSPACE** nella dimensione del DB (data complexity) e viene eseguito in tempo polinomiale nella dimensione di M e in tempo polinomiale nella dimensione di T.

Data un'ontologia O_m=<T,M,DB> in DL-Lite$_A$ con mappings e una UCQ Q su tale ontologia, l'algoritmo Answer(Q,O_m) è **LOGSPACE** nella dimensione dei dati (data complexity), PTIME nella dimensione di M, EXPTIME nella dimensione di Q e PTIME nella dimensione di T.

8.1 Linguaggi di query epistemici

Ricordiamo che in OWA il query answering in FOL dipende dal modello, per questo motivo ci siamo limitati a considerare Conjunctive Queries.

Limitazione CQ: mancanza operatore NOT.

Per aumentare il potere espressivo delle queries su ontologie è necessario introdurre il concetto di queries epistemiche.

Si utilizza l'operatore epistemico K che per formalizzare lo stato epistemico della KB.

Kφ siginifica: sia sa che φ è valida dalla base di conoscenza.

8.2 Epistemic Query Language

EQL = FOL + operatore epistemico (di conoscenza minimale) su KB

$\Phi := A(t) \mid P(t_1,\ldots,t_n) \mid t_1=t_2 \mid \neg\Phi \mid \Phi_1 \wedge\Phi_2 \mid \exists x.\ \Phi \mid K\Phi$

Una formula di EQL si valuta su un'interpretazione epistemica E,w dove E rappresenta tutti i modelli della base di conoscenza e w è un modello di E.

Formule soggettive: contengono K, si basano su ciò che la base di conoscenza conosce.

Formule oggettive: non contengono K, si basano su ciò che è vero in assoluto.

8.3 SparSQL

Il linguaggio per esprimere le queries EQL è SparSQL.

SparSQL = SQL + SPARQL

Query SparSQL:

SELECT *ListaAttributiOEspressioni*
FROM (*sparqltable(< QuerySparql >) alias*) +

[**where** *CondizioniSemplici*]
[**group by** *ListaAttributiDiRaggruppamento*]
[**having** *CondizioniAggregate*]
[**order by** *ListaAttributiDiOrdinamento*]

8.4 SparSQL

La clausola FROM estrae lo stato epistemico dell'ontologia attraverso tabelle (sparqltable).

Il linguaggio scelto per estrarre conoscenza dall'ontologia è SPARQL di cui si utilizza soltanto un frammento corrispondende ad UCQs.

Le tabelle SPARQL corrispondono a queries soggettive in EQL che utlizzano l'operatore K.

Semantica SparSQL

$$q(\mathbf{x}) : - \Phi (\boldsymbol{K} \alpha_1,\ldots, \boldsymbol{K}\alpha_n)$$

- Φ è una query SQL
- $\mathbf{K}\alpha_i$ è una UCQ espressa in SPARQL
- **x** è il vettore delle variabili che devono essere restituite dalla query

SAPIENZA
UNIVERSITÀ DI ROMA

9.1 Implementazione IMDB

1. Scrittura ontologia in DL-Lite$_A$
2. Scrittura dei mapping
3. Implementazione dell’ontologia
4. Interrogazione dell'ontologia

9.2 Ontologia in DL-Lite$_A$

Dimensioni dell'ontologia

12 Atomic Concept

60 Concept Attribute

56 Atomic Role

27 Role Attribute

522 Asserzioni

52 Mappings

Dimensioni dei files XML

TBox: 7.153 righe

Mapping: 2.926 righe

9.3 DL-Lite$_A$ ⇒ XML

ACTOR ⊑ PERSON

```
<inclusionAssertion>
    <basicC>
        <atomicC>Actor</atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>

    <atomicC>Person</atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

9.4 DL-Lite$_A$ ⇒ XML

∃(cityOfBirth) ⊑ PERSON

```
<inclusionAssertion>
    <basicC>
        <exists>
            <basicR dir="direct">

    <atomicR>cityOfBirth</atomicR>
            </basicR>
        </exists>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <atomicC>Person</atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

9.5 DL-Lite$_A$ ⇒ XML

∃(cityOfBirth)- ⊑ CITY

```
<inclusionAssertion>
    <basicC>
        <exists>
            <basicR dir="inverse">

    <atomicR>cityOfBirth</atomicR>
            </basicR>
        </exists>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <atomicC>City</atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

9.6 DL-Lite$_A$ ⇒ XML

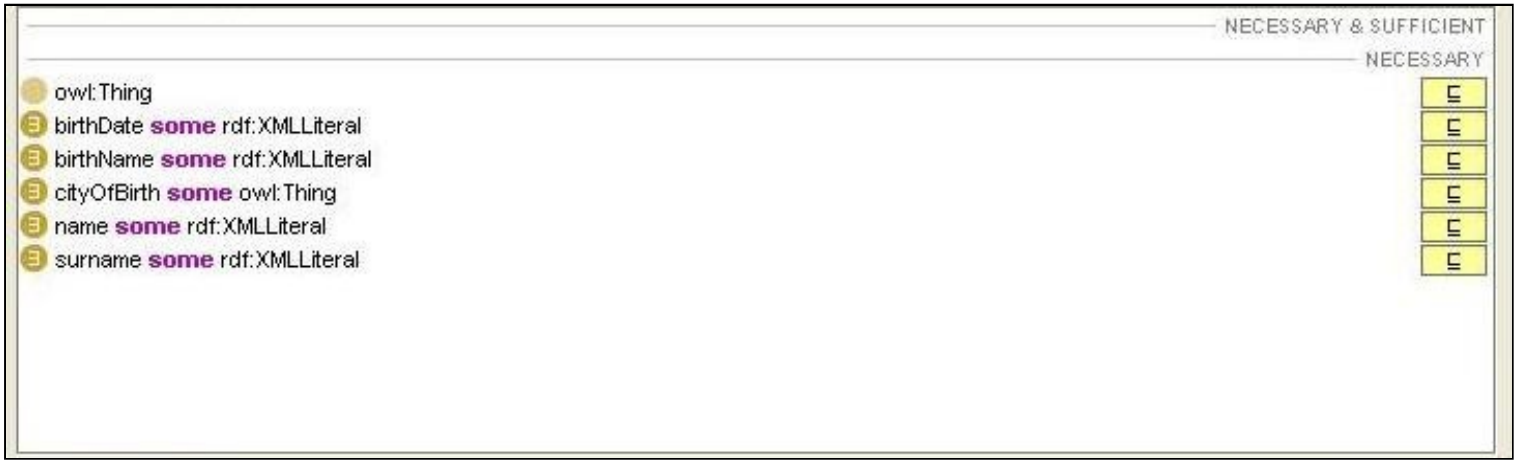
funct (cityOfBirth)

```
<funct>
    <basicR dir="direct">

    <atomicR>cityOfBirth</atomicR>
    </basicR>
</funct>
```

9.7 DL-Lite$_A$ ⇒ Protégé

PERSON ⊑ δ (name)
PERSON ⊑ δ (surname)
PERSON ⊑ δ (birthDate)
PERSON ⊑ δ (birthName)
PERSON ⊑ ∃ (cityOfBirth)



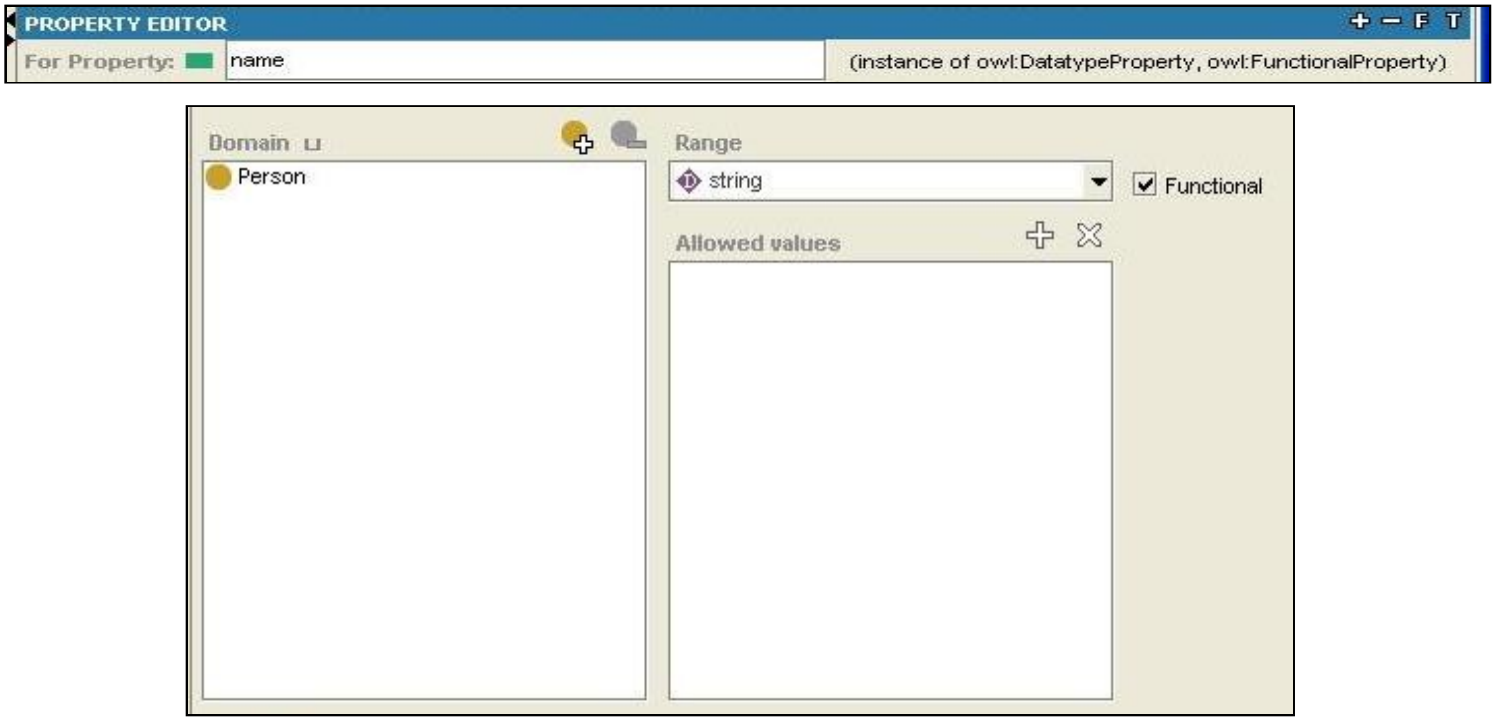
9.8 DL-LiteA ⇒ Protégé

ACTOR ⊑ PERSON

ACTOR ⊑ ∃ (actor2character)

9.9 DL-LiteA ⇒ Protégé

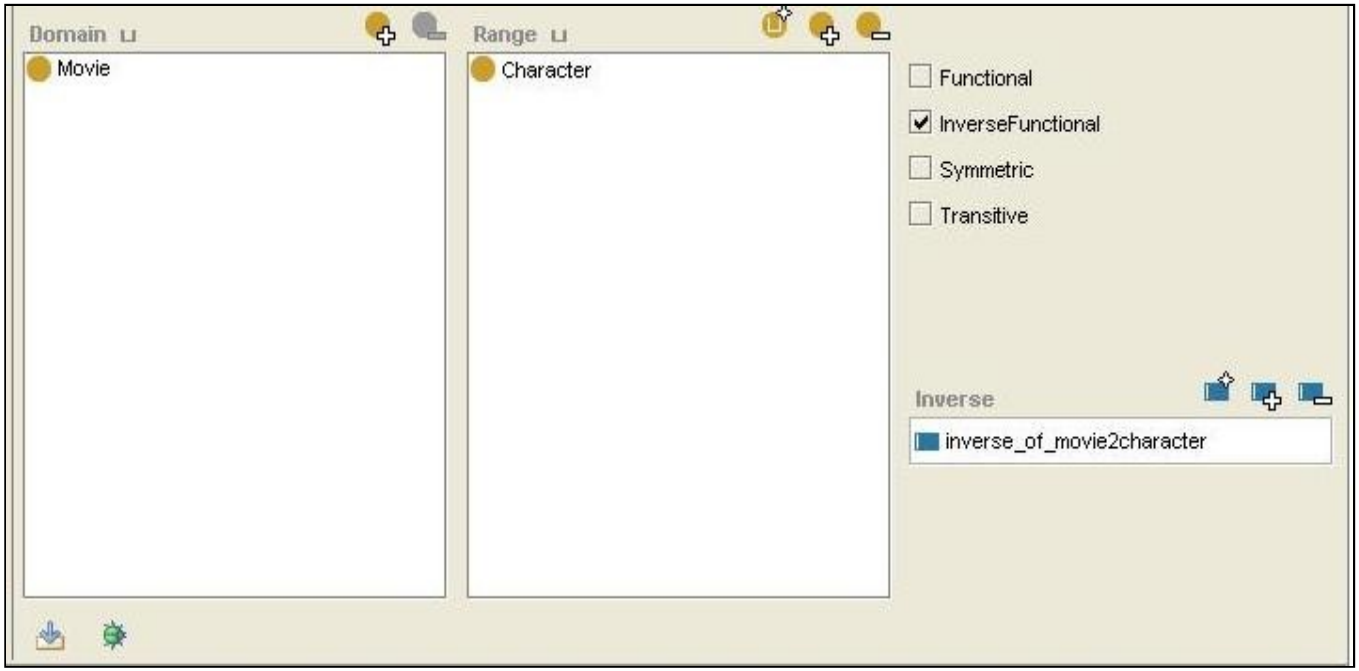
δ (name) ⊑ PERSON
funct (name)
ρ (name) ⊑ xsd: string



9.10 DL-LiteA ⇒ Protégé

$\exists$ (movie2character) $\sqsubseteq$ MOVIE

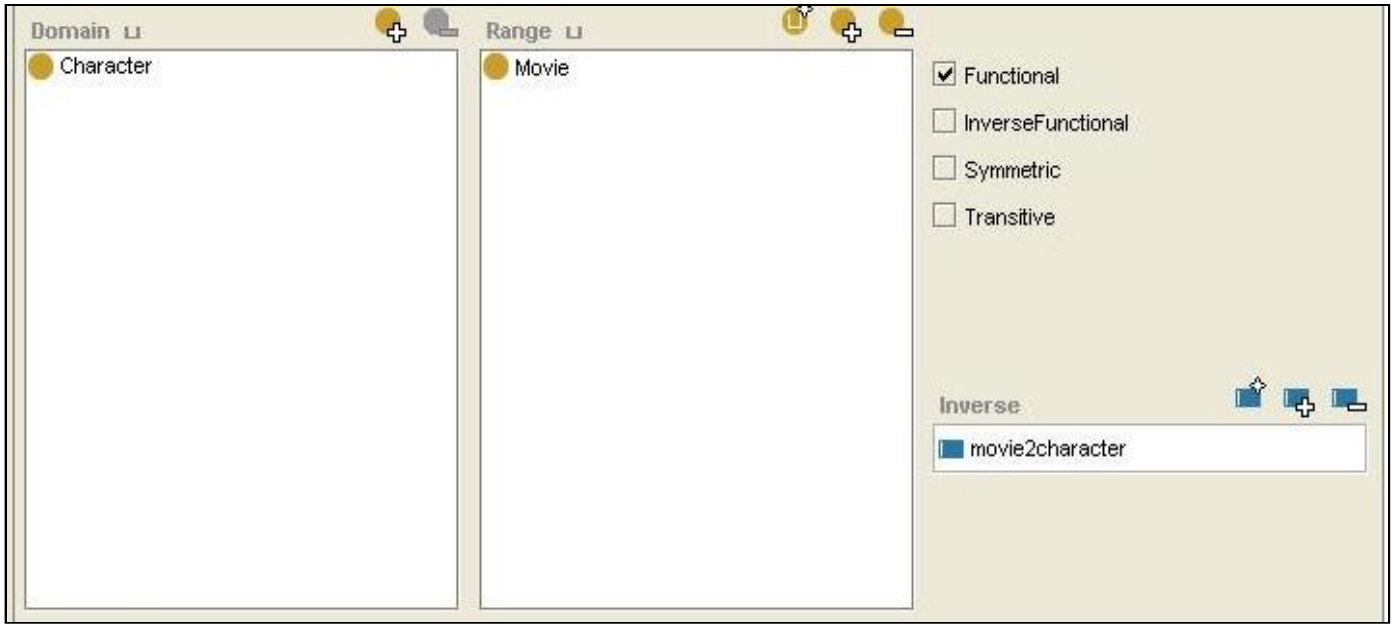
MOVIE $\sqsubseteq \exists$ (movie2character)



9.11 DL-LiteA ⇒ Protégé

$\exists$ (movie2character)- $\sqsubseteq$ CHARACTER

funct (movie2character)-



10.1 Mapping Person

Prima soluzione: Errata

PERSON ⊑ δ (name)
δ (name) ⊑ PERSON
funct (name)
PERSON ⊑ δ (surname)
δ (surname) ⊑ PERSON
funct (surname)

∃ (directedBy) ⊑ PERSON
∃ (directedBy)- ⊑ MOVIE
∃ (writtenBy) ⊑ PERSON
∃ (writtenBy)- ⊑ MOVIE

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
director
x → director(name,surname)
y → name xsd:String
z → surname xsd: String

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
writer
x → writer(name,surname)
y → name xsd:String
z → surname xsd: String

10.2 Mapping Person

Prima soluzione:
Errata

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
director
x → director(name,surname)
y → name xsd:String
z → surname xsd: String

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
writer
x → writer(name,surname)
y → name xsd:String
z → surname xsd: String

Grave problema:

Presenza di duplicati in Person!

L'utilizzo di funtori diversi come identificatori del Concept Person nei diversi mapping causa la presenza di più istanze della stessa persona, dal momento che nelle tabelle del DB (director,writer) possono essere presenti le stesse tuple.

10.3 Mapping Person

Seconda soluzione

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
director
x → director(name,surname)

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
writer **WHERE** name,surname
NOT IN (**SELECT** name,surname
FROM director)
x → writer(name,surname)

Si utlizzano funtori diversi in ogni mapping

Mediante un'opportuna query SQL si eliminano i duplicati (NOT IN)

Problemi:

- Eccessiva complessità delle queries dal momento che le tabelle da considerare sono 10
- Possiamo fare assunzioni sull'ordine in cui i mappings verranno considerati?

10.4 Mapping Person

Terza soluzione:
ADOTTATA

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
director
x → per(name,surname)

Person(x) ∧name(x,y) ∧
surname(x,z)
←
SELECT name, surname **FROM**
writer x → per(name,surname)

Si utlizza uno stesso funtore in tutti i mapping di Person

In questo modo i duplicati non vengono considerati grazie all'unicità degli identificatori

10.5 Mapping Actor

Prima soluzione: Errata

ACTOR ⊑ PERSON

Actor(x) ∧name(x,y) ∧surname(x,z)
←
SELECT id, name, surname **FROM**
actor
x → per(id)
y → name xsd:String
z → surname xsd: String

In un'ontologia l'identificatore di concetto è un termine della logica f(t) dove t normalmente è il valore che identifica l'istanza nel DB

Poiché la chiave di actor è id, come prima soluzione si era scelto il funtore per(id)

Problema: Poiché le altre persone sono identificate da per(name,surname) e un writer può essere anche un actor, sorge nuovamente il problema dei duplicati

10.6 Mapping Actor

Seconda soluzione: ADOTTATA

ACTOR ⊑ PERSON

Actor(x) ∧name(x,y) ∧surname(x,z)
←
SELECT name, surname **FROM**
actor
x → per(name, surname)
y → name xsd:String
z → surname xsd: String

Abbiamo deciso di identificatori anche gli Actor con per(name,surname)

Questa soluzione risolve il problema dei duplicati e non comporta perdita di informazione dal momento che nella base di dati non esistono due attori con la stessa coppia <name,surname>

Infatti gli omonimi sono differenziati attraverso l'aggiunta di un numero accanto al nome

Esempio:
<Mario(I), Rossi>
<Mario(II), Rossi>

10.7 Mapping TVSerie

Soluzione: ADOTTATA

TVSerie ⊑ Movie

TvSerie(x) ∧movieTitle(x,y) ∧episodeTitle(x,e)
←
SELECT id, SUBSTRING(SUBSTRING_INDEX(title, ':', 1),5) as titolo_serie, SUBSTRING_INDEX(title, ':', -1) as titolo_episodio
FROM movies
WHERE title like '%(TV)%‘

x → mov(ID)
y → titolo_serie xsd:String
e → titolo_episodio xsd:String

Nel DB le serie tv sono inserite nella stessa tabella dei film ma identificate dal tag (TV) nel titolo

11.1 Test

Conjunctive query testata:

q(x,y): movie2character(x,y)

Expanded query:
q(x,y): movie2character(x,y) Non c'è espansione!

Unfolded query sul DB:

```
select distinct alias_0.term1, alias_0.term2 from (
select distinct concat('mov(',movie,')') as term1,
   concat('char(',role,')') as term2 from actor2movie)
   alias_0
```

Tempo esecuzione query maggiore di 2 ore

(in realtà non termina perché lo swapping manda in stallo la macchina)

11.2 Test

Eliminazione dell'operatore DISTINCT:

```
select alias_0.term1, alias_0.term2 from (
select concat('mov(',movie,')') as term1,
   concat('char(',role,')') as term2 from actor2movie)
   alias_0
```

La query senza distinct termina restituendo il risultato corretto in soli 297 secondi!

Conclusione:
MySQL non riesce a gestire queries con l'operatore DISTINCT in maniera efficiente.

SOLUZIONE:
Eliminato l'operatore **DISTINCT** nel sorgente dell'Unfolding di DIG-Mastro.

11.3 Example Query SPARQL

Nomi e cognomi di tutti gli Actor:

```
SELECT $y $z WHERE {
       $x rdf:type 'Actor'.
       $x :name $y.
       $x :surname $z
}
```

Titolo del film 140438 e i personaggi del film:

```
SELECT $x $y WHERE {
       'getMovieObject(140438)' :movieTitle $x.
       'getMovieObject(140438)' :movie2character $y
}
```

11.4 Example Query EQL

Tutte le persone:

```
SELECT person.x, person.y,person.z
FROM sparqltable(
       SELECT $x $y $z
       WHERE { $x rdf:type 'Person'.
              $x name $y.
              $x surname $z }) person;
```

Tutti i film che NON sono serietv:

```
SELECT movies.x
FROM sparqltable(
              SELECT $x
              WHERE {$x rdf:type 'Movie'}) movies
WHERE movies.x NOT IN (
              SELECT series.y
              FROM sparqltable ( SELECT $y
              WHERE { $y rdf:type 'TVSerie'}) series );
```

Bibliografia

1. Linking Data to Ontologies. Journal on Data Semantics, 2008.

2. Ontologybased Database Access. SEBD, 2007.

3. Linking Data to Ontologies: The Description Logic DLLiteA. OWLED', 2006

4. Mastroi:Efficient integration of relational data through DL ontologies. DL07.

5. Data Integration Through DLLiteA Ontologies. SDKB'08.

6. Realizing Ontology Based Data Access: A Plugin for Protégé. IIMAS 08

7. Ontologybased database access with DIGMastro and the OBDA Plugin for Protégé. OWLED 2008.

8. Ontologie per accesso ai dati: tecniche per interrogazioni del primo ordine basate sulla chiusura dinamica della conoscenza. 2007.