Logic Programming in RuleML

Marcel A. Ball

March 23, 2005

Overview

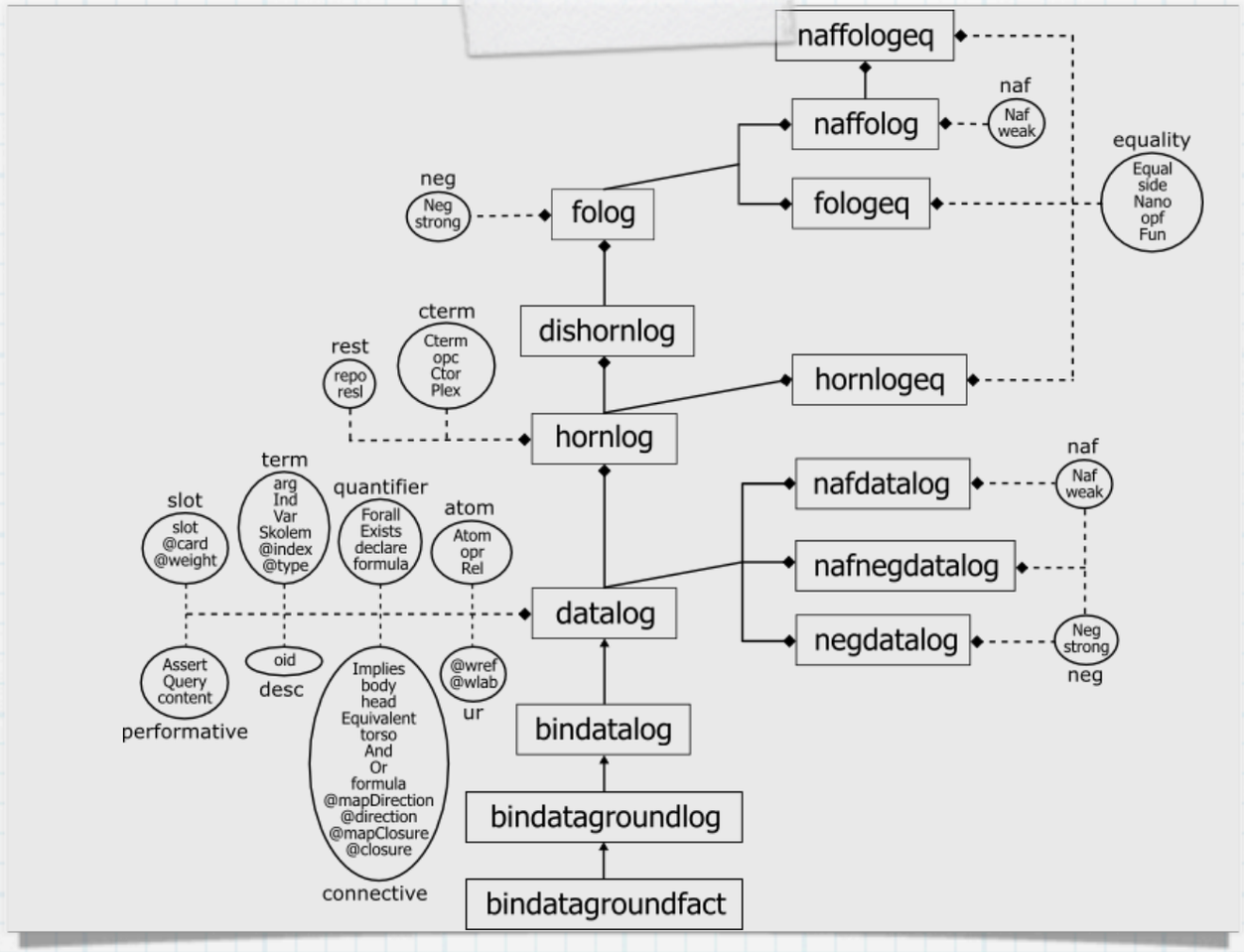
* Introduction to RuleML
* Basic RuleML Syntax
  * Break
* OO RuleML and OO jDREW
* OO jDREW Demo

Introduction to RuleML

- RuleML is an XML syntax for the mark-up of rules
- These slides are based on the 0.88 version of RuleML - the current stable release
  - Some tags are from the 0.89 release

RuleML Modularization

* RuleML is specified by a set of modular XSDs
* RuleML sub-languages are defined for:
  * folog (First Order Logic)
  * hornlog
  * datalog
* Focus for today is on hornlog and datalog sub-languages



RuleML Sub-languages

Simple Terms in RuleML: Constants

* Logical constant terms are represented with <Ind> tags in RuleML

Relfun/Prolog Constant	RuleML Constant
`peter`	`<Ind>Peter</Ind>`
`"John Doe"`	`<Ind>John Doe</Ind>`
`42`	`<Ind>42</Ind>`

Simple Terms in RuleML: Variables

* Logic variables are represented using the <Var> tag

Relfun/Prolog Variable	RuleML Variable
`Amount`	`<Var>amount</Var>`
`_`	`<Var />`
`_who`	`<Var>who</Var>`

Simple Terms in RuleML

* Symbol naming restrictions in Prolog and Relfun do not apply
* Constants can start with upper case
  * <Ind>John Doe</Ind> is valid
* Variables can start with lower case
  * <Var>who</Var> is also valid

Complex Terms in RuleML

- Complex terms (Structures) are represented using the <Cterm> tag
  - Constructor name represented by embedded <Ctor> tag
  - Argument terms embedded in <Cterm> tag
    - These arguments can be <Ind>, <Var>, <Cterm> or <Plex>

Complex Terms in RuleML

Relfun Structure

```
pair[i1, i2]
```

Prolog Structure

```
pair(i1, i2)
```

RuleML Structure

```
<Cterm>
   <Ctor>pair</Ctor>
   <Ind>i1</Ind>
   <Ind>i2</Ind>
</Cterm>
```

Complex Terms in RuleML: Lists

* Flat Lists from Relfun and Prolog can be represented using the <Plex> tag

Relfun/Prolog list

```
[i1,i2,..., in]
```

RuleML Plex

```
<Plex>
    <Ind>i1</Ind>
    <Ind>i2</Ind>
    ...
    <Ind>in</Ind>
</Plex>
```

Complex Terms in RuleML: Lists

* The RuleML <repo> (rest, positional) tag for the Prolog/Relfun "|" operator

Relfun/Prolog	RuleML

`[ Head | Rest ]`	`<Plex>`
	`  <Var>Head</Var>`
	`  <repo>`
	`  <Var>Rest</Var>`
	`  </repo>`
	`</Plex>`

RuleML Atomic Formulas

* Atomic Formulas are represented using the <Atom> tag
  * Relation name is represented as an embedded <Rel> tag
  * Arguments are embedded in <Atom> - these can be <Ind>, <Var>, <Cterm>, and <Plex>

RuleML Clauses: Facts

* Relfun/Prolog Fact

```
spending(“Peter Miller”, “min 500 euro”, “previous year”).
```

* RuleML Fact

```
<Atom>
   <Rel>spending</Rel>
   <Ind>Peter Miller</Ind>
   <Ind>min 500 euro</Ind>
   <Ind>previous year</Ind>
</Atom>
```

RuleML Clauses: Rules

* Rules are represented with the <Implies> tag
* First child element is the body of the rule
  can be either a single <Atom>
    or
  a conjunction of <Atom>s in an <And> tag
* Second child element is the head of the rule
  this must be an atomic formula (Atom)

RuleML Clauses: Rules

* Example Relfun/Prolog Rule

```
premium(Customer) :-
    spending(Customer, “min 500 euros”, “previous year”).
```

* Example RuleML Rule

```
<Implies>
  <Atom>
      <Rel>spending</Rel>
      <Var>Customer</Var>
      <Ind>min 500 euros</Ind>
      <Ind>previous year</Ind>
   </Atom>
   <Atom>
      <Rel>premium</Rel>
      <Var>Customer</Var>
   </Atom>
</Implies>
```

Body of Rule

Head of Rule

RuleML Knowledge Bases

* A RuleML knowledge base is a conjunction of clauses that are asserted to be true

```
<Assert>
    <And innerclose="universal">
        <!-- Clauses here -->
    </And>
</Assert>
```

RuleML Queries

* Queries are represented with <Query> tag
  * Contains one child to represent the body of the query - this is an <Atom> or a conjuction of <Atom>s in an <And>
* Example: `premium(Who)`

```
<Query>
   <Atom>
      <Rel>premium</Rel>
      <Var>who</Var>
   </Atom>
</Query>
```

Questions and Break

OO RuleML Features

* OO RuleML is a set of three orthogonal extensions to RuleML to facilitate Object-Oriented Knowledge Representation
  * Keyed non-positional arguments
  * Order-Sorted Types
  * URI Anchoring and Object Identifiers

OO jDREW Introduction

* OO jDREW is a reasoning engine that supports two of the OO RuleML features:
  * Keyed non-positional arguments
  * Order-Sorted types
* URI Anchoring and Object Identifiers to be supported in a future release
* Based upon the Bruce Spencer's jDREW
* More information available at http://www.jdrew.org/oojdrew

POSL Syntax

* A shorthand presentation syntax that integrates the positional syntax of Prolog with the slotted syntax of F-logic
* Only difference between Relfun and positional components of POSL is variables - must be prefixed with "?"
* Includes shorthand representation for the features of OO RuleML
* Can be used for knowledge bases in OO jDREW
  * Always used for queries and output in Top-Down version of OO jDREW

User Level Roles (Slots)

* The first of the three features added to RuleML to form OO RuleML
* Allows for non-positional (keyed) named arguments
* Removes ambiguity about the meaning of relational arguments
* When combined with the slotted rest (<resl>) allows omitting insignificant arguments - instead of having null values or anonymous variables

User Level Roles (Slots)

* Slots are represented with the <slot> tag
* <slot> contains two child elements
  * first must be an <Ind> - this is the name of the argument
  * second can be <Ind>, <Var>, <Cterm> or <Plex> - this is the value of the argument
* In POSL this is represented as `name->value`

User Level Roles (Slots)

* Consider the positional fact:

```
<Atom>
    <Rel>pow</Rel>
    <Ind>81</Ind>
    <Ind>3</Ind>
    <Ind>4</Ind>
</Atom>
```

* This is ambiguous
  * Can be interpreted as $3^4=81$ or $4^3=81$ or $3^{81}=4$, etc.

User Level Roles (Slots)

* This can be augmented with user-level roles to make the meaning clear

```
<Atom>
    <Rel>pow</Rel>
    <slot><Ind>result</Ind><Ind>81</Ind></slot>
    <slot><Ind>base</Ind><Ind>3</Ind></slot>
    <slot><Ind>exponent</Ind><Ind>4</Ind></slot>
</Atom>
```

* In this case it is clear that 3 is the base, 4 is the exponent, and 81 is the result
* In POSL: `pow(result->81; base->4; exponent->3).`

User Level Roles (Slots)

Fact

```
<Atom>
    <Rel>employee</Rel>
    <slot>
        <Ind>name</Ind>
        <Ind>John Doe</Ind>
    </slot>
    <slot>
        <Ind>age</Ind>
        <Ind>25</Ind>
    </slot>
    <slot>
        <Ind>sex</Ind>
        <Ind>male</Ind>
    </slot>
    ....
</Atom>
```

Query

```
<Query>
    <Atom>
        <Rel>employee</Rel>
        <slot>
            <Ind>sex</Ind>
            <Ind>male</Ind>
        </slot>
        <slot>
            <Ind>name</Ind>
            <Var>name</Var>
        </slot>
    </Atom>
</Query>
```

Fails

User Level Roles (Slots)

Fact

```
<Atom>
    <Rel>employee</Rel>
    <slot>
        <Ind>name</Ind>
        <Ind>John Doe</Ind>
    </slot>
    <slot>
        <Ind>age</Ind>
        <Ind>25</Ind>
    </slot>
    <slot>
        <Ind>sex</Ind>
        <Ind>male</Ind>
    </slot>
    ....
</Atom>
```

Query

```
<Query>
    <Atom>
        <Rel>employee</Rel>
        <slot>
            <Ind>sex</Ind>
            <Ind>male</Ind>
        </slot>
        <slot>
            <Ind>name</Ind>
            <Var>name</Var>
        </slot>
        <resl><Var /></resl>
    </Atom>
</Query>
```

Succeeds !

Order-Sorted Types

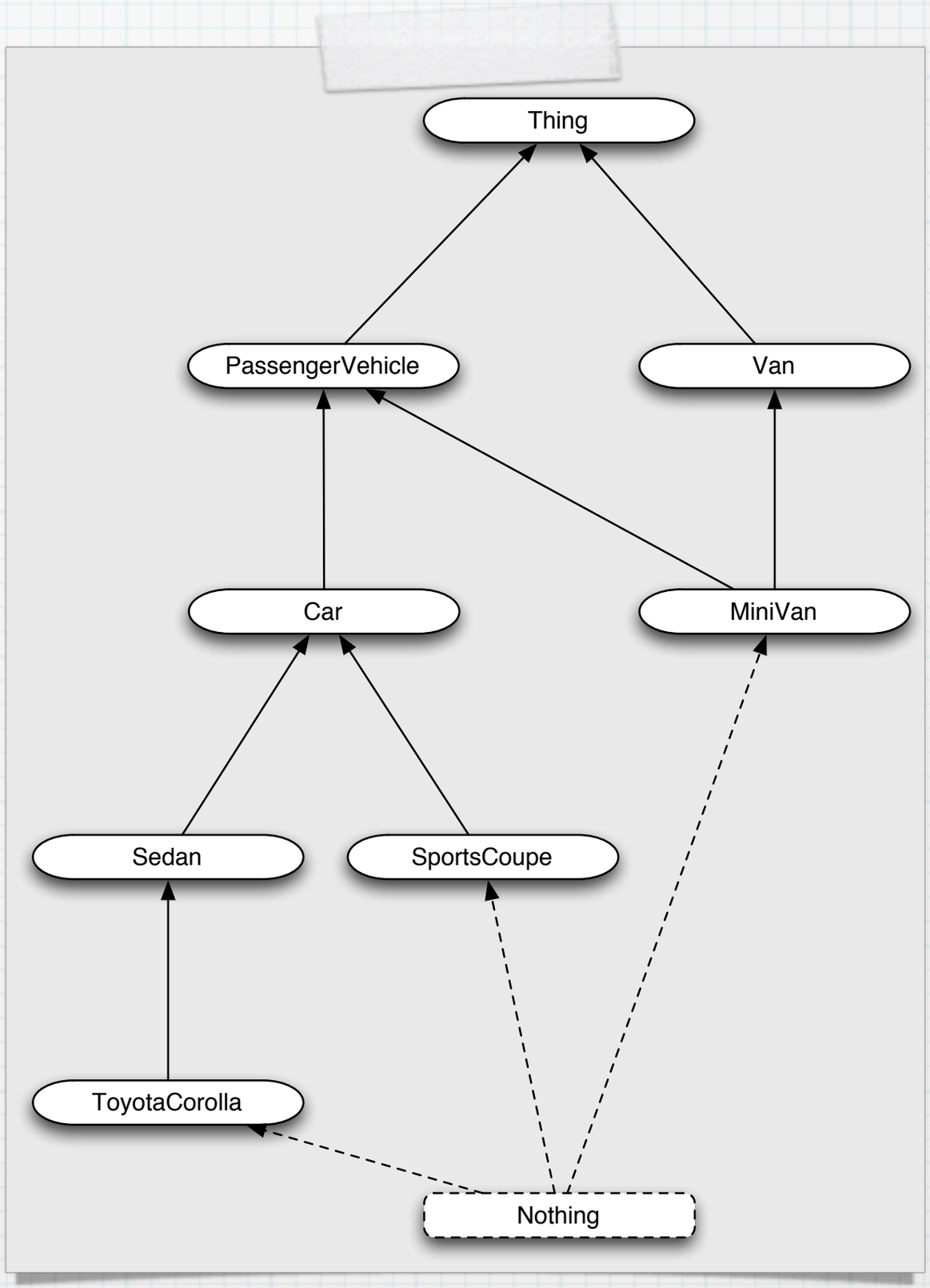
* Second major feature of OO RuleML that is supported by OO jDREW is Order-Sorted types
* Allows the user to define a partial order of types which can be used to add type information to terms
  * Restricts search space to clauses where the types restrictions are fulfilled

Order-Sorted Types

* Types in OO jDREW are defined using RDF Schema (RDFS)
  * Allows the reuse of lightweight taxonomies from the Semantic Web
* Currently only models taxonomic relationships between types
  * Cannot model the properties of types or the associated domain and range restrictions

Type Graphs

* Type information is represented as a directed acyclic graph (dag)
* Nodes of the graph represent types; edges represent sub-type -> parent-type relationship
* An Example type graph is included to the right



Types in OO RuleML

* Types can be assigned to to terms using the type attribute
  * Types are valid for <Ind>, <Var> and <Cterm> terms
  * <Plex>s and <Atom>s cannot be given types
* Examples:
  <Var type="Vehicle" />
  <Ind type="Sedan ">2000 Toyota Corolla</Ind>
* In POSL the type is indicated with `term : type`
  Example: `<Var type="Vehicle" />` is `?: Vehicle`

Typed Term Unification

* Unification of typed terms can be broken into three groups:
  * Unification of two typed variables
  * Unification of a typed variable and a typed non-variable
  * Unification of two typed non-variables

Typed Term Unification

* Unification of two typed variables:
  * The result of unifying two variables is a variable - the resulting type should inherit from both of the original types
* The most general type that meets this description is the greatest lower bound (glb) of the two types
* Unifying `<Var type="Van" />`
  and `<Var type="PassengerVehicle" />`
  Produces `<Var type="MiniVan" />`

Typed Term Unification

* Unification of typed variable and typed non-variable
  * To succeed type of non-variable must be the same or a sub-type of the type of the variable

* Examples:

  `<Var type="Car">vehicle</Var>` with

  `<Ind type="ToyotaCorolla">2003 Toyota Corolla</Ind>`

  will succeed

  `<Var type="MiniVan">vehcile</Var>` with

  `<Ind type="ToyotaCorolla">2003 Toyota Corolla</Ind>`

  will fail

Typed Term Unification

* Unification of two typed non-variable terms succeeds if and only if:
  * The type of term from the head of the fact or rule is the same as or inherits from the type of term from the body of the rule or query
  * Regular symbolic unification succeeds

OO jDREW Built-in Relations

* OO jDREW includes an easily expandable set of built-in relations
* The included built-ins are based upon the SWRL built-ins proposal; this covers the following areas:
  * comparisons, math, string operations, boolean operations, date/time operations, URI operations, and list operations

OO jDREW Built-in Relations

* OO jDREW currently has implementations of most of the built-ins from the following categories:
  * comparisons, math, string operations
* Date/Time comparisons are not currently supported
* Details of supported built-ins at http://www.jdrew.org/oojdrew/builtins.html

OO jDREW Built-in Relations

* Sample built-in calls:
  * addition of 2 integers
    * `add(?result, 8 : integer, 23 : integer)`
  * greater than comparison
    * `greaterThan(94 : integer, 34 : integer)`

Online Demos

* Online demos of OO jDREW available at
  http://www.jdrew.org/oojdrew/demo/
* Requires Java >= 1.4 with Java Web Start