OO RuleML and OO jDREW

MARCEL A. BALL

CS-1, NRC

(**JOINT WORK** : H. Boley, B. Spencer et al.)

Introduction : What is OO RuleML and OO jDREW

– Object-Oriented RuleML is an extension to RuleML comprised of 3 modules

Introduction : What is OO RuleML and OO jDREW

– Object-Oriented RuleML is an extension to RuleML comprised of 3 modules
– OO jDREW is a logic reasoning engine, similar in design to Bruce Spencer's jDREW reasoning engine, that currently implements two of the new features found in OO RuleML

OO RuleML : New Features

– User-Level Roles

OO RuleML : New Features

- User-Level Roles
- Order-Sorted Types

OO RuleML : New Features

– User-Level Roles
– Order-Sorted Types
– URI-Grounded Clauses

OO jDREW : Syntaxes

OO jDREW accepts clauses in two different syntaxes

OO jDREW : Syntaxes

OO jDREW accepts clauses in two different syntaxes

– An ASCII syntax based upon elements from prolog and F-logic

OO jDREW : Syntaxes

OO jDREW accepts clauses in two different syntaxes

- An ASCII syntax based upon elements from prolog and F-logic
- A XML syntax based upon RuleML with additional elements and attributes

Positional-Rolled ASCII Syntax

```
offer(name->"Honda Element" : vehicle; price->?X !).
```

Positional-Rolled ASCII Syntax

```
offer(name->"Honda Element" : vehicle; price->?X !).
```

– User level roles represented by `<rolename> ->`

Positional-Rolled ASCII Syntax

```
offer(name->"Honda Element" : vehicle; price->?X !).
```

- User level roles represented by `<rolename> ->`
- Variables are prefixed by ? instead of prolog uppercase first letter

Positional-Rolled ASCII Syntax

```
offer(name->"Honda Element" : vehicle; price->?X !).
```

- User level roles represented by `<rolename> ->`
- Variables are prefixed by ? instead of prolog uppercase first letter
- Types are appended to terms, seperated by `:`

Positional-Rolled ASCII Syntax

```
offer(name->"Honda Element" : vehicle; price->?X !).
```

- User level roles represented by `<rolename> ->`
- Variables are prefixed by ? instead of prolog uppercase first letter
- Types are appended to terms, seperated by `:`
- `!` is a rest paramater which will match will all unused user-roles in the unifying clause

OO RuleML XML Syntax

```
<fact>
  <_head>
    <atom>
      <_opr><rel>offer</rel></_opr>
      <_r n="name"><ind type="SUV">Honda Element</ind></_r>
      <_r n="price"><var>P</var></_r>
      <_rest/>
    </atom>
  </_head>
</fact>
```

OO RuleML : User-level Roles

– Allow ’object-centered’ sets of role-filler slots instead of positional arguments

OO RuleML : User-level Roles

– Allow ’object-centered’ sets of role-filler slots instead of positional arguments
– Unordered slots allows for easier inheritance, and for easier, more compact, queries

Blending rolled arguments with positional arguments

– We can blend positional arguments and rolled elements within atoms/cterms

Blending rolled arguments with positional arguments

- We can blend positional arguments and rolled elements within atoms/cterms
- This can used to simulate required and optional arguments, with the first n positional arguments being require, and optional rolled arguments

Queries/Rules with and without rest variables

– *FACT* : `offer(name->"Honda Element"; category->special; price->20000).`

Queries/Rules with and without rest variables

- *FACT* : `offer(name->"Honda Element"; category->special; price->20000).`
- *QUERY* : `offer(name->"Honda Element"; price->?X).`

Queries/Rules with and without rest variables

- *FACT* : `offer(name->"Honda Element"; category->special; price->20000).`
- *QUERY* : `offer(name->"Honda Element"; price->?X).`

  Will not unify

Queries/Rules with and without rest variables

- *FACT* : `offer(name->"Honda Element"; category->special; price->20000).`
- *QUERY* : `offer(name->"Honda Element"; price->?X).`

  Will not unify
- *QUERY* : `offer(name->"Honda Element"; price->?X !).`

Queries/Rules with and without rest variables

- *FACT* : `offer(name->"Honda Element"; category->special; price->20000).`
- *QUERY* : `offer(name->"Honda Element"; price->?X).`

  Will not unify
- *QUERY* : `offer(name->"Honda Element"; price->?X!).`

  Will unify - has rest paramater

OO RuleML : Order Sorted Types

– Makes taxonomies (in RDFS) available as inheritance pathways for term typing

OO RuleML : Order Sorted Types

- Makes taxonomies (in RDFS) available as inheritance pathways for term typing
- Similar to class hierarchies in traditional Object Oriented programming languages

OO RuleML : Order Sorted Types

- Makes taxonomies (in RDFS) available as inheritance pathways for term typing
- Similar to class hierarchies in traditional Object Oriented programming languages
- Makes it possible to write rules and queries that should only apply to certain types of data

Order Sorted Types : Unification

– Sorted unification of two typed variables :

Order Sorted Types : Unification

– Sorted unification of two typed variables : Uses the RDFS sort hierarchy to find the greatest lower bound (glb) of the types, which becomes the type of the unified variable, or unification fails if the types do not have a glb

Order Sorted Types : Unification (cont)

- Sorted unification of typed variable and ind :

Order Sorted Types : Unification (cont)

– Sorted unification of typed variable and ind : The ind must be of the same type as the variable, or be a subclass in the RDFS sort hierarchy

Order Sorted Types : Unification (cont)

- Sorted unification of typed variable and ind : The ind must be of the same type as the variable, or be a subclass in the RDFS sort hierarchy
- Sorted unification of two typed inds :

Order Sorted Types : Unification (cont)

- Sorted unification of typed variable and ind : The ind must be of the same type as the variable, or be a subclass in the RDFS sort hierarchy
- Sorted unification of two typed inds : The inds must be of the same types, or the ind in the query/rule-body must be a superclass of the other ind in the RDFS sort hierarchy

OO RuleML : URI Grounding

– Allows using URIs as unique object identifiers (OIDs) for facts and rules.

OO RuleML : URI Grounding

- Allows using URIs as unique object identifiers (OIDs) for facts and rules.
- This feature of OO RuleML has not yet been implemented in OO jDREW

Weighted Extension and Similarity Unification

– Arguments to atoms and cterms are given weights in the w(eight) attribute to the `_r` tag. (*There is no representation for weights in the ASCII PR syntax*)

Weighted Extension and Similarity Unification

- Arguments to atoms and cterms are given weights in the w(eight) attribute to the `_r` tag. (*There is no representation for weights in the ASCII PR syntax*)
- Uses a tree similarity algorithm based upon the one described in *A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments [Bhavsar, Boley, Yang 03]*

Weighted Extension and Similarity Unification

– Arguments to atoms and cterms are given weights in the w(eight) attribute to the `_r` tag. (*There is no representation for weights in the ASCII PR syntax*)
– Uses a tree similarity algorithm based upon the one described in *A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments [Bhavsar, Boley, Yang 03]*
– Creates a "fuzzy-prolog" where facts and rules are given certainties, and query results have a certainty after unification - representing how ’sure’ we are of the result

Uses : RACOFI