Ontologies - Querying Data through Ontologies

Serge Abiteboul   Ioana Manolescu   Philippe Rigaux
Marie-Christine Rousset   Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

November 17, 2011

Outline

1. Introduction
   - The Semantic Web
   - Ontologies and Reasoning
   - Illustration
2. 3 ontology languages for the Web
3. Reasoning in Description Logics
4. Querying Data through Ontologies
5. Conclusion

The Semantic Web

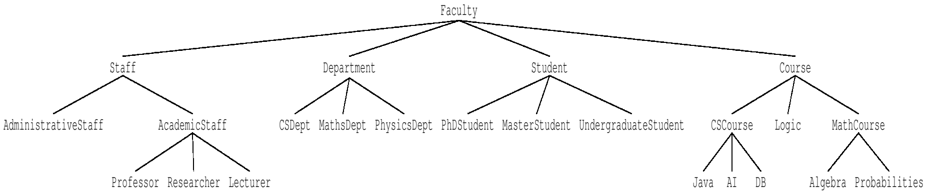
- A Web in which the resources are semantically described
  - annotations give information about a page, explain an expression in a page, etc.
- More precisely, a resource is anything that can be referred to by a URI
  - a web page, identified by a URL
  - a fragment of an XML document, identified by an element node of the document,
  - a web service,
  - a thing, an object, a concept, a property, etc.
- Semantic annotations: logical assertions that relate resources to some terms in pre-defined ontologies

Ontologies

- Formal descriptions providing human users a shared understanding of a given domain
  - A controlled vocabulary
- Formally defined so that it can also be processed by machines
- Logical semantics that enables reasoning.
- Reasoning is the key for different important tasks of Web data management, in particular
  - to answer queries (over possibly distributed data)
  - to relate objects in different data sources enabling their integration
  - to detect inconsistencies or redundancies
  - to refine queries with too many answers, or to relax queries with no answer

Classes and class hierarchy

- Backbone of the ontology
- AcademicStaff is a Class
- (A class will be interpreted as a set of objects)
- AcademicStaff isa Staff
- (isa is interpreted as set inclusion)



Relations

- Declaration of relations with their signature
- (Relations will be interpreted as binary relations between objects)
- `TeachesIn`(`AcademicStaff`, `Course`)
  - if one states that "*X* `TeachesIn` *Y*", then *X* belongs to `AcademicStaff` and *Y* to `Course`,
- `TeachesTo`(`AcademicStaff`, `Student`),
- `Leads`(`Staff`, `Department`)

Instances

- Classes have instances
- `Dupond` is an instance of the class `Professor`
- it corresponds to the fact: `Professor`(`Dupond`)

- Relations also have instances
- (`Dupond`,`CS101`) is an instance of the relation `TeachesIn`
- it corresponds to the fact: `TeachesIn`(`Dupond`,`CS101`)

- The instance statements can be seen as (and stored in) a database

Ontology = schema + instance

- Schema
  - The set of class and relation names
  - The signatures of relations and also constraints
  - The constraints that are used for two purposes
    - ★ checking data consistency (like dependencies in databases)
    - ★ inferring new facts
- Instance
  - The set of facts
  - The set of base facts together with the inferred facts should satisfy the constraints
- Ontology (i.e., Knowledge Base) = Schema + Instance

Outline

1. Introduction
2. 3 ontology languages for the Web
3. Reasoning in Description Logics
4. Querying Data through Ontologies
5. Conclusion

3 ontology languages for the Web

- RDF: a very simple ontology language
- RDFS: Schema for RDF
  - Can be used to define richer ontologies
- OWL: a much richer ontology language

- We next present them rapidly
- We will introduce further a family of ontology languages: Description logics

RDF: Resource Description Framework

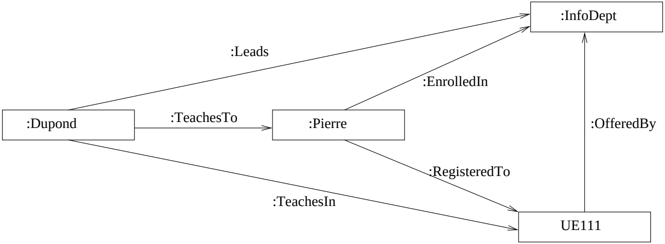
- RDF facts are triplets

$\langle$ `:Dupond :Leads :CSDept` $\rangle$
$\langle$ `:Dupond :TeachesIn :UE111` $\rangle$
$\langle$ `:Dupond :TeachesTo :Pierre` $\rangle$
$\langle$ `:Pierre :EnrolledIn :CSDept` $\rangle$
$\langle$ `:Pierre :RegisteredTo :UE111` $\rangle$
$\langle$ `:UE111 :OfferedBy :CSDept` $\rangle$

- Linked open data: publish open data sets on the Web
  - By September 2011, 31 billions RDF triplets

RDF graph

- A set of RDF facts defines
  - a set of relations between objects
  - an RDF graph where the nodes are objects:



RDF semantics

- A triplet $\langle s \;\; P \;\; o \rangle$ is interpreted in first-order logic (FOL) as a fact $P(s, o)$
- Example:

$$\begin{array}{l}\texttt{Leads}(\texttt{Dupond}, \texttt{CSDept}) \\ \texttt{TeachesIn}(\texttt{Dupond}, \texttt{UE111}) \\ \texttt{TeachesTo}(\texttt{Dupond}, \texttt{Pierre}) \\ \texttt{EnrolledIn}(\texttt{Pierre}, \texttt{CSDept}) \\ \texttt{RegisteredTo}(\texttt{Pierre}, \texttt{UE111}) \\ \texttt{OfferedBy}(\texttt{UE111}, \texttt{CSDept})\end{array}$$

RDFS: RDF Schema

- Not detailed here: the schema in RDF is super simplistic
- An RDF Schema defines the schema of a richer ontology

RDF Schema

- Do net get confused: RDFS can use RDF as syntax, i.e., RDFS statements can be themselves expressed as RDF triplets using some specific properties and objects used as RDFS keywords with a particular meaning.

- Declaration of classes and subclass relationships
  - ⟨ `Staff` `rdf:type` `rdfs:Class` ⟩
  - ⟨ `Java` `rdfs:subClassOf` `CSCourse` ⟩
- Declaration of instances (beyond the pure schema)
  - ⟨ `Dupond` `rdf:type` `AcademicStaff` ⟩

RDF Schema - continued

- Declaration of relations (properties in RDFS terminology)
  - ⟨ RegisteredTo rdf:type rdf:Property ⟩
- Declaration of subproperty relationships
  - ⟨ LateRegisteredTo rdfs:subPropertyOf RegisteredTo ⟩
- Declaration of domain and range restrictions for predicates
  - ⟨ TeachesIn rdfs:domain AcademicStaff ⟩
  - ⟨ TeachesIn rdfs:range Course ⟩
  - TeachesIn(AcademicStaff, Course)

RDFS logical semantics

RDF and RDFS statements	**FOL translation**	**DL notation**
⟨ i rdf:type C ⟩	$C(i)$	$i : C$ or $C(i)$
⟨ i P j ⟩	$P(i,j)$	$i\,P\,j$ or $P(i,j)$
⟨ C rdfs:subClassOf D ⟩	$\forall X\,(C(X) \Rightarrow D(X))$	$C \sqsubseteq D$
⟨ P rdfs:subPropertyOf R ⟩	$\forall X \forall Y\,(P(X,Y) \Rightarrow R(X,Y))$	$P \sqsubseteq R$
⟨ P rdfs:domain C ⟩	$\forall X \forall Y\,(P(X,Y) \Rightarrow C(X))$	$\exists P \sqsubseteq C$
⟨ P rdfs:range D ⟩	$\forall X \forall Y\,(P(X,Y) \Rightarrow D(Y))$	$\exists P^{-} \sqsubseteq D$

- Ignore for now DL column
- This is just a notation
- We will come back to it to discuss Description logics

OWL: Web Ontology Language

- OWL extends RDFS with the possibility to express additional constraints
- Main OWL constructs
  - Disjointness between classes
  - Constraints of functionality and symmetry on predicates
  - Intentional class definitions
  - Class union and intersection
- We will see these are all expressible in Description logics

OWL constructs

- Ignore again the DL column
- Disjointness between classes:

OWL notation	FOL translation	DL notation
⟨ C owl:disjointWith D ⟩	$\forall X (C(X) \Rightarrow \neg D(X))$	$C \sqsubseteq \neg D$

- Constraints of functionality and symmetry on predicates:

OWL notation	FOL translation	DL notation
⟨ *P* rdf:type owl:FunctionalProperty ⟩	$\forall X \forall Y \forall Z$ $(P(X,Y) \wedge P(X,Z) \Rightarrow Y = Z)$	$(\textit{funct}\, P)$ or $\exists P \sqsubseteq (\leq 1\, P)$
⟨ *P* rdf:type owl:InverseFunctionalProperty ⟩	$\forall X \forall Y \forall Z$ $(P(X,Y) \wedge P(Z,Y) \Rightarrow X = Z)$	$(\textit{funct}\, P^{-})$ or $\exists P^{-} \sqsubseteq (\leq 1\, P^{-})$
⟨ *P* owl:inverseOf *Q* ⟩	$\forall X \forall Y (P(X,Y) \Leftrightarrow Q(Y,X))$	$P \equiv Q^{-}$
⟨ *P* rdf:type owl:SymmetricProperty ⟩	$\forall X \forall Y (P(X,Y) \Rightarrow P(Y,X))$	$P \sqsubseteq P^{-}$

Definition of intentional classes in OWL

- Goal: allow expressing complex constraints such as:
  - departments can be lead only by professors
  - only professors or lecturers may teach to undergraduate students.
- The keyword `owl:Restriction` is used in association with a blank node class, and some specific restriction properties:
  - `owl:someValuesFrom`
  - `owl:allValuesFrom`
  - `owl:minCardinality`
  - `owl:maxCardinality`

OWL Semantics

OWL notation	FOL translation	DL notation
_a owl:onProperty P _a owl:allValuesFrom C	$\forall Y \, (P(X,Y) \Rightarrow C(Y))$	$\forall P.C$
_a owl:onProperty P _a owl:someValuesFrom C	$\exists Y \, (P(X,Y) \wedge C(Y))$	$\exists P.C$
_a owl:onProperty P _a owl:minCardinality *n*	$\exists Y_1 \ldots \exists Y_n (P(X,Y_1) \wedge \ldots \wedge P(X,Y_n) \wedge \bigwedge_{i,j \in [1..n], i \neq j} (Y_i \neq Y_j))$	$(\geq n P)$
_a owl:maxCardinality *n*	$\forall Y_1 \ldots \forall Y_n \forall Y_{n+1}$ $(P(X,Y_1) \wedge \ldots \wedge P(X,Y_n) \wedge P(X,Y_{n+1})$ $\Rightarrow \bigvee_{i,j \in [1..n+1], i \neq j} (Y_i = Y_j))$	$(\leq n P)$

Unnamed new classes by example

- Departments can be lead only by professors

- Define the set of objects that are lead by professors

```
_a  rdfs:subClassOf    owl:Restriction
_a  owl:onProperty     Leads
_a  owl:allValuesFrom  Professor
```

- Now specify that all departments are lead by professors

```
Department  rdfs:subClassOf  _a
```

Union and Intersection of Classes by example

- only professors or lecturers may teach to undergraduate students

```
_a  rdfs:subClassOf      owl:Restriction
_a  owl:onProperty       TeachesTo
_a  owl:someValuesFrom   Undergrad
_b  owl:unionOf          (Professor, Lecturer)
_a  rdfs:subClassOf      _b
```

- This corresponds to an inclusion axiom in Description Logic:

$$\exists\ \mathit{TeachesTo.UndergraduateStudent} \sqsubseteq \mathit{Professor} \sqcup \mathit{Lecturer}$$

- `owl:equivalentClass` corresponds to double inclusion:

$$\mathit{MathStudent} \equiv \mathit{Student} \sqcap \exists\ \mathit{RegisteredTo.MathCourse}$$

Outline

1. Introduction
2. 3 ontology languages for the Web
3. Reasoning in Description Logics
   - $\mathcal{ALC}$
   - Polynomial DLs
4. Querying Data through Ontologies
5. Conclusion

Description Logics

- Philosophy: isolate decidable fragments of first-order logic allowing reasoning on complex logical axioms over unary and binary predicates
- These fragments are called Description Logics

- The DL jargon:
  - the classes are called concepts
  - the properties are called roles.
  - the ontology (the knowledge base) = Tbox + Abox
  - the schema is called the Tbox
  - the instance is called the Abox

The DL family

- Few constructs: atomic concepts and roles, inverse of roles, unqualified restriction on roles, restricted negation
- Revisit RDFS checking out the DL column
- If you don't like the syntax: neither do I

Semantics of main conctructs

- $I(C_1 \sqcap C_2) = I(C_1) \cap I(C_2)$
- $I(\forall R.C) = \{o_1 \mid \forall\ o_2\ [(o_1, o_2) \in I(R) \Rightarrow o_2 \in I(C)]\}$
- $I((\exists R.C) = \{o_1 \mid \exists o_2.[(o_1, o_2) \in I(R)\ \wedge\ o_2 \in I(C)]\}$
- $I(\neg C) = \Delta^I \setminus I(C)$
- $I(R^-) = \{(o_2, o_1) \mid (o_1, o_2) \in I(R)\}$

Defining a particular description logic

- Define how to construct complex concepts and roles starting from atomic concepts and roles
  - $\textit{Professor} \sqcup \textit{Lecturer}$ (those who are either professor or lecturer)
- Choose the constraints you want to consider
- The complexity of the logic depends on these choices

Reasoning problems studied in DL

- Satisfiability checking: Given a DL knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, is $\mathcal{K}$ satisfiable?
- Subsumption checking: Given a Tbox $\mathcal{T}$ and two concept expressions C and D, does $\mathcal{T} \models C \sqsubseteq D$?
- Instance checking: Given a DL knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, an individual e and a concept expression C, does $\mathcal{K} \models C(e)$?
- Query answering: Given a DL knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, and a concept expression C, finds the set of individuals e such that $\mathcal{K} \models C(e)$?

Remarks

- For DLs with full negation: instance checking and subsumption checking can be reduced to (un)satisfiability checking
  - $\mathcal{T} \models C \sqsubseteq D \Leftrightarrow \langle \mathcal{T}, \{(C \sqcap \neg D)(a)\}\rangle$ is unsatisfiable.
  - $\langle \mathcal{T}, \mathcal{A}\rangle \models C(e) \Leftrightarrow \langle \mathcal{T}, \mathcal{A} \cup \{\neg C(e)\}\rangle$ is unsatisfiable.
- For DLs without negation: instance checking can be reduced to subsumption checking by computing the most specific concept satisfied by an individual in the Abox (denoted $msc(\mathcal{A}, e)$)
  - $\langle \mathcal{T}, \mathcal{A}\rangle \models C(e) \Leftrightarrow \mathcal{T} \models msc(\mathcal{A}, e) \sqsubseteq C$

$\mathcal{ALC}$: the prototypical DL

- (standard) An $\mathcal{ALC}$ Abox is made of a set of facts of the form $C(a)$ and $R(a, b)$ where a and b are individuals, R is an atomic role and C is a possibly complex concept
- $\mathcal{ALC}$ constructs:
  - conjunction $C_1 \sqcap C_2$,
  - existential restriction $\exists R.C$
    - $\exists Y\,(R(X, Y) \wedge C(Y))$
  - negation $\neg C$.
- As a result, $\mathcal{ALC}$ also contains de facto:
  - disjunctions $C_1 \sqcup C_2$ ($\equiv \neg(\neg C_1 \sqcap \neg C_2)$),
  - value restrictions ($\forall R.C \equiv \neg(\exists R.\neg C)$),
  - $\top$ ($\equiv A \sqcup \neg A$) and $\bot$ ($\equiv A \sqcap \neg A$).

$\mathcal{ALC}$ - continued

- An $\mathcal{ALC}$ Tbox may contain inclusion constraints between concepts and roles

$$MathCourse \sqsubseteq Course$$
$$LateRegisteredTo \sqsubseteq RegisteredTo$$

- An $\mathcal{ALC}$ Tbox may contain General Concept Inclusions (GCIs): $\exists TeachesTo.UndergraduateStudent \sqsubseteq Professor \sqcup Lecturer$

Tableau method

- Reasoning is based on tableau calculus - a classical method in logic for checking satisfiability
- Extensively used in Description logics for implementing reasoners
- Technique
  - Get rid of the Tbox by recursively unfolding the concept definitions
  - Transform the resulting Abox so that negations applies only to atomic concepts
  - Try to construct a model or raise a contradiction
- We illustrate the technique with a simple example without GCIs
- In general, much more involved

Tableau method

- For satisfiability checking of a DL knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$
  - $\mathcal{T} = \{C_1 \equiv A \sqcap B,\ C_2 \equiv \exists R.A,\ C_3 \equiv \forall R.B,\ C_4 \equiv \forall R.\neg C_1\}$
  - $\mathcal{A} = \{C_2(a), C_3(a), C_4(a)\}$
- Get rid of the Tbox, by recursively unfolding the concept definitions:
  - $\mathcal{A}' = \{(\exists R.A)(a), (\forall R.B)(a), (\forall R.\neg(A \sqcap B))(a)\} \equiv \langle \mathcal{T}, \mathcal{A} \rangle$
- Transform the concepts expressions in $\mathcal{A}'$ into negation normal form
  - $\mathcal{A}'' = \{(\exists R.A)(a), (\forall R.B)(a), (\forall R.(\neg A \sqcup \neg B))(a)\}$
- Apply tableau rules to extend the resulting Abox until no rule applies anymore:
  - From an extended Abox which is complete (no rule applies) and clash-free (no obvious contradiction), a so-called canonical interpretation can be built, which is a model of the initial Abox.

Tableau rules for $\mathcal{ALC}$

- The $\sqcap$-rule:
  Condition: $\mathcal{A}$ contains $(C \sqcap D)(a)$ but not both $C(a)$ and $D(a)$
  Action: add $\mathcal{A}' = \mathcal{A} \cup \{C(a), D(a)\}$
- The $\sqcup$-rule:
  Condition: $\mathcal{A}$ contains $(C \sqcup D)(a)$ but neither $C(a)$ nor $D(a)$
  Action: add $\mathcal{A}' = \mathcal{A} \cup \{C(a)\}$ and $\mathcal{A}'' = \mathcal{A} \cup \{D(a)\}$
- The $\exists$-rule:
  Condition: $\mathcal{A}$ contains $(\exists R.C)(a)$ but there is no c such that $\{R(a,c), C(c)\} \subseteq \mathcal{A}$
  Action: add $\mathcal{A}' = \mathcal{A} \cup \{R(a,b), C(b)\}$ where b is a new individual name
- The $\forall$-rule:
  Condition: $\mathcal{A}$ contains $(\forall R.C)(a)$ and $R(a,b)$ but not $C(b)$
  Action: add $\mathcal{A}' = \mathcal{A} \cup \{C(b)\}$

Illustration on the example

- The result of the application of the tableau method to $\mathcal{A}'' = \{(\exists R.A)(a), (\forall R.B)(a), (\forall R.(\neg A \sqcup \neg B))(a)\}$ gives the following Aboxes:
  - $\mathcal{A}''_1 = \mathcal{A}'' \cup \{R(a,b), A(b), B(b), \neg A(b)\}$
  - $\mathcal{A}''_2 = \mathcal{A}'' \cup \{R(a,b), A(b), B(b), \neg B(b)\}$

- They both contain a clash:
  $\mathcal{A}''$ (and the equivalent original knowledge base) is correctly decided unsatisfiable by the algorithm

Complexity

- The tableau method shows that the satisfiability of $\mathcal{ALC}$ knowledge bases is decidable but with a complexity that may be exponential because of the disjunction construct and the associated $\sqcup$-*rule*.
- Satisfiability checking in $\mathcal{ALC}$ (and thus also subsumption and instance checking) is in fact EXPTIME-complete
- Additional constructs like those in the fragment OWL DL of OWL do not change the complexity class of reasoning (which remains EXPTIME-complete)
- OWL Full is undecidable

DLs for which reasoning is polynomial

- $\mathcal{FL}$: conjunction $C_1 \sqcap C_2$, value restrictions $\forall R.C$ and unqualified existential restriction $\exists R$
  - For Tboxes without GCIs, subsumption checking is polynomial
  - For Tboxes with (even simple) GCIs,subsumption checking is co-NP complete
- $\mathcal{EL}$: conjunctions $C_1 \sqcap C_2$ and existential restrictions $\exists R.C$
  - Subsumption checking in $\mathcal{EL}$ is polynomial even for general Tboxes.
- $\mathcal{FLE}$: conjunction $C_1 \sqcap C_2$, value restrictions $\forall R.C$, and existential restrictions $\exists R.C$
  - Subsumption checking in $\mathcal{FLE}$ is NP-complete
- The DL-LITE family: a good trade-off, specially designed for guaranteeing query answering through ontologies to be polynomial in data complexity.

Outline

1 Introduction

2 3 ontology languages for the Web

3 Reasoning in Description Logics

4 Querying Data through Ontologies
- Querying using RDFS
- Querying using DL-LITE
- Complexity

5 Conclusion

Querying using RDFS

- RDFS statements can be used to infer new triples
- Example
  - Base fact $\mathit{ResponsibleOf}(\mathit{durand}, \mathit{ue111})$
  - Use the statement $\langle \mathit{ResponsibleOf}$ `rdfs:domain` $\mathit{Professor} \rangle$
    i.e., the logical rule: $\mathit{ResponsibleOf}(X, Y) \Rightarrow \mathit{Professor}(X)$
  - With substitution {X/durand, Y/ue111}
  - Infer fact $\mathit{Professor}(\mathit{durand})$
  - Use the statement $\langle \mathit{Professor}$ `rdfs:subClassOf` $\mathit{AcademicStaff} \rangle$
    i.e., the rule $\mathit{Professor}(X) \Rightarrow \mathit{AcademicStaff}(X)$
  - With substitution {X/durand}
  - Infer fact $\mathit{AcademicStaff}(\mathit{durand})$
  - etc.

The saturation algorithm

- Keep infering new facts until a fixpoint is reached
- Note: Only polynomially many facts can be added
- PTIME

Querying using DL-LITE

- Develop as a good compromise between expressive power and reasonable complexity of query answering
- RDFS simpler and very used but limited
- More complex DL: query answering is unfeasible

The DL-LITE family

- Three kinds of axioms: positive inclusions (PI), negative inclusions (NI) and functionality constraints (func)
- Captures the main constraints used in Databases and Software Engineering
- Different variants
  - DL-LITE$_{\mathcal{R}}$: no functionality constraints
  - DL-LITE$_{\mathcal{F}}$: no role inclusion
  - DL-LITE$_{\mathcal{A}}$: no functionality constraints on roles involved in role inclusions

PI: Positive inclusion and incompleteness

- One of the following forms:

DL notation	**Corresponding logical rule**
$B \sqsubseteq \exists P$	$B(X) \Rightarrow \exists Y P(X, Y)$
$\exists Q \sqsubseteq \exists P$	$Q(X, Y) \Rightarrow \exists Z P(X, Z)$
$B \sqsubseteq \exists P^-$ $\exists Q \sqsubseteq \exists P^-$	$B(X) \Rightarrow \exists Y P(Y, X)$ $Q(X, Y) \Rightarrow \exists Z P(Z, X)$
$P \sqsubseteq Q^-$ or $P^- \sqsubseteq Q$	$P(X, Y) \Rightarrow Q$(Y,X)

where P and Q denote properties and B denotes a class.

DL notation	**Corresponding logical rule**
$Professor \sqsubseteq \exists TeachesIn$	$Professor(X) \Rightarrow \exists Y\, TeachesIn(X, Y)$
$Course \sqsubseteq \exists RegisteredIn^-$	$Course(X) \Rightarrow \exists Y\, RegisteredIn(Y, X)$

- Not safe
- From $Professor(durand)$, I know there is some y $TeachesIn(durand, y)$
- Incompleteness: I don't know y
- Saturation may not terminate

Negative inclusion and inconsistencies

- Negative inclusion takes one of the forms:

DL notation
$B_1 \sqsubseteq \neg B_2$
$R_1 \sqsubseteq \neg R_2$

  - where B_1 and B_2 are either classes or expressions of the form $\exists P$ or $\exists P^-$ for some property P
  - and where R_1 and R_2 are either properties or inverses of properties.

- Students do not teach courses

DL notation	**Corresponding logical rule**
$Student \sqsubseteq \neg\exists TeachesIn$	$Student(X) \Rightarrow \neg\exists Y\, TeachesIn(X, Y)$ or equivalently, $\exists Y\, TeachesIn(X, Y) \Rightarrow \neg Student(X)$

- The knowledge base may be inconsistent
- Not possible with RDFS ontologies

Key constraints and more inconsistencies.

- Axioms of the form $(funct\ P)$ or $(funct\ P^{-})$ where P is a property

DL notation	corresponding logical rule
$(funct\ P)$	$P(X,Y) \wedge P(X,Z) \Rightarrow Y = Z$
$(funct\ P^{-})$	$P(Y,X) \wedge P(Z,X) \Rightarrow Y = Z$

- Key constraints also lead to inconsistencies
- Example:
  - $(funct\ ResponsibleOf^{-})$
  - A course must have a unique professor responsible for it
  - If we have $ResponsibleOf(durand, ue111)$ and $ResponsibleOf(dupond, ue111)$
    The KB is inconsistent

Query answering: Example

- Abox:
  - $Professor(Jim)$, $HasTutor(John, Mary)$, $TeachesTo(John, Bill)$
- Tbox:
  - $Professor \sqsubseteq \exists TeachesTo$
  - $Student \sqsubseteq \exists HasTutor$
  - $\exists TeachesTo^{-} \sqsubseteq Student$
  - $\exists HasTutor^{-} \sqsubseteq Professor$
  - $Professor \sqsubseteq \neg Student$
- Queries: conjunctive queries on concepts and atomic roles
  - $q_0(x) \leftarrow TeachesTo(x, y) \wedge HasTutor(y, z)$

Query answering: Principles of reformulation

- Transform the query into FO queries over the database

- FO queries are used to check for inconsistencies of the KB
- FO queries are used to evaluate the result

- The FO queries can be evaluated using a database engine with query optimization

- Because of incompleteness, not always possible

Query answering by example (no inconsistency)

- Tbox: $\mathcal{T}$
  - $Professor \sqsubseteq \exists TeachesTo$
  - $Student \sqsubseteq \exists HasTutor$
  - $\exists TeachesTo^{-} \sqsubseteq Student$
  - $\exists HasTutor^{-} \sqsubseteq Professor$
  - $Professor \sqsubseteq \neg Student$
- Query:
  - $q_0(x) \leftarrow TeachesTo(x, y) \wedge HasTutor(y, z)$
- Reformulations of q_0 given the the Tbox $\mathcal{T}$:
  - $q_1(x) \leftarrow TeachesTo(x, y) \wedge Student(y)$
  - $q_2(x) \leftarrow TeachesTo(x, y) \wedge TeachesTo(z', y)$
  - $q_3(x) \leftarrow TeachesTo(x, y')$
  - $q_4(x) \leftarrow Professor(x)$
  - $q_5(x) \leftarrow HasTutor(u, x)$
- Main result (holds for DL-LITE$_{\mathcal{A}}$ but not for full DL-LITE):
  - For any Abox $\mathcal{A}$ such that $\mathcal{T} \cup \mathcal{A}$ is satisfiable:
    Answer(q_0, $\mathcal{T} \cup \mathcal{A}$) = $\bigcup_i$ Answer(q_i, $\mathcal{A}$)

Illustration

- $q_0(x) \leftarrow \mathit{TeachesTo}(x, y) \wedge \mathit{HasTutor}(y, z)$
- $\mathit{Student} \sqsubseteq \exists \mathit{HasTutor}$
- $\mathit{HasTutor}(y, z) \leftarrow \mathit{Student}(y)$
- $q_1(x) \leftarrow \mathit{TeachesTo}(x, y) \wedge \mathit{Student}(y)$

Example (ctd)

- Abox: $\mathcal{A}$
  - $Professor(Jim)$, $HasTutor(John, Mary)$, $TeachesTo(John, Bill)$
- Query
  - $q_0(x) \leftarrow TeachesTo(x, y) \wedge HasTutor(y, z)$
- Reformulations of q_0 given the the Tbox $\mathcal{T}$:
  - $q_1(x) \leftarrow TeachesTo(x, y) \wedge Student(y)$
  - $q_2(x) \leftarrow TeachesTo(x, y) \wedge TeachesTo(z', y)$
  - $q_3(x) \leftarrow TeachesTo(x, y')$
  - $q_4(x) \leftarrow Professor(x)$
  - $q_5(x) \leftarrow HasTutor(u, x)$
- Result of the evaluation of the reformulations over $\mathcal{A}$:
  - Answer(q_0, $\mathcal{T} \cup \mathcal{A}$) = $\{Mary, Jim, John\}$

Consistency checking by example

- Tbox: $\mathcal{T}'$
  - $Professor \sqsubseteq \exists TeachesTo$
  - $Student \sqsubseteq \exists HasTutor$
  - $\exists TeachesTo^{-} \sqsubseteq Student$
  - $\exists HasTutor^{-} \sqsubseteq Professor$
  - $Professor \sqsubseteq \neg Student$
  - $\exists TeachesTo \sqsubseteq \neg Student$
  - $\exists HasTutor \sqsubseteq Student$
- Saturation of the NIs (possibly using the PIs):
  - $\exists TeachesTo \sqsubseteq \neg \exists HasTutor$
- Translation of each NI into a boolean conjunctive query:
  - $q_{unsat} \leftarrow TeachesTo(x, y) \wedge HasTutor(x, y')$
- Evaluation of q_{unsat} on the Abox $\mathcal{A}$:
  - $\{Professor(Jim), HasTutor(John, Mary), TeachesTo(John, Bill)\}$
  - Answer(q_{unsat}, $\mathcal{A}$) = true
- Main result:
  - $\mathcal{T}' \cup \mathcal{A}$ is inconsistent iff there exists a q_{unsat} such that Answer($q_{unsat}, \mathcal{A}$) = true

- Closure of a Tbox: derive new statements
- From $\exists \mathit{TeachesTo} \sqsubseteq \neg \mathit{Student}$
- Derive $\mathit{Student} \sqsubseteq \neg \exists \mathit{TeachesTo}$
- From $\exists \mathit{HasTutor} \sqsubseteq \mathit{Student}$ and $\mathit{Student} \sqsubseteq \neg \exists \mathit{TeachesTo}$
- Derive $\exists \mathit{HasTutor} \sqsubseteq \neg \exists \mathit{TeachesTo}$
- From $\exists \mathit{HasTutor} \sqsubseteq \neg \exists \mathit{TeachesTo}$
- Derive $\exists \mathit{TeachesTo} \sqsubseteq \neg \exists \mathit{HasTutor}$

FOL reducibility of data management in DL-LITE

Query answering and data consistency checking can be performed in two separate steps:

1. A reasoning step with the Tbox alone (i.e., the ontology without the data) and some conjunctive queries
2. An evaluation step of conjunctive queries over the data in the Abox (without the Tbox)
   - makes it possible to use an SQL engine
   - thus taking advantage of well-established query optimization strategies supported by standard relational DBMS

Complexity results

- The reasoning step on Tbox is polynomial in the size of the Tbox
  - Produces a polynomial number of reformulations and of *unsat* queries
- The evaluation step over the Abox has the same data complexity as standard evaluation of conjunctive queries over relational databases
  - in AC_0 (strictly contained in *LogSpace* and thus in *P*)
- The interaction between role inclusion constraints and functionality constraints makes reasoning in DL-LITE *P*-complete in data complexity
  - full DL-LITE is not FOL-reducible
  - Reformulating a query may require recursion

Problem with full DL-LITE by example

- Let the Tbox (R and P are two properties and S is a class):
  $R \sqsubseteq P$
  $(funct\ P)$
  $S \sqsubseteq \exists R$
  $\exists R^{-} \sqsubseteq \exists R$
- and the query: $q(x)$:- $R(z, x)$
- $r_1(x)$:- $S(x_1), P(x_1, x)$ is a reformulation of the query q given the Tbox
  - from $S(x_1)$ and the PI $S \sqsubseteq \exists R$, it can be inferred: $\exists y\ R(x_1, y)$, and thus $\exists y\ P(x_1, y)$ (since R $\sqsubseteq$ P).
  - from the functionality constraint on P and $P(x_1, x)$, it can be inferred: $y = x$, and thus: $R(x_1, x)$
  - Therefore: $\exists x_1 S(x_1) \wedge P(x_1, x) \models \exists z R(z, x)$ (i.e., $r_1(x)$ is contained in the query $q(x)$)

Problem with full DL-LITE by example - continued

- r_1 is not the only one reformulation of the query
- In fact, there exists an *infinite* number of different reformulations for $q(x)$:
- for $k \geq 2$, $r_k(x)$:- $S(x_k), P(x_k, x_{k-1}), \ldots, P(x_1, x)$ is also a reformulation:
  - from $S(x_k)$ and the PI $S \sqsubseteq \exists R$, it can be inferred: $\exists y_k\ R(x_k, y_k)$, and thus $\exists y_k\ P(x_k, y_k)$ (since $\mathrm{R} \sqsubseteq \mathrm{P}$).
  - from the functionality constraint on P and $P(x_k, x_{k-1})$, it can be inferred: $y_k = x_{k-1}$, and thus: $R(x_k, x_{k-1})$
  - Now, based on the PI $\exists \mathrm{R}^- \sqsubseteq \exists \mathrm{R}$: $\exists y_{k-1}\ R(x_{k-1}, y_{k-1})$,
  - and with the same reasoning as before, we get $y_{k-1} = x_{k-2}$, and thus: $R(x_{k-1}, x_{k-2})$.
  - By induction, it can be inferred: $R(x_1, x)$, and therefore $r_k(x)$ is contained in the query $q(x)$.

Problem with full DL-LITE by example - end

- One can show that for each k, there exists an Abox such that the reformulation r_k returns answers that are not returned by the reformulation $r_{k'}$ for $k' < k$.
- Thus, there exists an infinite number of *non redundant* conjunctive reformulations.

Outline

1. Introduction
2. 3 ontology languages for the Web
3. Reasoning in Description Logics
4. Querying Data through Ontologies
5. Conclusion

Conclusion

- The scalability of reasoning on Web data requires light-weight ontologies
- One can use a description logic for which reasoning is feasible (polynomial)
- For Aboxes stored as relational databases, it is even preferable that query answering can be performed with a relational query (using query reformulation)
- Full OWL is too complex
- Consider extensions of RDFS