The RuleML Family of Web Rule Languages

PPSWR'06, Budva, Montenegro, 10 June 2006
Revised, RuleML'06, Athens, GA, 11 Nov. 2006

Harold Boley
University of New Brunswick, Canada
National Research Council of Canada

Introduction

- Rules are central to the Semantic Web
- Rule interchange in an open format is important for e-Business
- RuleML is the de facto open language standard for rule interchange/markup
- Collaborating with W3C (RIF), OMG (PRR, SBVR), OASIS, DARPA-DAML, EU-REWERSE, and other standards/gov'nt bodies

RuleML Enables ...

Rule { modelling / markup / translation / **interchange** / execution / publication / archiving } in { UML / RDF / **XML** / ASCII }

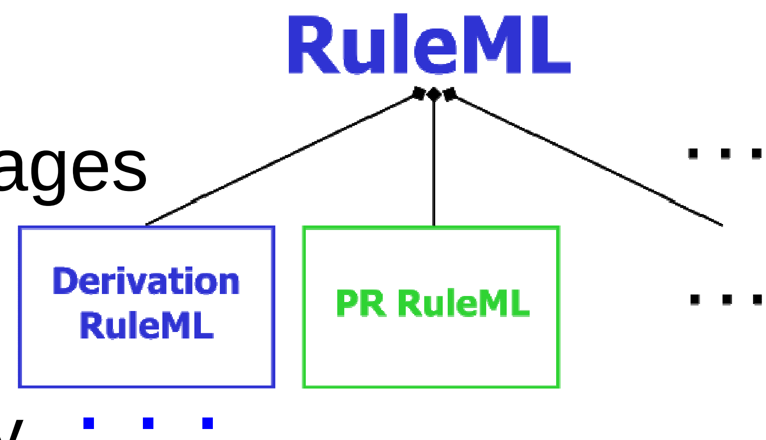
RuleML Identifies ...

- Expressive sublanguages
  - for Web rules
  - started with
    - *Derivation* rules: extend SQL views
    - *Reaction* rules: extend SQL triggers
  - to empower their subcommunities

RuleML Specifies ...

- Derivation rules via XML Schema:
  - All sublanguages:(OO) RuleML 0.91
  - First Order Logic: FOL RuleML 0.91
  - With Ontology language: SWRL 0.7
    - A Semantic Web Rule Language Combining OWL (W3C) and RuleML
  - With Web Services language: SWSL 0.9
- Translators in & out (e.g. Jess) via XSLT

Modular Schemas

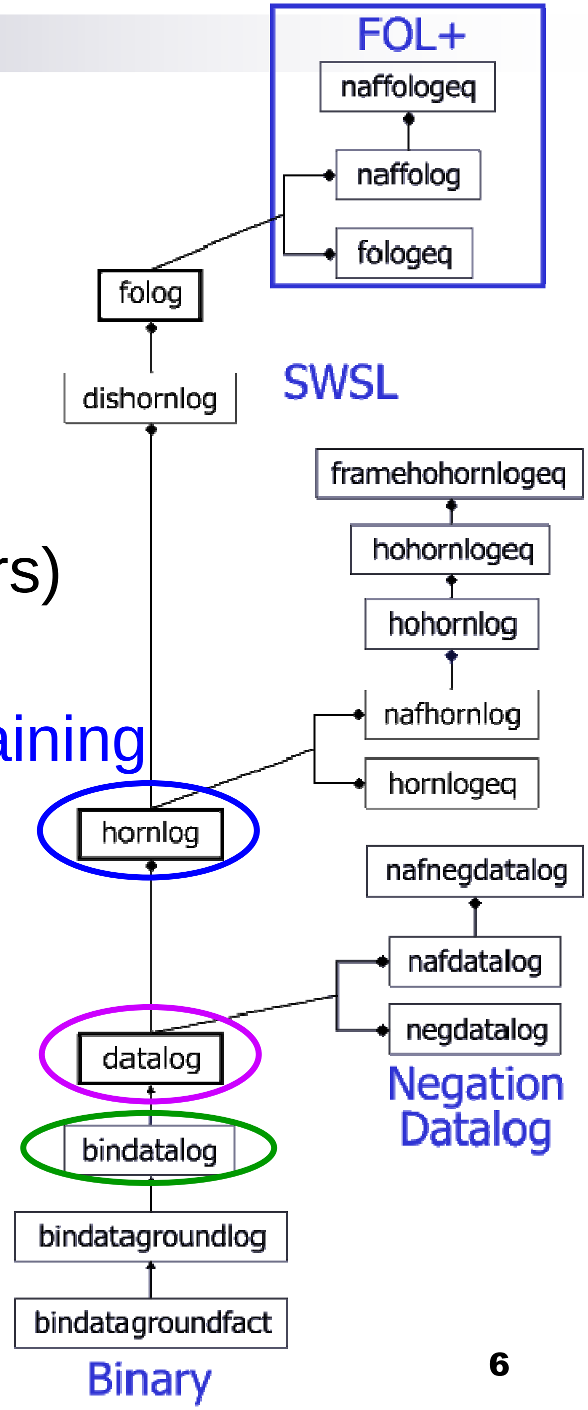


"RuleML is a **family** of sublanguages whose **root** allows access to the language as a whole and whose **members** allow to identify customized subsets of the language."

- RuleML: Rule Markup Language
  - RuleML derivation rules (shown here) and production rules defined in XML Schema Definition (XSD)
  - Each XSD of the family corresponds to the expressive class of a specific RuleML sublanguage
- The most recent schema specification of RuleML is always available at http://www.ruleml.org/spec
- Current release: RuleML 0.91
- Previews: http://wiki.ruleml.org/XSD_Workplan

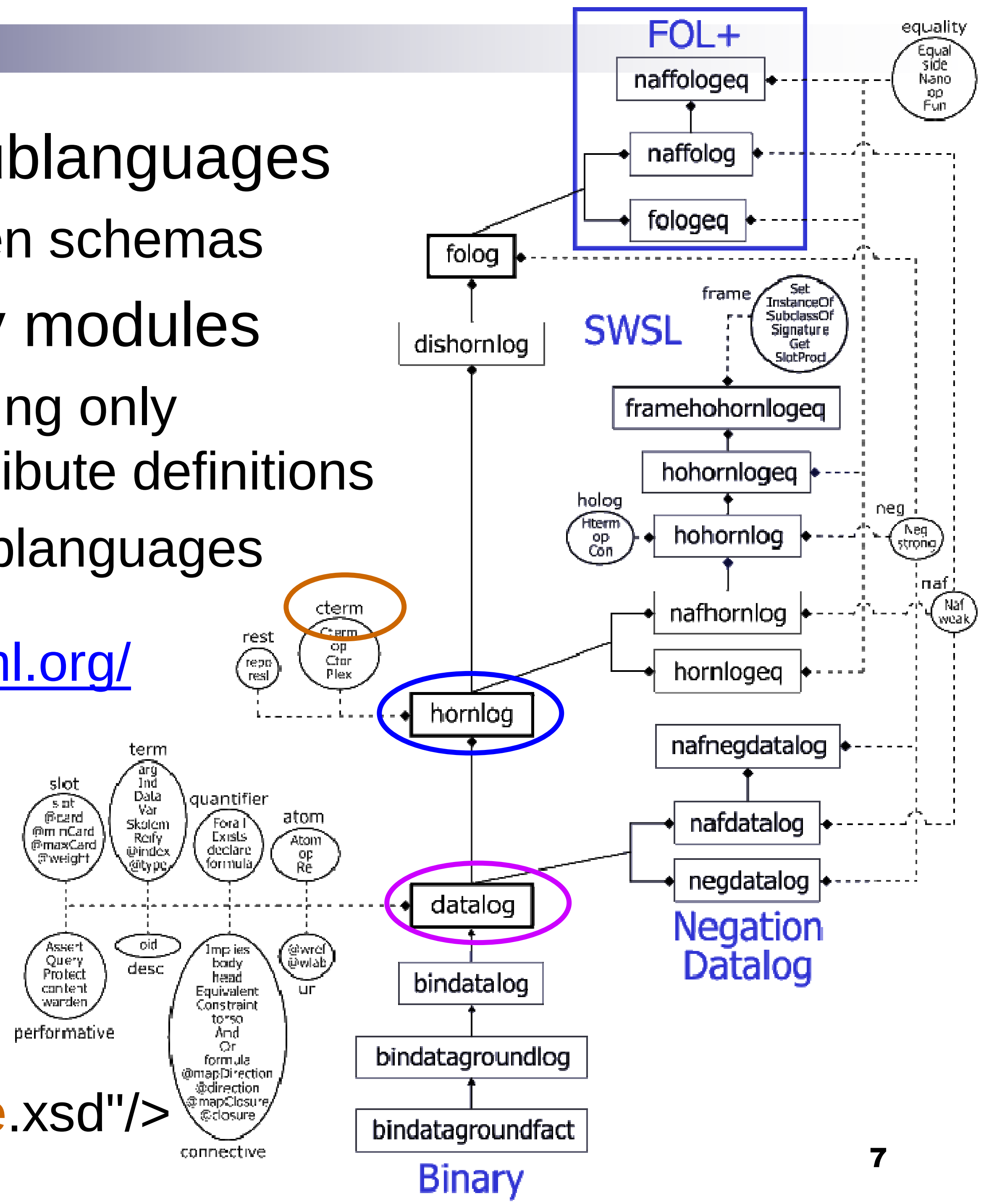
Schema Modularization

- XSD URIs identify expressive classes
  - Receivers of a rulebase can validate applicability of tools (such as Datalog vs. Hornlog interpreters)
  - Associated with semantic classes (such as function-free vs. function-containing Herbrand models)
- Modularization (Official Model)
  - Aggregation: e.g., Datalog *part of* Hornlog
  - Generalization: e.g., Bindatalog *is a* Datalog



- **Rectangles** are sublanguages
  - Inheritance between schemas
- **Ovals** are auxiliary modules
  - Elementary, including only element and/or attribute definitions
  - Become *part of* sublanguages

E.g., in http://www.ruleml.org/0.91/xsd/hornlog.xsd
<xs:redefine schemaLocation= "datalog.xsd">
<xs:include schemaLocation= "modules/cterm_module.xsd"/>

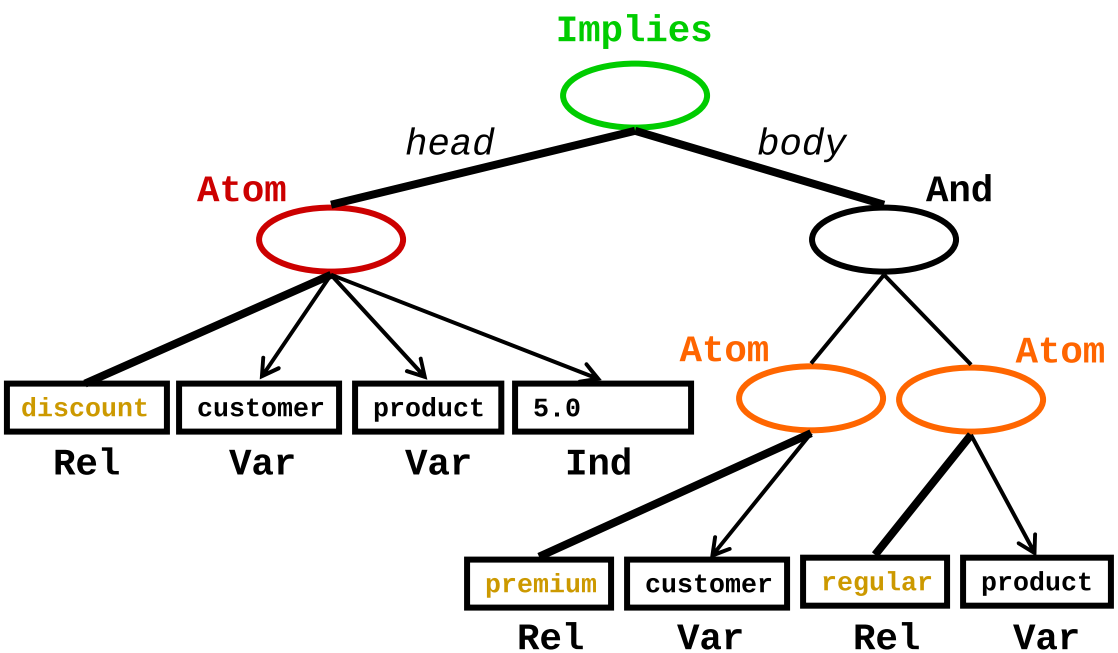


Bring Datalog to the Semantic Web

- Start with n-ary relations (not binary properties)
- Keep **Var**iable typing optional (reuse RDFS' subClassOf taxonomies as sort lattices)
- Allow signature declarations of arities and types
- Employ function-free facts as well as Horn rules (rather than 1st: RDF descriptions; 2nd: RDF rules)
- Use function-free Herbrand model semantics (querying stays decidable)
- Provide three syntactic levels:
  - User-oriented: Prolog-like, but with "?"-variables
  - Abstract: MOF/UML diagrams
  - XML serialization: Datalog RuleML

Business Rule: Positional

"The **discount** for a *customer* buying a *product* is 5 percent if the *customer* is **premium** and the *product* is **regular**."



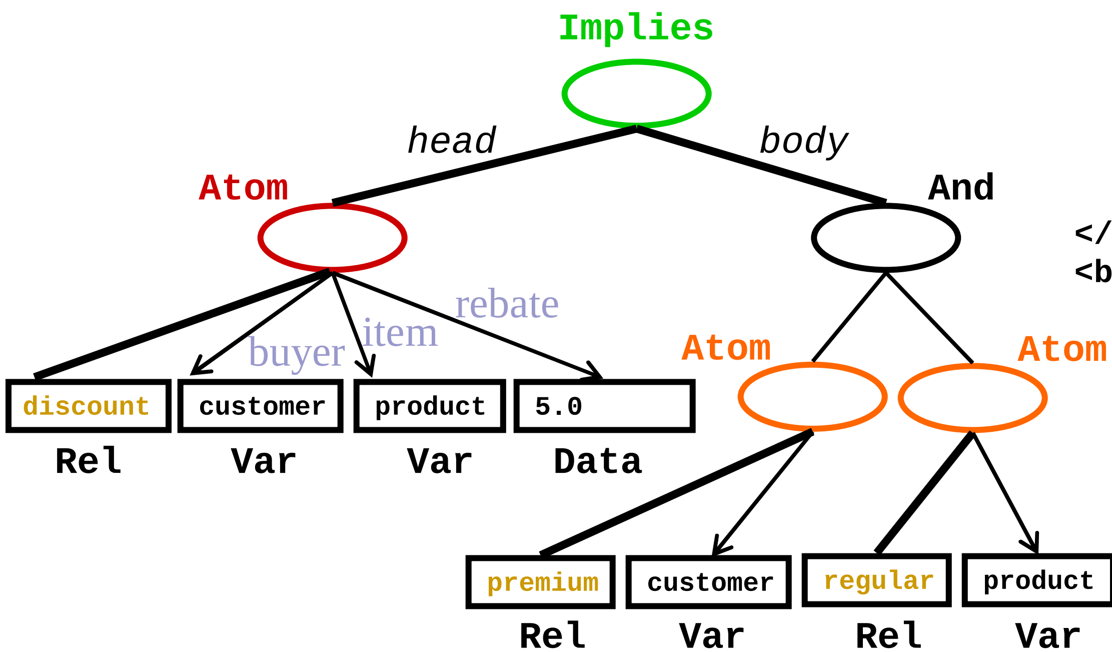

```
<Implies>
  <head>
    <Atom>
      <Rel>discount</Rel>
      <Var>customer</Var>
      <Var>product</Var>
      <Ind>5.0</Ind>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>customer</Var>
      </Atom>
      <Atom>
        <Rel>regular</Rel>
        <Var>product</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

Extend Datalog for the Semantic Web (I)

- Allow slots as name**`->`**filler pairs in **`Atom`**s (cf. F-logic's methods and RDF's properties)
- Extend optional types and signatures for slots
- Add optional object identifiers (**`oid`**s) to atoms
- Separate **`Data`** literals from **`Ind`**ividual constants

Business Rule: Slotted (for OO)

"The **discount** for a *customer* buying a *product* is 5 percent if the *customer* is **premium** and the *product* is **regular**."




```
<Implies>
  <head>
    <Atom>
      <Rel>discount</Rel>
      <slot><Ind>buyer</Ind><Var>customer</Var></slot>
      <slot><Ind>item</Ind><Var>product</Var></slot>
      <slot><Ind>rebate</Ind><Data>5.0</Data></slot>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>customer</Var>
      </Atom>
      <Atom>
        <Rel>regular</Rel>
        <Var>product</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

Extend Datalog for the Semantic Web (II)

- Permit IRI webizing for **`Data`** (XML Schema Part 2), **`Ind`**ividuals (RDF's **`resource`**s), **`Rel`**ations, **`slot`** names, types (RDFS' classes), and **`oid`**s (RDF's **`about`**)
- Introduce **`Module`** (scope) construct for clauses (cf. RDF's named graphs)
- Add scoped-default (**`Naf`**), strong (**`Neg`**), scoped-default-of-strong negation (unscoped: cf. ERDF)
- Integrate with Description Logics
  - Homogeneous (SWRL, Datalog RuleML + OWL-DL)
  - Hybrid (AL-log, Datalog^{DL}, DL+log, ...)

Bring Horn Logic to the Semantic Web

- Augment Datalog with uninterpreted **`Fun`**ctions and their **`Expr`**essions; also for extended Datalog
- Augment Datalog's Herbrand model semantics with such **`Fun`**ctions (querying becomes undecidable)
- Extend Datalog syntaxes
  - XML Schema of Hornlog RuleML inherits and augments XML Schema of Datalog RuleML
- Add **`Equal`**ity and **`in`**terpreted **`Fun`**ctions (XML serialization: attribute **`in="yes"`**)
- Reuse XQuery/XPath functions and operators as built-ins

Specify a First-Order Logic Web Language

- Layer on top of either
  - Disjunctive Datalog: **`Or`** in the head generalizing Datalog
  - Disjunctive Horn Logic: **`Or`** in head of near-Horn clauses
- Alternatively, layer on top of either
  - Disjunctive Datalog with restricted strong **`Neg`**ation
  - Disjunctive Horn Logic with restricted strong **`Neg`**
- Permit unrestricted **`Or`**, **`And`**, strong **`Neg`**, and quantifiers **`Forall`** and **`Exists`** to obtain FOL
- Use semantics of classical FOL model theory
- Extend Hornlog RuleML syntax to FOL RuleML

Equality for Functions

- Functional programming (FP) plays increasing Web role: MathML, XSLT, XQuery
- Functional RuleML employs orthogonal notions freely combinable with Relational RuleML
- Also solves a Relational RuleML issue, where the following ‘child-of-parent’ elements are separated:
  - Constructor (**`Ctor`**) of a complex term (**`Cterm`**)
  - User-defined function (**`Fun`**) of a call (**`Nano`**)
- Proceed to a **logic with equality**

Function Interpretedness (I)

- Different notions of ‘function’ in LP and FP:
- **LP:** *Uninterpreted functions* **denote** unspecified values when applied to arguments, not using function definitions
- **FP:** *Interpreted functions* **compute** specified returned values when applied to arguments, using function definitions
- E.g.: **`first-born`**: Man × Woman → Human
  - Uninterpreted: **`first-born(John, Mary)`** denotes first-born
  - Interpreted: using **`first-born(John, Mary) = Jory`**, so the application returns **`Jory`**

Function Interpretedness (II)

- Uninterpreted **`<Ctor>`** vs. interpreted **`<Fun>`** functions now distinguished with attribute values: **`<Fun in="no">`** vs. **`<Fun in="yes">`**
- Function applications with **`Cterm`** vs. **`Nano`** then uniformly become **`Expr`**essions
- Two versions of example marked up as follows (where "*u*" stands for **`"no"`** or **`"yes"`**):

```
<Expr>
 <Fun in="u">first-born</Fun>
  <Ind>John</Ind>
  <Ind>Mary</Ind>
</Expr>
```

Unconditional Equations

- Modified **`<Equal>`** element permits both symmetric and oriented equations
- E.g.: **`first-born(John, Mary) = Jory`** can now be marked up thus:

```
<Equal oriented="yes">
 <lhs>
  <Expr>
   <Fun in="yes">first-born</Fun>
   <Ind>John</Ind>
   <Ind>Mary</Ind>
  </Expr>
 </lhs>
 <rhs>
  <Ind>Jory</Ind>
 </rhs>
</Equal>
```

Conditional Equations

- Use **`<Equal>`** as the conclusion of an **`<Implies>`**, whose condition may employ other equations
- E.g.: **`?B = birth-year(?P)`** ⇒ **`age(?P) = subtract(this-year(),?B)`**

```
<Implies>
 <Equal oriented="no">
   <Var>B</Var>
   <Expr>
     <Fun in="yes">birth-year</Fun>
     <Var>P</Var>
   </Expr>
 </Equal>
 <Equal oriented="yes">
   <Expr>
     <Fun in="yes">age</Fun>
     <Var>P</Var>
   </Expr>
   <Expr>
    <Fun in="yes">subtract</Fun>
    <Expr>
      <Fun in="yes">this-year</Fun>
    </Expr>
    <Var>B</Var>
   </Expr>
 </Equal>
</Implies>
```

Accommodate SWSL-Rules

- HiLog: Higher-order **`Var`**iables, **`Con`**stants, and **`Hterm`**s (complex terms and atomic formulas at the same time)
- **`Equal`**: As in Horn Logic with (un**`oriented`**) **`Equal`**ity
- Frames:
  - Value molecules: **`Atom`**s with an **`oid`**, an optional **`Rel`** class, and zero or more name**`->`**filler instance **`slot`**s
  - Signature molecules: name**`=>`**filler class **`slot`**s, which can have **{**min**:**max**}** cardinality constraints
- Reification: A formula (e.g., a rule) embedded in a **`Reify`** element is treated (e.g., unified) as a term
- **`Skolem`**s: Unnamed, represent new individual constants (like RDF's blank nodes); otherwise, uniquely named ones

HiLog Examples: Hterms (I)

- First-order terms: **`f(a,?X)`**

```
<Hterm>
  <op><Con>f</Con></op>
  <Con>a</Con>
  <Var>X</Var>
</Hterm>
```

- Variables over function symbols: **`?X(a,?Y)`**

```
<Hterm>
  <op><Var>X</Var></op>
  <Con>a</Con>
  <Var>Y</Var>
</Hterm>
```

HiLog Examples: Hterms (II)

- Parameterized function symbols: **`f(?X,a)(b,?X(c))`**

```
<Hterm>
  <op>
    <Hterm>
      <op><Con>f</Con></op>
      <Var>X</Var>
      <Con>a</Con>
    </Hterm>
  </op>
  <Con>b</Con>
  <Hterm>
    <op><Var>X</Var></op>
    <Con>c</Con>
  </Hterm>
</Hterm>
```

Equality Example

- **Equal**ity **:=:** in rule head: **`f(a,?X):=:g(?Y,b) :- p(?X,?Y).`**

```
<Implies>
  <head>
    <Equal>
      <Hterm>
        <op><Con>f</Con></op>
        <Con>a</Con>
        <Var>X</Var>
      </Hterm>
      <Hterm>
        <op><Con>g</Con></op>
        <Var>Y</Var>
        <Con>b</Con>
      </Hterm>
    </Equal>
  </head>
  <body>
   <Hterm>
     <op><Con>p</Con></op>
     <Var>X</Var>
     <Var>Y</Var>
   </Hterm>
  </body>
</Implies>
```

Frame Example: Value Molecule

- Parameterized-name**->**filler **slot**: **o[f(a,b) -> 3]**

```
<Atom>
  <oid><Con>o</Con></oid>
  <slot>
    <Hterm>
      <op><Con>f</Con></op>
      <Con>a</Con>
      <Con>b</Con>
    </Hterm>
    <Con>3</Con>
  </slot>
</Atom>
```

Reification Example: Reified Rule

- **$**Rule as slot filler: **`john[believes -> ${p(?X) implies q(?X)}].`**

```
<Hterm>
   <oid>john</oid>
   <slot>
       <Con>believes</Con>
       <Reify>
          <Implies>
             <body>
                <Hterm>
                   <op><Con>p</Con></op>
                   <Var>X</Var>
                </Hterm>
             </body>
             <head>
                <Hterm>
                   <op><Con>q</Con></op>
                   <Var>X</Var>
                </Hterm>
             </head>
          </Implies>
       </Reify>
   </slot>
</Hterm>
```

Skolem Examples (I):

- Named **Skolem**: **holds(a,_#1) and between(1,_#1,5).**

```
<And>
  <Hterm>
    <op><Con>holds</Con></op>
    <Con>a</Con>
    <Skolem>1</Skolem>
  </Hterm>
  <Hterm>
    <op><Con>between</Con></op>
    <Con>1</Con>
    <Skolem>1</Skolem>
    <Con>5</Con>
  </Hterm>
</And>
```

Skolem Examples (II):

- Unamed **Skolem**: **holds(a,_#) and between(1,_#,5).**

```
<And>
  <Hterm>
    <op><Con>holds</Con></op>
    <Con>a</Con>
    <Skolem/>
  </Hterm>
  <Hterm>
    <op><Con>between</Con></op>
    <Con>1</Con>
    <Skolem/>
    <Con>5</Con>
  </Hterm>
</And>
```

Proceed towards Modal Logics

- Modal operators **generically** viewed as special **`Rel`**ations at least one of whose arguments is a proposition represented as an **`Atom`** with an uninterpreted **`Rel`**ation (including another modal operator, but not an arbitrary formula)
  - *Alethic* **`necessary`** (□) and **`possible`** (◇)
  - *Deontic* **`must`** and **`may`** (e.g., in business rules)
  - Open for *temporal* (e.g., when planning/diagnosing reactive rules), *epistemic* (e.g., in authentication rules), and further modal operators
- Towards a unified framework for multi-modal logic based on Kripke-style possible worlds semantics

Modal Examples: Alethic Operator

- Necessity: □ **prime(1)**

```
<Atom>
  <Rel modal="yes">necessary</Rel>
  <Atom>
    <Rel in="no">prime</Rel>
    <Data>1</Data>
  </Atom>
</Atom>
```

Modal Examples: Epistemic Operator

- Knowledge: **knows(Mary,material(moon,rock))**

```
<Atom>
  <Rel modal="yes">knows</Rel>
  <Ind>Mary</Ind>
  <Atom>
    <Rel in="no">material</Rel>
    <Ind>moon</Ind>
    <Ind>rock</Ind>
  </Atom>
</Atom>
```

Modal Examples: Epistemic Reasoning

- Veridicality axiom: $Knows_{Agent}\, proposition \rightarrow proposition$

  $Knows_{Mary}$ **`material(moon,rock)`** → **`material(moon,rock)`**

Serialization in previous slide

→

```
<Atom>
  <Rel in="yes">material</Rel>  <!-- "yes" is default -->
  <Ind>moon</Ind>
  <Ind>rock</Ind>
</Atom>
```

Modal Examples: Nested Operators

- Knowledge of Necessity: **knows(Mary,□prime(1))**

```
<Atom>
  <Rel modal="yes">knows</Rel>
  <Ind>Mary</Ind>
  <Atom>
    <Rel modal="yes" in="no">necessary</Rel>
    <Atom>
      <Rel in="no">prime</Rel>
      <Data>1</Data>
    </Atom>
  </Atom>
</Atom>
```

Protect Knowledge Bases by Integrity Constraints

- A knowledge base KB is a formula in any of our logic languages
- An integrity constraint IC is also a formula in any of our logic languages, which may be chosen independently from KB
- KB obeys IC

  iff

  KB entails IC

  (Reiter 1984, 1987)
  - Entailment notion of 1987 uses epistemic modal operator
- Serialization: **`<Entails>`** KB IC </ **`Entails>`**

Integrity Constraint Example: Rule with ∃-Head

- Adapted from (Reiter 1987):
  IC = {$(\forall x) emp(x) \Rightarrow (\exists y) ssn(x,y)$}
  KB_1 = {**emp(Mary)**} — KB_1 violates IC
  KB_2 = {**emp(Mary), ssn(Mary,1223)**} — KB_2 obeys IC

<**Entails**> KB_i IC </**Entails**>

KB_1:

```
<Atom>
  <Rel>emp</Rel>
  <Ind>Mary</Ind>
</Atom>
```

KB_2:

```
<Rulebase>
  <Atom>
    <Rel>emp</Rel>
    <Ind>Mary</Ind>
  </Atom>
  <Atom>
    <Rel>ssn</Rel>
    <Ind>Mary</Ind>
    <Data>1223</Data>
  </Atom>
</Rulebase>
```

IC:

```
<Forall>
  <Var>x</Var>
  <Implies>
    <Atom>
      <Rel>emp</Rel>
      <Var>x</Var>
    </Atom>
    <Exists>
      <Var>y</Var>
      <Atom>
        <Rel>ssn</Rel>
        <Var>x</Var>
        <Var>y</Var>
      </Atom>
    </Exists>
  </Implies>
</Forall>
```

Approach Production and Reaction Rules

- Share Condition (C) part with earlier languages as proposed for the RIF Condition Language
- Develop Action (A) part of Production Rules via a taxonomy of actions on KBs (Assert, Retract, ...), on local or remote hosts, or on the surroundings
- Develop Event (E) part of Reaction Rules via a corresponding taxonomy
- Create CA and ECA families bottom-up and map to relevant languages for Semantic Web Services
- Serialized: <**`Reaction`**> E C A </**`Reaction`**>
- See http://ibis.in.tum.de/research/ReactionRuleML TG

RDF Rules

- RDF-like Rules: Important RuleML sublanguage
  - Datalog: Relational databases augmented by views
  - RDF Properties: Slots permit non-positional, keyed arguments
  - RDF URIs/IRIs: Anchors provide **`O`**bject **`id`**entity via webzing through URIs/IRIs
    - **`oid`**s: Can be **`Ind`**ividuals, **`Var`**iables, etc.
    - **`iri`**s: Now used for both RDF's **`about`** and **`resource`**
  - RDF Blank Nodes: F-logic/Flora-2 Skolem-constant approach
    - E.g., Skolem generator '_' becomes **`<Skolem/>`**

“For a product whose price is greater than 200 and whose weight is less than 50, no shipping is billed.”

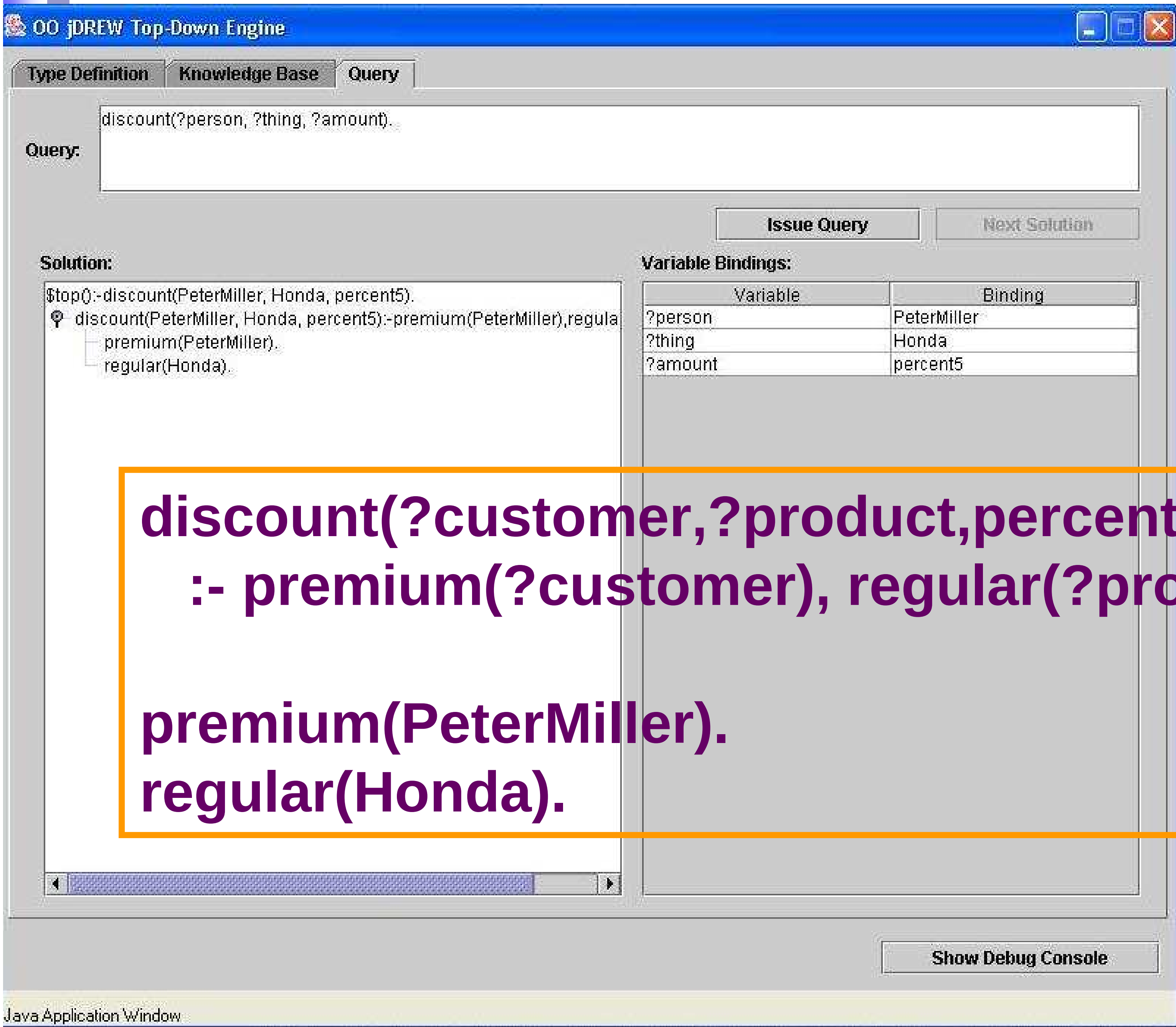
```
<Implies>
  <body>
    <And>
      <Atom>
        <oid><Var>x</Var></oid>
        <Rel>product</Rel>
        <slot><Ind iri=":price"/><Var>y</Var></slot>
        <slot><Ind iri=":weight"/><Var>z</Var></slot>
      </Atom>
      <Atom>
        <Rel iri="swrlb:greaterThan"/><Var>y</Var><Data>200</Data>
      </Atom>
      <Atom>
        <Rel iri="swrlb:lessThan"/><Var>z</Var><Data>50</Data>
      </Atom>
    </And>
  </body>
  <head>
    <Atom>
      <oid><Var>x</Var></oid>
      <Rel>product</Rel>
      <slot><Ind iri=":shipping"/><Data>0</Data></slot>
    </Atom>
  </head>
</Implies>
```

Bidirectional Interpreters in Java

- Two varieties of reasoning engines
  - **T**op-**D**own: backward chaining
  - **B**ottom-**U**p: forward chaining
- jDREW: *Java Deductive Reasoning Engine for the Web* includes both TD and BU
  http://www.jdrew.org
- OO jDREW: *Object-Oriented* extension to jDREW
  http://www.jdrew.org/oojdrew
- Java Web Start online demo available at
  http://www.jdrew.org/oojdrew/demo.html

OO jDREW Slots

- Normalized atoms and complex terms
  - **oid**s (object identifier)
  - Positional parameters (in their original order)
  - Positional rest terms
  - Slotted parameters (in the order encountered)
  - Slotted rest terms
- Efficient unification algorithm
  - Linear O(m+n): instead of O(m*n)
    - No need for positional order
    - Slots internally sorted
  - Steps:
    - Scan two lists of parameters
      - Matching up roles and positions for positional parameters
      - Unifying those parameters
    - Add unmatched roles to list of rest terms
    - Generate dynamically a Plex (RuleML's closest equivalent to a list) for a collection of rest terms

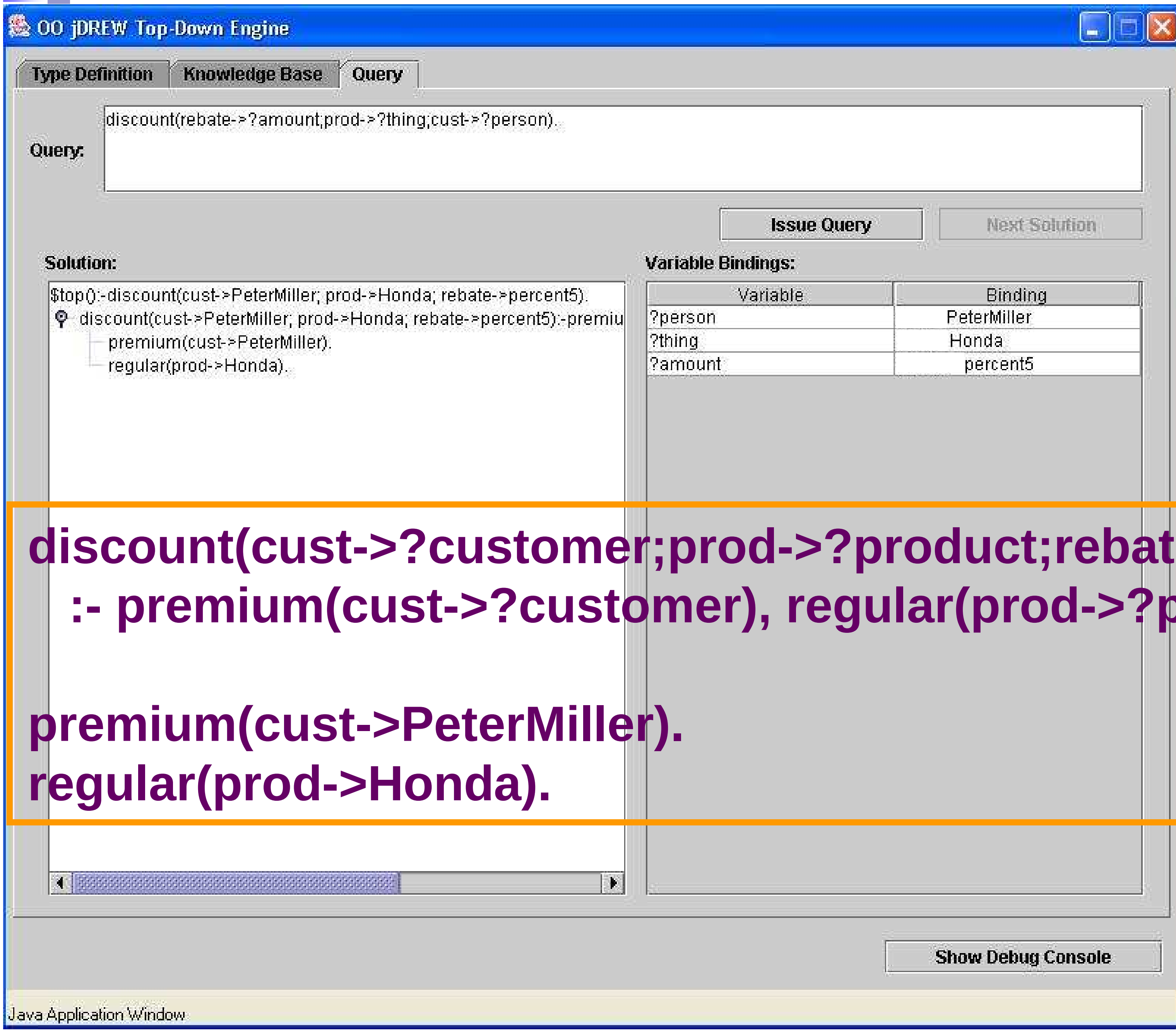


discount(?customer,?product,percent5)
:- premium(?customer), regular(?product).

premium(PeterMiller).
regular(Honda).

positional

POSL syntax



slotted

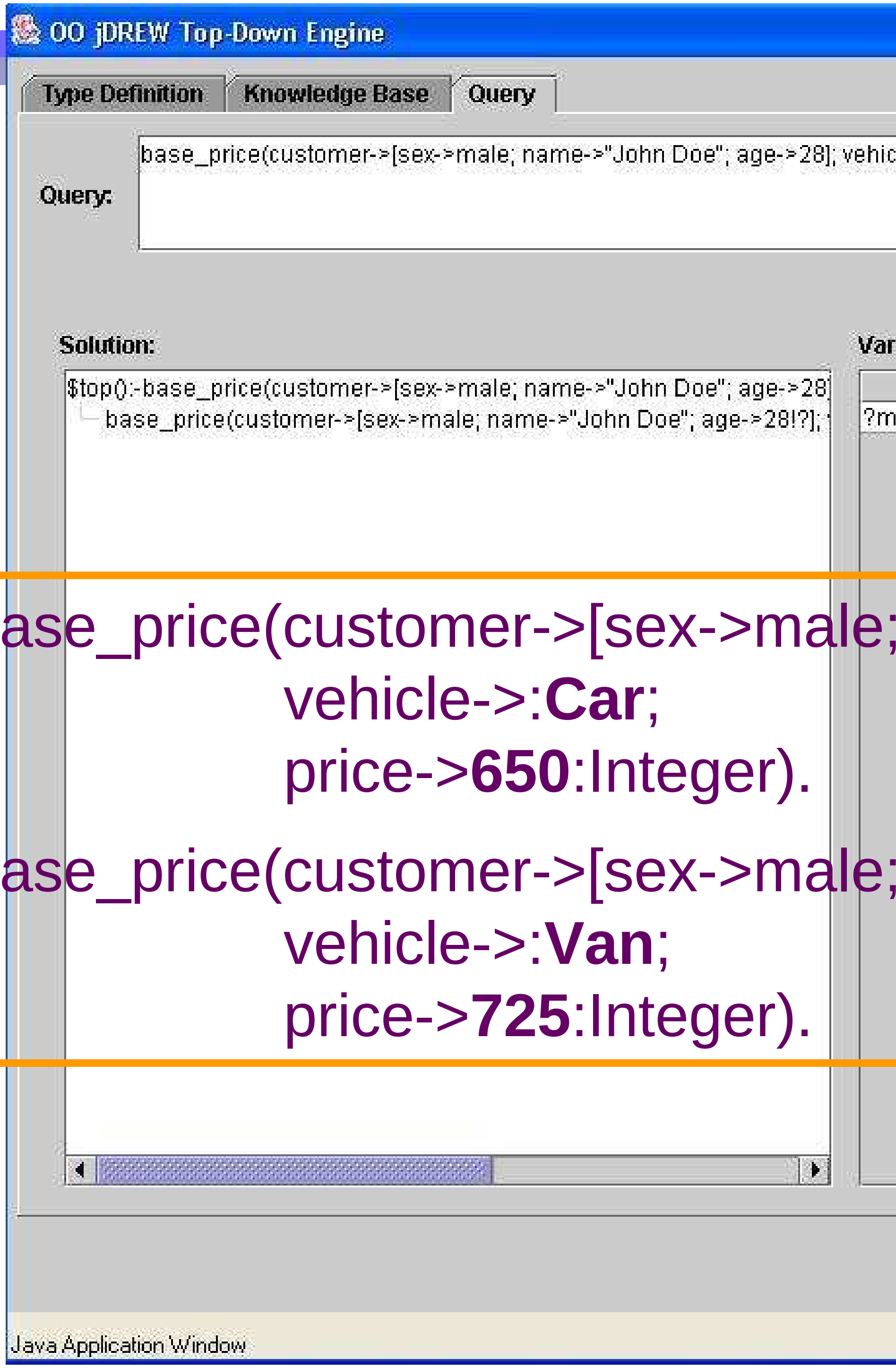
POSL syntax

discount(cust->?customer;prod->?product;rebate->percent5)
:- premium(cust->?customer), regular(prod->?product).

premium(cust->PeterMiller).
regular(prod->Honda).

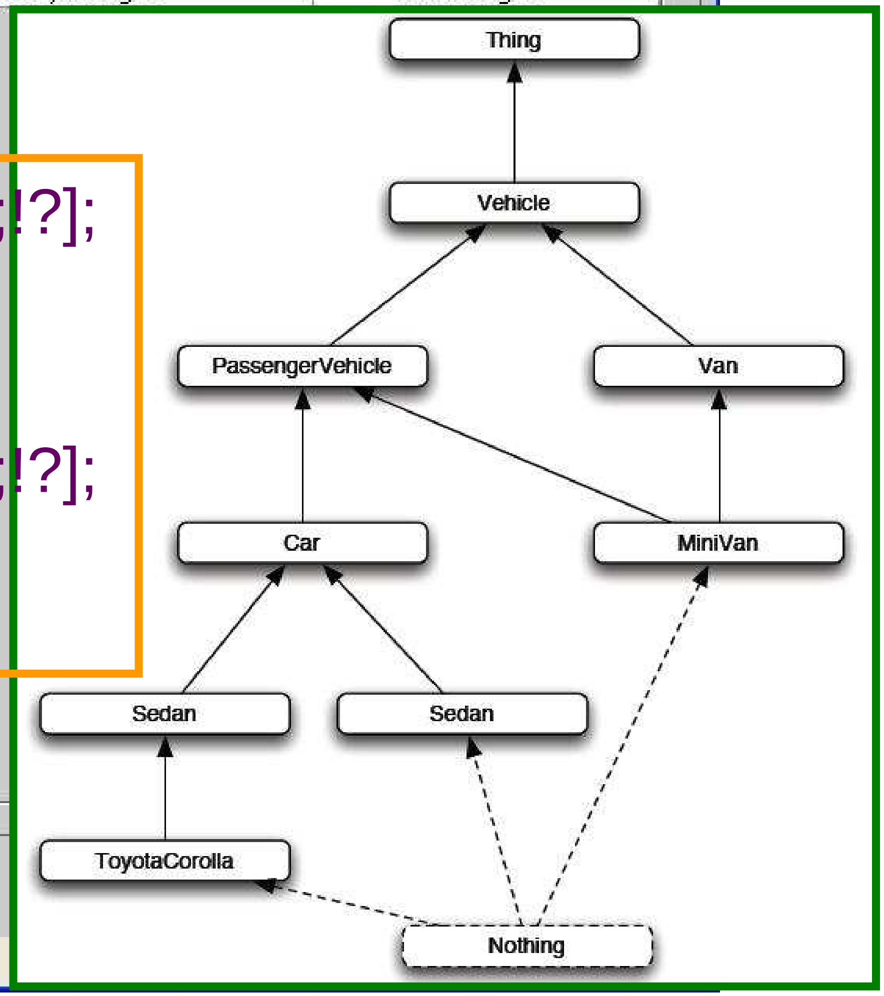
OO jDREW Types

- Order-sorted type system
  - RDF Schema: lightweight taxonomies of the Semantic Web
  - To specify a partial order for a set of classes in RDFS
- Advantages
  - Having the appropriate types specified for the parameters
  - To restrict the search space
  - Faster and more robust system than when reducing types to unary predicate calls in the body
- Limitations
  - Only modeling the taxonomic relationships between classes
  - Not modeling properties with domain and range restrictions



base_price(customer->[sex->male;!?];
vehicle->:**Car**;
price->**650**:Integer).

base_price(customer->[sex->male;!?];
vehicle->:**Van**;
price->**725**:Integer).



OO jDREW OIDs

- **`oid`**: Object Identifier
- Currently: symbolic names
  - In **`<Atom>`** & **`<Implies>`**
- Planned: **`iri`** attribute
- E.g., give name to fact keep(Mary, ?object).

```
<Atom>
 <oid><Ind>mary-12</Ind></oid>
 <Rel>keep</Rel>
 <Ind>Mary</Ind>
 <Var>object</Var>
</Atom>
```

```
<Atom>
 <oid><Ind iri="http://mkb.ca"/></oid>
 <Rel>keep</Rel>
 <Ind>Mary</Ind>
 <Var>object</Var>
</Atom>
```

```
<Atom>
 <oid><Var>object</Var></oid>
 <Rel>keep</Rel>
 <Ind>Mary</Ind>
 <Var>object</Var>
</Atom>
```

Conclusions

- RuleML is modular family, whose root allows to access the language as a whole and whose members allow customized subsets
- New members joining, e.g. Fuzzy RuleML
- Concrete & abstract syntax of RuleML
  - Specified by modular XSD (shown here) & MOF
- Formal semantics of OO Hornlog RuleML
  - Implemented by OO jDREW BU & TD
- Interoperability/Interchange of/with RuleML
  - Realized by translators, primarily via XSLT