The Java Deductive Reasoning Engine for the Web

Train people to build the Rule-based web services

- Courses on systems employing rule engines and Internet applications
  - Writing deduction engines in Java/C/C++
  - Interfacing with Internet API
  - Old techniques (Prolog 30 years ago)
  - New techniques from CADE System Competition
- Meier and Warren's book: Programming in Logic, 1988
  - Updated in Java?
  - Specific to Prolog at low level

Will such a course work?

- No
  - Guts of Prolog, Internet API's, how to program in logic
  - At least three courses here
- Yes
  - Students understand recursion
  - How to build a tree
    how to search a space
    - Propositional theorem prover
  - how to interface to Internet

This talk

- Architecture for building deduction systems
  - first order
  - easily configured
    - forward or backward
  - embedded
  - supports calls to and from rest of system
- Tour of internals
  - backward & forward engines
  - tree/proof
  - terms
  - bindings
  - discrimination tree
- Prototypes

Choose the right abstractions

- Goal, Unifier, ProofTree
- use Java iterators: pay as you go
  - for finding the next proof
- Make every Goal responsible for its list of matching clauses
  - hasNextMatchingClause()
  - attachNextMatchingClause()
- Place Goals in stack of backtrack points
  - popped in reverse chronological order

Propositional Prover

```
initially proofTree has an open Goal
loop
   if(proofTree.hasNoOpenGoal())
        halt('success');
   else
        Goal g = proofTree.selectOpenGoal();
        g.createMatchingClauseList();
        if(g.hasMoreMatchingClauses())
             DefiniteClause c = g.nextClause();
             g.attachClause(c);
             choicePoints.push(g);
        else
             chronologicalBacktrack();
```

```
chronologicalBacktrack
      while(choicePoints.nonEmpty())
            Goal g = choicePoints.pop();
            g.removeAttachedClause();
            if(g.hasMoreClauses())
                  return;
      halt('failure')
```

Remove bindings

Create bindings

Java Iterator

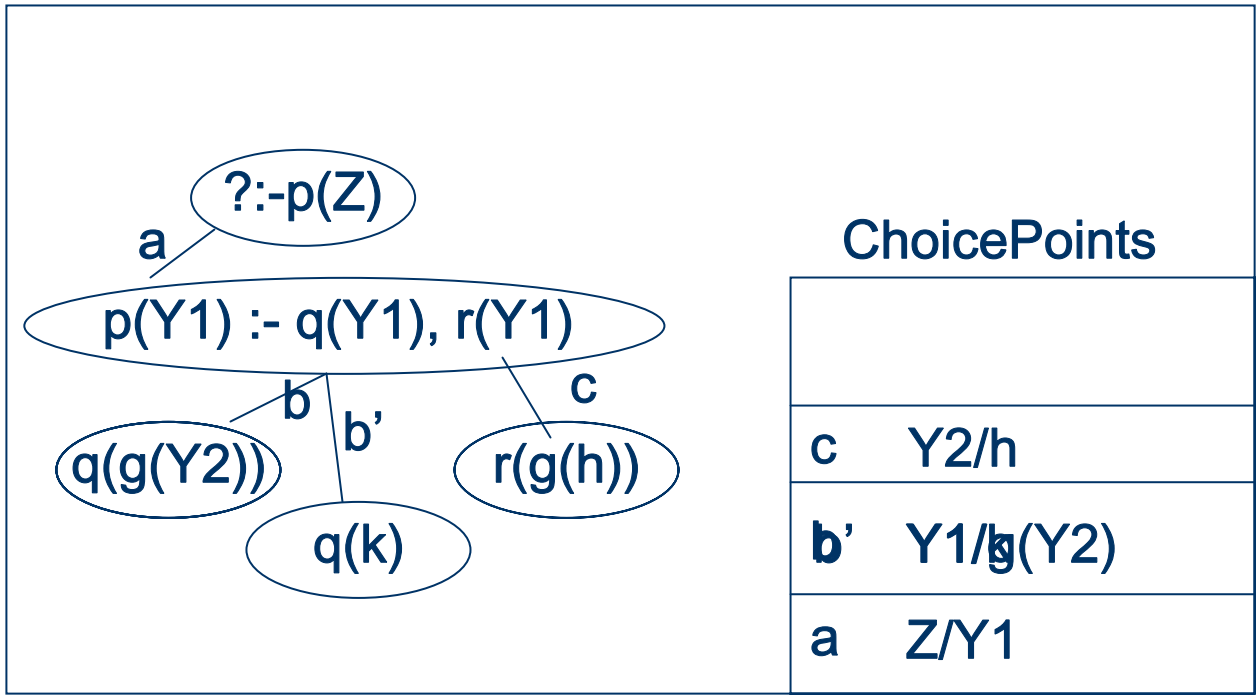
Tree with initial goal p(Z)

p(X) :- q(X), r(X).

q(g(X)).

q(k).

r(g(h))



Moving to First Order Logic

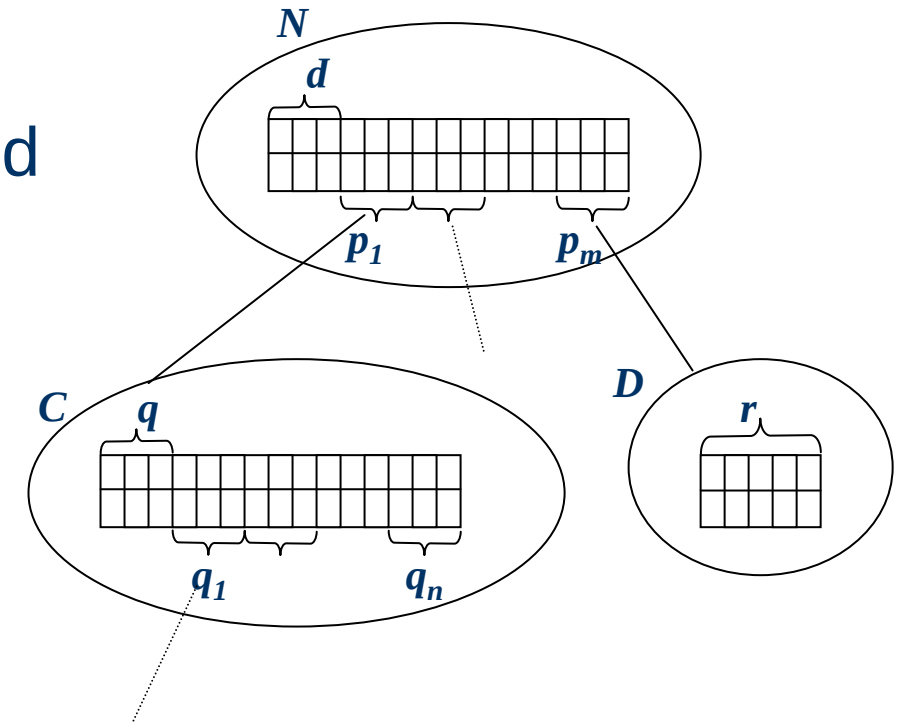
- Students struggle with variables
  - Unification
  - Composition of substitutions
  - Unbinding on backtracking
- Can we hide the hard stuff?
  - Powerful abstraction

Shallow or deep variables?

- Deep
  - variable binding is a list of replacements
  - traverse list for each lookup
  - undoing: remove most recent replacements
    $\{X \leftarrow f(Y)\} \bullet \{Y \leftarrow a\}$
- Shallow
  - an array of (all) variables and their current values
    [X ← f(a)
    Y ← a
    …]
  - undoing: pop stack of previous values (trail)

Goal Tree and flatterms

- Each node has head and body atoms
- Body atoms form goals
  - attach children
- resolved p_1 from
  $d \leftarrow p_1, \ldots, p_m$
  against q from
  $q \leftarrow q_1, \ldots, q_n$
- resolved p_m against $r \leftarrow$.



Flatterms to represent atoms

- *j*-DREW uses flatterms
  - Array of pairs:
    - symbol table ref
    - length of subterm
  - Not structure sharing
- Flatterms save theorem provers time and space (de Nivelle, 1999)
- Data transfer between deduction engine and rest of application

Symbol Table

	name	arity
1	p	2
2	f	1
3	g_1	0
4	g_2	0
5	h	0
6	h	1

Flatterm for $p(f(g_1), h, h(g_2), g_1)$

	symbol	length
1	1	7
2	2	2
3	3	1
4	5	1
5	6	2
6	4	1
7	3	1

Variables are clause-specific

- Variables use negative indexes
- Bindings are references to flatterm & position

- Unifier
  $X \leftarrow g_2$
  $Y \leftarrow f(g_2)$
  $W \leftarrow h(g_2)$
  $Z \leftarrow f(g_2)$

Flatterm for left
$p(f(h(X)), h(Y), f(X), Y)$

	symbol	length
1	1	9
2	2	3
3	6	2
4	-1	1
5	6	2
6	-2	1
7	2	2
8	-1	1
9	-2	1

	position	side
-1	6	right
-2	5	right

Flatterm for right
$p(f(W), h(f(g_2)), Z, Z)$

	symbol	length
1	1	8
2	2	2
3	-1	1
4	6	3
5	2	2
6	4	1
7	-2	1
8	-2	1

	position	side
-1	3	left
-2	5	right

Composing and undoing Bindings

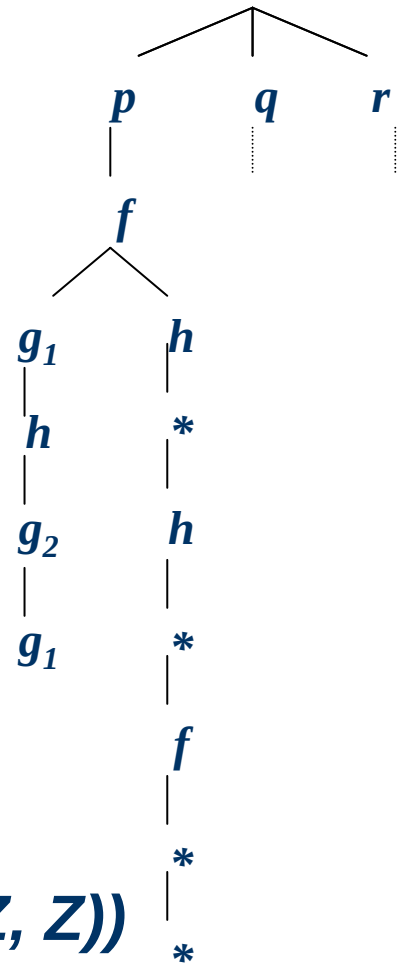
- Local shallow bindings currently do not allow composition
  - bindings must be done to a flatterm
  - new binding on a new flatterm
- Backtracking is integrated with unbinding
  - for quick unbinding, we use a stack of flatterms for each goal.

Discrimination trees

- Given a goal we want to access matching clauses quickly
- Every-argument addressing
  - unlike Prolog's first argument addressing
- Useful for RDF triples
  - a pattern may have variable in first argument
  - rdf(X, ownedby, 'Ora Lassila')

Discrimination trees

- Given a goal, want to access input clauses with matching heads quickly
- Index into clauses via a structure built from heads
- Replace vars by *
  - imperfect discrimination
- merge prefixes as much as possible
  - a tree arises

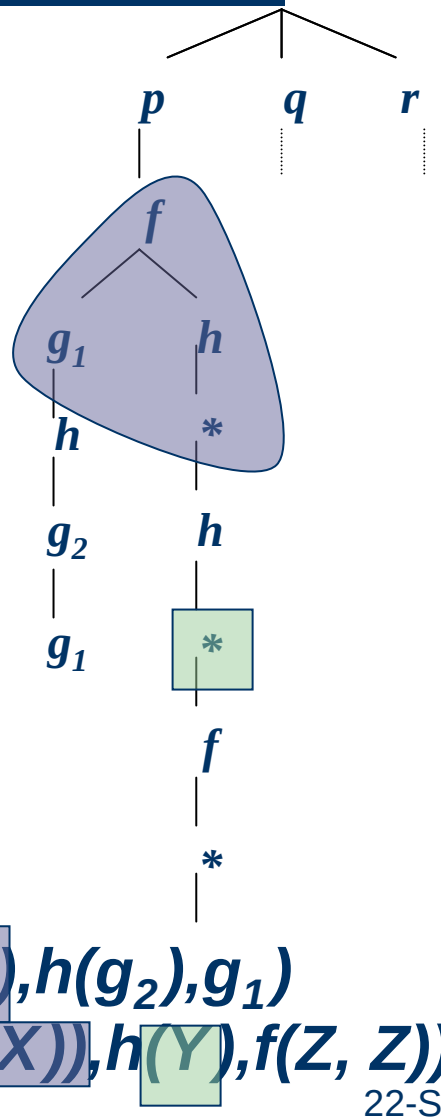


- **We added**
  p(f(g1),h(g2),g1)
  p(f(h(X)),h(Y),f(Z, Z))

Finding heads for goal p(X,h(g_2),Y)

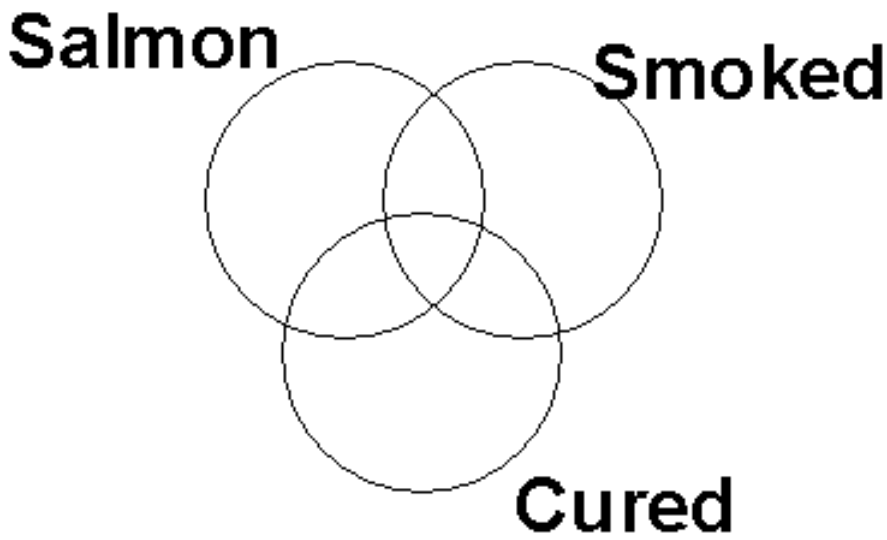
- replace vars in goal by *
  - *p(*,h(g_2),*)*
- Find instances of goal
  - * in goal, skip subtree

- **Find generalizations of goal**
  - *** in tree, skip term in goal**
- **Find unifiable**
  - **combination of both**

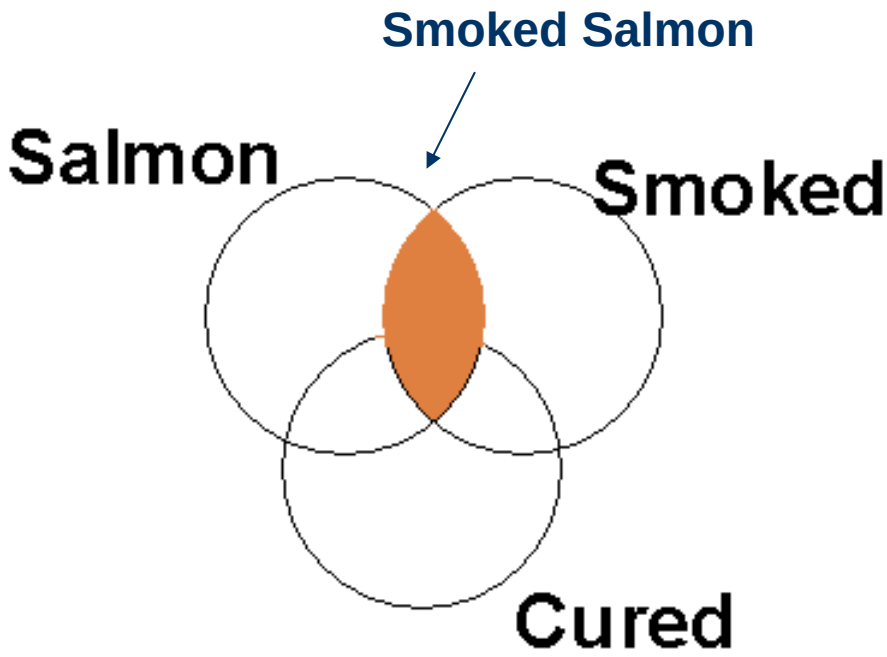


Iterator for matching clauses

- We use Java idioms where possible
- Java's iterators give access to sequence
  - next()
  - hasNext()
- Used for access to sequence of matching clauses
  - used in discrimination tree for access to roots leaves of skipped tree
    (McCune's term: jump-list)

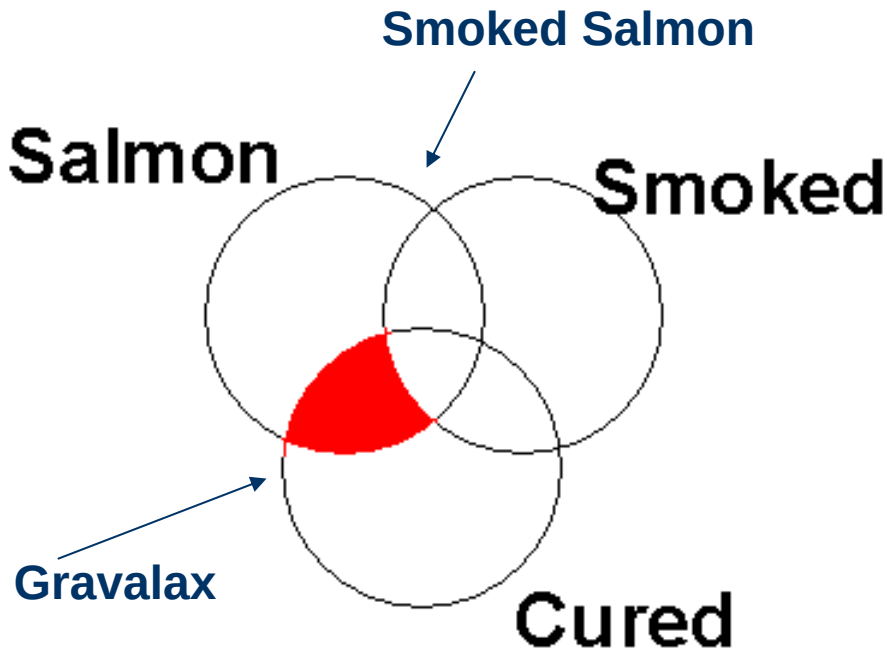

Salmon
Smoked
Cured

of Smoked and Salmon

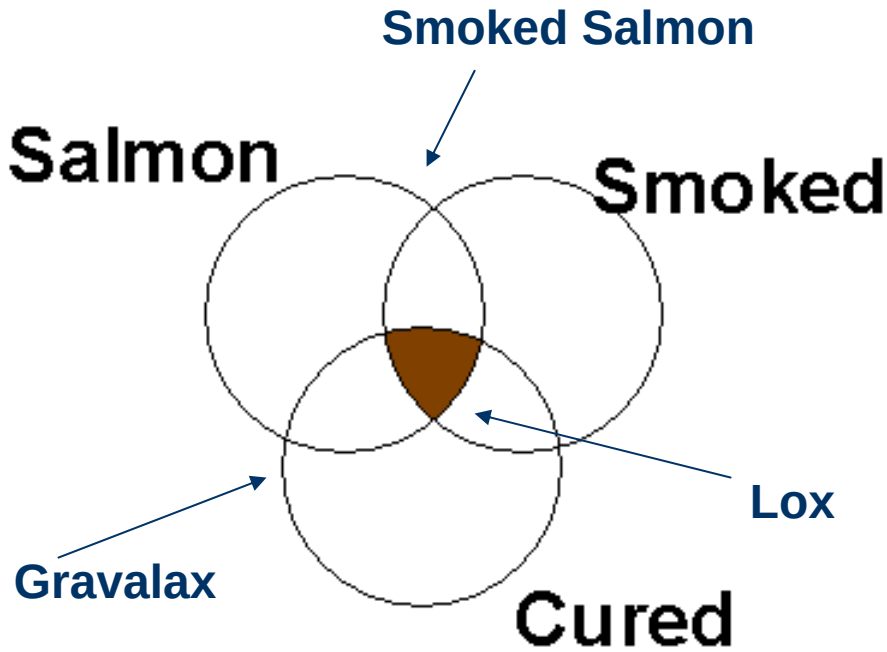


of Smoked and Salmon

•Gravalax is the intersection of Cured and Salmon, but not Smoked



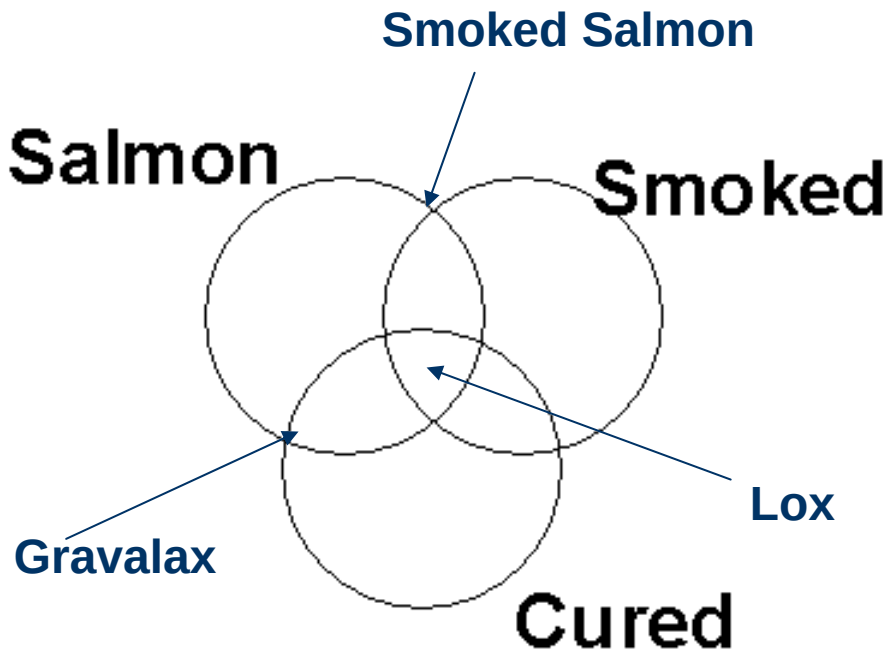
of Smoked and Salmon



- Gravalax is the intersection of Cured and Salmon, but not Smoked

- Lox is Smoked, Cured Salmon

- A search for keywords Salmon and Cured should return pages that mention Gravalax, even if they don't mention Salmon and Cured
- A search for Salmon and Smoked will return pages with smoked salmon, should also return pages with Lox, but not Gravalax



The Semantic Web vision is to make information on the web "understood" by computers, for searching, categorizing, …

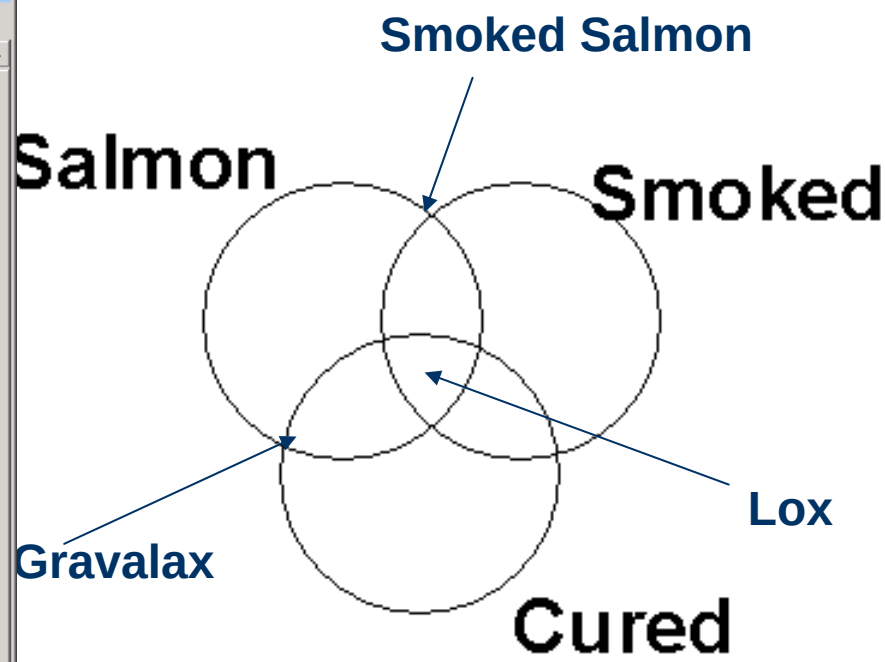
emacs@SPENCER-NB: c:/MyFiles/salmon.owl

Buffers Files Tools Edit Search Mule UNB Help

```
<owl:Class rdf:ID="Smoked"/>
<owl:Class rdf:ID="Cured" />
<owl:Class rdf:ID="Salmon" />

<owl:Class rdf:ID="SmokedSalmon">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:resource="#Smoked"/>
      <owl:Class rdf:resource="#Salmon" />
   </owl:insersectionOf>
</owl:Class>

<owl:Class rdf:ID="Gravalax">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:resource="#Cured"/>
      <owl:Class rdf:resource="#Salmon" />
   </owl:insersectionOf>
   <owl:disjointWith rdf:resource="#Smoked"/>
</owl:Class>

<owl:Class rdf:ID="Lox">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:resource="#Smoked"/>
      <owl:Class rdf:resource="#Cured"/>
      <owl:Class rdf:resource="#Salmon" />
   </owl:insersectionOf>
</owl:Class>
```

--** salmon.owl 10:22AM (Fundamental)--L10--C

Auto-saving...done



One possible encoding

Search criteria:

```
retrieve(P) :-
    about(P, cured),
    about(P, salmon).
```

A search for keywords Salmon and Cured should return pages that mention Gravalax, even if they don't mention Salmon and Cured.

Ontology:

```
about(P, cured) :-
  about(P, gravalax).
about(P, salmon) :-
  about(P, gravalax).
```

```
about(p1, gravalax).
```

retrieve(p1) succeeds

```
retrieve(P) :-
   about(P, smoked),
   about(P, salmon).

about(P, cured) :-
  about(P, lox).
about(P, salmon) :-
  about(P, lox).
about(P, smoked) :-
  about(P, lox).
about(P, cured) :-
  about(P, gravalax).
about(P, salmon) :-
  about(P, gravalax).
```

A search for Salmon and Smoked will return pages with smoked salmon, should also return pages with Lox, but not Gravalax.

```
about(p1, gravalax).
about(p2, lox).

retrieve(p1) fails
retrieve(p2) succeeds
```

Working Prototypes:

- Basic Prolog Engine
  - Accepts RuleML, or Prolog, or mixture
  - Iterator for instances of the top goal
  - Main loop is same code as propositional theorem prover (shown earlier)
  - Builds, displays deduction tree
    - available to rest of system
  - Negation as failure

More working prototypes: Variants of Top-Down Engine

- User directed
  - User selects goals
  - User chooses clauses
    - keeps track of clauses still left to try
  - Good teaching tool
- Bounded search
  - iteratively increase bound
  - every resolution in search space will eventually be tried
  - a fair selection strategy
- Original variable names supplied
  - particularly important for RuleML

Bottom-Up / Forward Chaining

- Set of support prover for definite clauses
- Facts are supports
- Theorem: Completeness preserved when definite clause resolutions are only between first negative literal and fact.
  - Proof: completeness of lock resolution (Boyer's PhD)
- Use standard search procedure to reduce redundant checking (next)
- Unlike OPS/Rete, returns proofs and uses first order syntax for atoms

Theorem Prover's Search Procedure

- Priority queue
  - new facts
- Discrimination trees:
  - used facts
  - rules, indexed on first goal

```
main loop
    select new fact
    for each matching rule
        resolve
        process new result
    add to old facts
```

```
process new result(C)
   if C is rule
      for each old fact matching first goal
          resolve
          process new result
      add C to rules
   else
      add C to new facts
```

Event – Condition - Action

- Suppose theorem prover saturates
  - may need datalog, subsumption
  - Then a new fact is added from
    - push process
    - Java event listener
  - adding a fact restarts saturation
    - could generate new Java events
- ECA interaction with Java 1.1 events

j-DREW sound and complete

- Sound unification
- Search complete variant
  - fair search procedure rather than depth-first
  - uses increasing bounds
- Sound negation
  - delay negation-as-failure subgoals
  - until ground or until only NAF goals remain

Related Work

- j-DREW compared to Prolog
  - j-DREW not compiled
  - More flexible
    - Dynamic additions
    - Web-ized
    - Programmer's API
  - Performance requirements different
    - *j*-DREW unlikely to yield megalips

Summary

- Architecture for Java-based reasoning engines
  - forward & backward
- Backward: variable binding/unbinding automatic
  - tied with choicepoints
  - configurable
- Integrated with other Java APIs
- Small footprint
  - Depolyed as thread, on server, on client, mobile
- Dynamic additions to rules
  - Integration of RuleML and Prolog rules in same proofs
- Proofs available