Introduction to Contextual Logic Programming

Nuno Lopes

February 4, 2008

Modularity in Prolog

Module systems:

- re-use of code
- development of libraries

Modularity in Prolog

Module systems:

- re-use of code
- development of libraries

No unique module system for Prolog:

- SICStus Prolog
- SWI Prolog
- Logtalk (OOP)
- CIAO Prolog
- XSB
- ...

Contextual Logic Programming (CxLP)

- Contextual Logic Programming, L. Monteiro and A. Porto (1989)
  - standard predicates and goals
  - modules (a.k.a. units)
  - contexts (sequence of units)
  - “extended” goal derivation system

Units and contexts

Unit Set of clauses associated by a name

Context Sequence of units

- calling context
- execution context

Units and contexts

Unit Set of clauses associated by a name

Context Sequence of units

- calling context
- execution context

Context resolution Executing a goal G in a context C (calling context)

- Locate the first unit u, in C, that contains a definition of G
- Execute G's body, in the context C' (remainder of C that starts with unit u: execution context)

CxLP vs OOP

Context and object instance: A context is a list of units which can be described as an object *instance*

Predicate and method: A predicate present in a unit is equivalent to a method definition in OO

Goal and message: a goal executed in a context can be interpreted as sending a message to an object

GNU Prolog/CX

GNU Prolog/CX

- implementation of CxLP

GNU Prolog/CX

GNU Prolog/CX

- implementation of CxLP
- introduces unit arguments to CxLP:

GNU Prolog/CX

GNU Prolog/CX

- implementation of CxLP
- introduces unit arguments to CxLP:
  - act as a “unit global” variable
  - allow for contexts and units to be parametrized
  - similar to instance variables in OOP, variables whose scope is the entire unit

Context Operators

Context manipulation:

- U :> G   Context extension: extends the current context with U and evaluates G
- C :< G   Context switch: evaluates G in the context C
- :^ G   Supercontext: evaluates G in the *parent* context

Context Operators

Context manipulation:

U :> G Context extension: extends the current context with U and evaluates G

C :< G Context switch: evaluates G in the context C

:^ G Supercontext: evaluates G in the *parent* context

Context query:

:< C Current context: unifies C with the current context

:> C Calling context: unifies C with the calling context

GNU Prolog/CX

- Units:

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

item(A).
```

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

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

GNU Prolog/CX

- Units:

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

item(A).
```

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

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

- Contexts:

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

GNU Prolog/CX

- Units:

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

item(A).
```

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

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

- Contexts:

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

GNU Prolog/CX

- Units:

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

item(A).
```

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

item(B).
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(B)).

item(B).
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(B)).

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

- Contexts:

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

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

Examples

```
:- unit(dict(ST)).

dict(ST).

lookup(KEY, VALUE) :- ST=[KEY=VALUE|_].
lookup(KEY, VALUE) :- ST=[_|STx],
                      dict(STx) :> lookup(KEY, VALUE).
```

Examples

```
:- unit(dict(ST)).

dict(ST).

lookup(KEY, VALUE) :- ST=[KEY=VALUE|_].
lookup(KEY, VALUE) :- ST=[_|STx],
                      dict(STx) :> lookup(KEY, VALUE).
```

```
?- dict(D) :> ( lookup(a, 1),
                lookup(b, 2),
                lookup(a, X) ).

D = [a=1,b=2|_]
X = 1
```

WAM

Objective: in Minimum Context, Salvador Abreu and Daniel Diaz (2003).

Changes to the WAM:

- store the calling context and current context
- save the contexts on the creation of a choice point
- predicate call triggers the context resolution (new instruction)

Overhead:

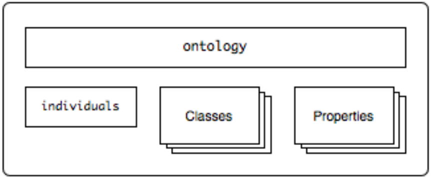
- 1.5 slowdown when compared to *regular* Prolog (no optimisations)
- with optimisations: no relevant overhead

Questions?

XPTO

XPTO **P**rolog **T**ranslation of **O**ntologies

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.

Query examples

James Bailey, François Bry, Tim Furche, and Sebastian Schaffert. Web and semantic web query languages: A survey. Reasoning Web (2005).

Query examples

James Bailey, François Bry, Tim Furche, and Sebastian Schaffert. Web and semantic web query languages: A survey. Reasoning Web (2005).

```
PREFIX   books: http://example.org/books#
PREFIX   rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
SELECT   ?essay, ?author, ?authorName, ?translator
FROM     http://example.org/books
WHERE    (?essay books:author       ?author),
         (?author books:authorName ?authorName)
OPTIONAL (?essay books:translator ?translator)
```

```
| ?- /> author(AUTHOR) :> item(ESSAY),
     /> authorName(AUTHORNAME) :> item(AUTHOR),
     /> optional(translator(TRANSLATOR)) :> item(ESSAY).
```

```
PREFIX    books: http://example.org/books#
CONSTRUCT (?x books:co-author ?y)
FROM      http://example.org/books
WHERE     (?book books:author ?x)
          (?book books:author ?y)
AND       (?x neq ?y)
```

```
| ?- /> author(X) :> item(BOOK),
     /> author(Y) :> item(BOOK),
     X \= Y,
     I = coauthor(X,Y).
```

Questions?