XPTO Prolog Treatment for Ontologies

Contextual Logic Programming for Ontology Representation and Querying

Nuno Lopes Cláudio Fernandes Salvador Abreu

Universidade de Évora

June 30, 2007

Presentation and Motivation

- Semantic Web Ontologies
- CxLP

Presentation and Motivation

- Semantic Web Ontologies
- CxLP
  - Ontology representation
  - Integration with ISCO and other data sources

Presentation and Motivation

- Semantic Web Ontologies
- CxLP
  - Ontology representation
  - Integration with ISCO and other data sources
- Query the representation internally

Presentation and Motivation

- Semantic Web Ontologies
- CxLP
  - Ontology representation
  - Integration with ISCO and other data sources
- Query the representation internally

Based on the representation:

- Enable being queried using SPARQL

Presentation and Motivation

- Semantic Web Ontologies
- CxLP
  - Ontology representation
  - Integration with ISCO and other data sources
- Query the representation internally

Based on the representation:

- Enable being queried using SPARQL
- Be able to query SPARQL web services

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).

item(A).
```

```
:- unit(bar(A)).

item(A).
item(A) :- :^ item(A).
```

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).

item(A).
```

```
:- unit(bar(A)).

item(A).
item(A) :- :^ item(A).
```

- Contexts:

```
foo(b) :> item(X).
```

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).

item(A).
```

```
:- unit(bar(A)).

item(A).
item(A) :- :^ item(A).
```

- Contexts:

```
foo(b) :> item(X).          X = b
```

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).

item(A).
```

```
:- unit(bar(A)).

item(A).
item(A) :- :^ item(A).
```

- Contexts:

```
foo(b) :> item(X).                X = b

foo(1) :> bar(a) :> item(X).
```

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).

item(A).
```

```
:- unit(bar(A)).

item(A).
item(A) :- :^ item(A).
```

- Contexts:

```
foo(b) :> item(X).                    X = b

foo(1) :> bar(a) :> item(X).   X = a
```

GNU Prolog/CX

- Units:

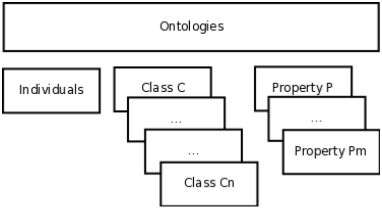
```
:- unit(foo(A)).

item(A).
```

```
:- unit(bar(A)).

item(A).
item(A) :- :^ item(A).
```

- Contexts:

```
foo(b) :> item(X).                  X = b

foo(1) :> bar(a) :> item(X).        X = a ;
                                    X = 1
```

Representation of the ontology



Ontologies are represented using units:

- one unit that lists the classes and properties of the ontology;
- another unit for individuals;
- one for each OWL class
- one for each property

Ontology Unit

This unit represents the ontology information:

- XML namespaces
- headers
- classes
- properties

Individuals Unit

Individuals are stored along with their class

```
individual_class(CLASS, INDIVIDUAL).
```

Individuals Unit

Individuals are stored along with their class
`individual_class(CLASS, INDIVIDUAL).`

Properties of each individual are stored as triples in the predicate `property/3`.
`property(INDIVIDUAL, PROPERTY, VALUE).`

Individuals Unit

Individuals are stored along with their class
`individual_class(CLASS, INDIVIDUAL).`

Properties of each individual are stored as triples in the predicate `property/3`.
`property(INDIVIDUAL, PROPERTY, VALUE).`

Individual relations:

- `differentFrom(IND1, IND2).`
- `sameAs(IND1, IND2).`

Class Units

- Each unit represents a class of the ontology
- Stores as facts the information about the class
  - restrictions on the individual properties
  - class inheritance
- some predicates that help querying the representation:
  - class_name(NAME)
  - superClassOf(CLASS)

Property Units

Each property unit contains the information relative to a specific property.

- type of the property (datatype or object)
- domain and range
- property inheritance and property relations.

Property Units

Each property unit contains the information relative to a specific property.

- type of the property (datatype or object)
- domain and range
- property inheritance and property relations.

These units also define the predicate to access its value, given the individual name.

```
item(B) :-
 :^ item(B),
 property(B, hasMaker, A).
```

Querying the representation

- The most direct way of retrieving the class individuals is to use the goal `item/1`
- The `item/1` goal binds, by backtrack, its argument to each individual of the class.
- There is also the possibility of querying all the individuals in the ontology by omitting a class in the query.

```
| ?- 'ClassName' /> item(A).
A = 'IndividualName'
```

- The value of the properties can be accessed by including the unit that represents the property in the context query.
- The argument of the property unit will be bound to the value of the property for the corresponding individual.

```
| ?- 'IceWine' /> hasFlavor(F) :> hasBody(B) :>
  item(I).
B = 'Medium'
F = 'Moderate'
I = 'SelaksIceWine' ?
```

Other query forms

- **individual/1** unifies its argument with the name of the individual (same as `item/1`)
- **class/1** unifies its argument with the class of the individual.
- **property/2** allows to query for the property name based on the property value.
- **optional/1** receives as its argument a another defined unit and will succeed with the results if the unit specified in its argument succeeds. Otherwise it will succeed leaving any variables in its argument unbound.

Answering SPARQL queries

Answering SPARQL queries

- SPARQL parser written using Flex and Bison

Answering SPARQL queries

- SPARQL parser written using Flex and Bison
- Generates a context that represents the query

Answering SPARQL queries

- SPARQL parser written using Flex and Bison
- Generates a context that represents the query
- Context is triggered to obtain the query results

Answering SPARQL queries

- SPARQL parser written using Flex and Bison
- Generates a context that represents the query
- Context is triggered to obtain the query results
- And formatted according to the XML specefications

```
SELECT
       ?flavor ?body
WHERE {
   ?t :hasFlavor    ?flavor   .
   ?t :hasBody      ?body   .
}
```

```
[where([set([
     triple(A,hasFlavor,B),
     triple(A,hasBody,C) ])
       ]),
 select([flavor=B,body=C]),
 vars([flavor=B,body=C,t=A]),
]
```

SPARQL engine

- Each SPARQL functionality is implemented as a unit
- The `triple/3` unit is responsible for instantiating the variables in the query by accessing the representation of the ontology.

SPARQL engine

- Each SPARQL functionality is implemented as a unit
- The `triple/3` unit is responsible for instantiating the variables in the query by accessing the representation of the ontology.

```
/> property(hasFlavor,F) :> item(I).
```

Mapping Prolog/CX queries to SPARQL

- Merge the reasoning of the system internal knowledge base with external ontologies available from third parties by means of the SPARQL query language:

Mapping Prolog/CX queries to SPARQL

- Merge the reasoning of the system internal knowledge base with external ontologies available from third parties by means of the SPARQL query language:
  - Translates a Prolog/CX query into SPARQL;
  - Sends the SPARQL query to the indicated Semantic Web SPARQL service;
  - Fetch the XML result file, parse it and return the solutions as Prolog variable bindings.

Formal Query Form

```
QUERY := sparql(IRI) /> P1 ... Pn :> ITEM
 URI  :=  url
 P    := property(VALUE) || where(PROP, VALUE)
 ITEM := item(INDIVIDUAL)
```

Formal Query Form

```
QUERY := sparql(IRI) /> P1 ... Pn :> ITEM
 URI  :=  url
 P    := property(VALUE) || where(PROP, VALUE)
 ITEM := item(INDIVIDUAL)
```

```
?- sparql('http://xmlarmyknife.org/api/rdf/sparql/') />
   hasBody(A) :> hasColor(B) :> item(IND).

A ='http://www.w3.org/2001/sw/WebOnt/wine#Medium'
B ='http://www.w3.org/2001/sw/WebOnt/wine#SelaksIceWine'
IND ='http://www.w3.org/2001/sw/WebOnt/wine#White' ? ;

```

Conclusions

- Representation of the ontology
- Integrates well with databases
- SPARQL enabled query engine
- Capable of querying SPARQL web services

Conclusions

- Representation of the ontology
- Integrates well with databases
- SPARQL enabled query engine
- Capable of querying SPARQL web services

Future work:

- Allow multiple ontologies to be loaded
- Semantics of OWL
- Complete the SPARQL support in answering queries
- Complete the external query SPARQL generation

Questions?