Semantic Web Technologies

Ontologies and Rules with F-Logic

Jos de Bruijn
jos.debruijn@deri.org

Digital Enterprise Research Institute (DERI)
University of Innsbruck, Austria

May 17, 2006

Outline

LP Syntax I

- LP is based on the Horn subset of FOL
- Vocabulary:
  - Constants `a, b, c, john, mary, 1, "bla"`
  - Predicates `p, q, r, married-to, manager-of`
  - Function symbols `f, g, h`
  - Variables `x, y, z`
- Terms:
  - Constants
  - Variables
  - Constructed terms (i.e. function symbol with arguments)
- Terms (examples):
  - `a, x, john, father-of(john), father-of(x), father-of(father-of(john))`
- Atoms:

LP Syntax II

- For n-ary predicate symbol p and terms $t_1, \ldots, t_n$, $p(t_1, \ldots, t_n)$ is an atom
- Examples: `p(x)`, `p(a)`, `hasName(john, "John")`, `married-to(john, mary)`, `hasGrandfather(x, father-of(father-of(x)))`
- A **ground** atom is an atom without variables
- Literals:
  - A positive literal is an atom: `p(x)`, `q(x)`, ...
  - A negative literal is a negated atom: `not p(x)`, `not q(x)`, ...
  - A ground literals is a literal without variables
- Rule:
  - The **head** of the rule consists of one positive literal H
  - The **body** of the rule consists of a number of literals $B_1, \ldots, B_n$
    `H :- B1,...,Bn`
  - $B_1, \ldots, B_n$ are also called **subgoals**

LP Syntax III

- Examples:

```
hasUncle(x,z) :- hasParent(x,y), hasBrother(y,z).
orphan(x) :- not hasParent(x,y).
hasParent(x,f(x)) :- person(x), not orphan(x).
```

- Note:
  `:-` (implication) is sometimes written as $\leftarrow$
  , is sometimes written as $\wedge$:

$$H \leftarrow B_1 \wedge \ldots \wedge B_n$$

- Fact: A ground atom (i.e. rule without a body)
  - `person(john).`
- Query:
  - A rule without a head:

LP Syntax IV

```
?- B1,...,Bn.
```

- Examples:

```
?- hasUncle(john,z).
?- married-to(john,mary).
?- person(x).
```

- Consider:

```
u(x,z) :- p(x,y), b(y,z).
p(john,jack).
b(jack,mike).

?- u(john,mike).
?- u(john,x).
```

- Built-in predicates:
  - Arithmetic comparators: >, <, =<, >=, =

LP Syntax V

```
3 < 4, 5 > x, ...
<(3,4), >(5,x), ...
```

- (in)equality: `=, !=`

  ```
  5=x, z=f(x), john!=mary, john=x, jack=father(john)
  ```

- May only occur in rule bodies

- Built-in functions:
  - Arithmetic functions: `+, -, *, /`

    ```
    1+2, 5/6, 5*x, ...

    +(1,2), /(5,6), *(5,x), ...
    ```

- Built-in functions can be reduced to predicates:
  `+(1,x) > *(8,5)` **reduces to** `y > z, add(1,x,y), multiply(8,5,z)`

LP Subsets

- Full logic programming
  - Function symbols
  - No negation
- Datalog as subset
  - No function symbols
  - No negation
  - Safe rules:
    - All variables are limited
  - A variable is limited if:
    - It appears as an argument in an ordinary (non-built-in) predicate of the body, or
    - The variable X appears in a subgoal $X = a$ or $a = X$, where a is a constant, or
    - The variable X appears in a subgoal $X = Y$ or $Y = X$, where Y is limited

LP Subsets (examples)

- Full Logic Programming:

```
hasParent(x,f(x)) :- person(x).
hasParent(x,y) :- person(x).
bigger(x,y) :- x > y.
```

- Datalog:

```
adult(x) :- person(x), hasAge(x,y), y >= 18.
hasParent(x,y) :- person(x), hasMother(x,y).
hasParent(x,y) :- person(x), hasFather(x,y).

ancestor(x,y) :- hasParent(x,y).
ancestor(x,z) :- ancestor(x,y), ancestor(y,z).
```

- Logic Programming vs. Deductive Databases
  - Logic Programming: using logic as a programming language (e.g. Prolog)
  - Deductive Databases: using logic as a database (query) language, e.g. Datalog

Dependency graph and recursive LP

- Dependency graphs
  - Each predicate is a node
  - There is an arc from predicate p to predicate q if they both occur in a rule where q is in the head and p in a subgoal
- A program is **recursive** iff its dependency graph contains a cycle
  - All predicates on such a cycle are called recursive
- Recursive:

```
ancestor(x,y) :- hasParent(x,y).
ancestor(x,z) :- ancestor(x,y), ancestor(y,z).
```

- Not recursive:

```
hasParent(x,f(x)) :- person(x).
hasParent(x,y) :- person(x).
bigger(x,y) :- x > y.
```

Datalog

- A logic for querying databases
- Extensional Database (EDB) consists of facts

```
person(john).
father(john,jack).
brother(jack,mike).
```

- Intentional Database(IDB) consists of non-ground rules

```
adult(x) :- person(x), hasAge(x,y), y >= 18.
parent(x,y) :- person(x), mother(x,y).
parent(x,y) :- person(x), father(x,y).
uncle(x,z) :- parent(x,y), brother(y,z).
```

- Consider the query:

```
?- uncle(john,mike).
```

Reasoning Problems

- Query answering:
  - Ground queries, e.g. is Mike an uncle of John?

    ```
    ?- uncle(john,mike).
    ```

  - Non-ground queries, e.g. who are the uncles of John?

    ```
    ?- uncle(john,x).
    ```

- Nonground queries can be reduced to ground queries, e.g.:
  - Replace `?- uncle(john,x)` With the three queries:

    ```
    ?- uncle(john,john).
    ?- uncle(john,jack).
    ?- uncle(john,mike).
    ```

- Answering ground queries is equivalent to entailment of ground facts:
  - For program P and ground atom A, does the following entailment hold?

$$P \models A$$

LP Semantics

- Model-theoretic semantics
  - What does a program/database mean in terms of its models?
  - We define the **minimal Herbrand model**.
- Computational semantics
  - How to compute the model of a program/database?
  - We define the direct consequence operator T_P for program P, which computes all consequences,
- Correspondence between model-theoretic and computational semantics
  - The fixpoint of T_P is the minimal Herbrand model!
- LP semantics is equivalent with First-Order semantics wrt. entailment of **ground facts**
  - Note: this only hold for logic programming without negation!

Herbrand Semantics: the idea

- Consider only particular first-order interpretations
  - Domain = set of ground terms (Herbrand universe)
  - Interpret ground terms as themselves: $t^I = t$
- Intersection of all models is **minimal model**

Ground Terms, ground atoms and the Herbrand Universe

- terms not containing any variables are **ground terms**
- Similarly, a ground atom is an atom not containing variables
- The **Herbrand Universe** U_L is the set of all ground terms which can be formed from constants and function symbols in program P.
  Example: `a, b, f(a), f(b), f(f(a)), f(f(b)), f(f(f(a))),...`
- The Herbrand Base B_P is the set of all ground atoms which can be built from predicate symbols in P, using ground terms from U_P as arguments.
  Example: `p(a),p(b),q(a),q(b),p(f(a)),q(f(a)),...`

Herbrand models

- A Herbrand Interpretation I_P is a subset of the Herbrand Base B_P.
- A Herbrand Model M_P is a Herbrand Interpretation which makes every formula true, i.e.:
  - Every fact in P is in M_P, and
  - For every rule R in P holds: if every positive literal in the body is in M_P, then also the head literal is in M_P.

  Note: this only works for positive programs, i.e., programs **without** negation!
- The semantics of a program P is characterized in terms of the **least** Herbrand Model, which is the intersection of all possible Herbrand Models.
- Each positive program has one **unique** least Herbrand Model.

Fixpoint Semantics

- Let I_P be a Herbrand interpretation and P a positive program:

$$T_P(I) = \{A \in B_P : \texttt{A} : -\texttt{A}_1, .., \texttt{A}_\texttt{n} \text{ is a ground instance of a rule in } P \text{ such that } A_1, ..., A_n \in I_P\}$$

  is called the **immediate consequence operator**.
- T_P is **monotonic**: $T_P^2(I) = T_P(T_P(I)) \supseteq T_P(I)$
- Thus, T_P has a fixpoint T_P^∞
- For a positive program, the least fixpoint of T_P is equivalent to the minimal Herbrand Model M_P:

$$lfp(T_P) = M_P$$

Example

```
a(x) :- b(x).
b(x) :- e(x).
l(karl,x) :- a(x).
b(franz).
e(hansi).
```

$$
\begin{aligned}
T_P(\emptyset) &= \{b(franz), e(hansi)\} && (1)\\
T_P^2(\emptyset) &= T_P(\emptyset) \cup \{a(franz), b(hansi)\} && (2)\\
T_P^3(\emptyset) &= T_P^2(\emptyset) \cup \{l(karl, franz), a(hansi)\} && (3)\\
T_P^4(\emptyset) &= T_P^3(\emptyset) \cup \{l(karl, hansi)\} && (4)\\
T_P^5(\emptyset) &= T_P^4(\emptyset) && (5)
\end{aligned}
$$

i.e. $lfp(T_P) = T_P^4(\emptyset) = M_P$ Note: $T_P^2(\emptyset) = T_P(T_P(\emptyset))$, $T_P^3 = T_P(T_P^2(\emptyset))$, etc...

Example (cont'd)

A simple example where the least Herbrand model is infinite:

```
p(a).
p(f(x)) :- p(x).
```

$$lfp(T_P) = \{p(a), p(f(a)), p(f(f(a))), \ldots\}$$

Negation

- How do we handle negation in Logic Programs?
- Negation-as-failure: whenever a fact is not entailed by the knowledge base, its negation is entailed
- Example:

```
p(a).
q(x) :- p(x)
```

entails:

```
p(a).
q(a).
not p(b).
not q(b).
not r(a).
not r(b).
...
```

Examples

- Using negative literals in rules.
- Example:

```
p(a).
q(x) :- not p(x).
```

entails:

```
p(a).
not q(a).
not p(b).
not q(b).
not r(a).
not r(b).
...
```

Examples

- Example:

```
p(a).
p(b).
r(a).
r(c).
q(x) :- not p(x), r(x).
```

  entails:

```
p(a).
p(b).
r(a).
r(c).
q(c).
```

- We typically leave out the entailed negative facts, since the negation of every fact which is not entailed, is entailed

Querying

- Ground queries:
  - The answer to a ground query Q_g is "yes" if:
    - Every positive literal in Q_g is in the minimal model of the program, and
    - Every negative literal in Q_g is not in the minimal model of the program.
  - Otherwise, the answer is "no".
- Non-ground queries are instantiated, in the same way as for logic programs without negation

Query Example

```
p(a).
p(b).
r(a).
r(c).
q(x) :- not p(x), r(x).
```

```
?- not q(a), r(c).
```

answer: **YES**

```
?- r(c), not q(c).
```

answer: **NO**

```
?- r(X), not q(X).
```

answer:

$X = \mathrm{a}$.

Outline

- Characterizing negative information in Herbrand models
- Characterizing negative information using the Fixpoint operator
- Problems with the Fixpoint operator under negation
- A popular solution: stratified negation

Extending Herbrand Semantics

- A Herbrand Model M_P is a Herbrand Interpretation which makes every formula true, i.e.:
  - Every fact in P is in M_P, and
  - For every rule R in P holds: if every positive literal in the body is in M_P and every negative literal in the body is not in M_P, then also the head literal is in M_P.
- The semantics of a program P is characterized in terms of the least Herbrand Model.
- But: is the least Herbrand model still unique?

Extending Herbrand Semantics (II)

Consider the following examples:

```
p(a).
q(b) :- not p(b).
```

The minimal Herbrand Model M_P is: $\{p(a), q(b)\}$

```
p(a).
q(x) :- not p(x).
```

The minimal Herbrand Model M_P is: $\{p(a)\}$

Extending Herbrand Semantics (III)

Consider the following examples:

```
r(a).
q(x) :- not p(x), r(x).
p(x) :- not q(x), r(x).
```

Has two minimal Herbrand Models: $\{r(a), q(a)\}, \{r(a), p(a)\}$

```
a :- not b.
b :- not a.
```

Has two minimal Herbrand Models: $\{a\}, \{b\}$

Extending T_P

- Let I_P be a Herbrand interpretation and P a general program:

$$T_P(I) = \{A \in B_P \mid A :- L_1, .., L_i, not\ L_{i+1}, .., not\ L_n \text{ is a ground instance of a rule in } P \text{ such that } L_1, ..., L_i \in I_P \text{ and } L_{i+1}, ..., L_n \notin I_P\}$$

  is the extended immediate consequence operator.
- Is the least fixpoint of T_P equivalent to the minimal Herbrand Model M_P?
- **Note: we have already seen cases of several minimal Herbrand models.**

Extending T_P (cont'd)

Consider the following examples:

```
p(a).
q(b) :- not p(b).
```

Applications of T_P:
(1){p(a)}
(2){p(a),q(b)}

(2) is the least fixpoint and is equivalent to M_P.

Extending T_P (cont'd)

Consider the following examples:

```
r(a).
q(x) :- not p(x), r(x).
p(x) :- not q(x), r(x).
```

Applications of T_P:
(1){r(a)}
(2){r(a),p(a),q(a)}

(2) is a fixpoint, but not a Herbrand model! Remember, the Herbrand models are: $\{r(a), q(a)\}, \{r(a), p(a)\}$

Thus, naive extension of the fixpoint operator is not possible!

Solution: Stratified Programs

- Recall: dependency graph
  - Each predicate is a node
  - There is an edge from predicate p to predicate q if they both occur in a rule where q is in the head and p in a subgoal
  - Edges with negation are marked
- A predicate is in the same stratum as all predicates connected with a positive edge.
- If there is a negative edge leading from a predicate p to a predicate q, then the stratum of q is one higher than the stratum of p.
- If every predicate occurs in at most one stratum, the program is stratifiable.
- For stratified programs, we can use the extended fixpoint operator T_P!

Examples of Stratification

```
p(a).
q(b) :- not p(b).
```

Is stratified with:
Stratum 0: {p} Stratum 1: {q}

```
p(a).
r(a).
q(x) :- not p(x), r(x).
```

Is stratified with:
Stratum 0: {p} Stratum 1: {q,r}

Examples of Stratification

```
r(a).
q(x) :- not p(x), r(x).
p(x) :- not q(x), r(x).
```

Is not stratified. **Why**?

```
a :- not b.
b :- not a.
```

Is not stratified. **Why**?

F-Logic extensions: motivation

- Predicates not really appropriate to capture object typing and attributes.
- Consider: John is a Lawyer, has the particular name "John Smith" and has two children named "Mary" and "Jill". Furthermore, Lawyer is a particular kind of Human.
- Expressed using Logic Programming:

```
lawyer(john).
hasName(john, "John Smith").
hasChild(mary).
hasChild(jill).
hasName(jill, "Jill").
hasName(mary, "Mary").
person(x) :- lawyer(x).
```

- This encoding:
    - Disregards the structure of the natural language sentences
    - Information about the particular object is no longer grouped

F-Logic Introduction

- Consider: John is a Lawyer, has the particular name "John Smith" and has two children named "Mary" and "Jill". Furthermore, Lawyer is a particular kind of Human.
- Expressed using F-Logic:

```
john:lawyer[name->"John Smith",
            child->>{mary[name->"Mary"],
            jill[name->"Jill"]}].

lawyer :: person.
```

- F-Logic also allows specifying signature information:

```
person[name=>string,
       child=>>person].
```

F-Logic

- Frame Logic (F-Logic) allows capturing different information about an object in a frame.
- Basic building blocks are molecules
- isa molecules: `A : B` (class membership) or `A :: B` (subclassing), for objects `A,B`

  For example:

  ```
  john:lawyer, laywer::person, student::person,
  jack:student, X:Y, john:X
  ```

F-Logic (2)

- attribute molecules: `A[B op C]` with op ∈ {`->`, `->>`, `=>`, `=>>`} and objects `A,B,C`

  For example:
  Single valued attributes: `john[name->"John Smith"]`, `jill[marriedTo->mark]`

  Set-values attributes: `john[children->>mary, jill]`

  Single valued attribute signature: `X[name=>string]`, `employee[salary=>amount]`

  Set-values attribute signature: `person[children=>>person]`, `X[employee=>person]`

F-Logic (3)

- Arbitrary terms (functions, variables) can occur in place of object ids, e.g.:

```
father(john)[name->"Jack"], X[name->"John"]
```

- Molecules starting with the same object ID may be combined and nested:

```
john:lawyer[name->"John Smith";
            child->>{mary[name->"Mary"];
            jill[name->"Jill"]}].

lawyer :: person.
```

- F-Logic allows the use of predicate symbols, besides the molecules
- Full F-Logic is a (syntactical) extension to First-Order Logic; we only consider an F-Logic extension to Logic Programming.

F-Logic Programming

- An F-Logic vocabulary V consists of:
  - Object identifiers: `a, b, john, jack, f, g, father`, etc...
  - Variables: `X, Y, Z, ...`
  - Predicate symbols: `p, q, ...`
- Given an F-Logic vocabulary V:
  - Every object identifier is a term
  - Every variables is a term
  - If `f` is an object identifier and `t1,...,tn` are terms, then `f(t1,...,tn)` is a term, also called a constructed term

    Thus, every object identifier can be used as term constructor!

F-Logic Programming (2)

- If `p` is a predicate symbol and `t1,...,tn` are terms, then `p(t1,...,tn)` is an atom.
- For `A,B,C` are terms, we define molecules as follows:
  - `A::B` and `A:B` are **isa** molecules
  - `A[B op C]` with op ∈ {`->`, `->>`, `=>`, `=>>`} is an attribute molecule
- A **positive literal** is either an atom `A` or a molecule `M`
- A **negative** literal is a either a negated atom `not A` or a negated molecule `not B`

F-Logic Programming (3)

- An F-Logic rule is of the form: `H :- B1,...,B1.`

  For positive head-literal `H` and body-literals `B1,...,Bn`.
- Fact: A ground atom or molecule (i.e. rule without a body): `H.`
- Query: A rule without a head:

```
?- B1,..,Bn.
```

F-Logic Example

- Facts (cf. database):

```
bob:empl[name->"Bob";
         age->40;
         affiliation->cs1].
mary:faculty[affiliation->cs1].
cs1:dept[dname->"CS";
         mngr->bob]].
```

- Class information (cf. ontology, database schema):

```
person[name=>string;
       friends=>>person;
       children=>>child(person)].
empl::person[affiliation=>department;
             boss=>empl].
faculty::empl[boss=>(faculty,manager);
              avgSalary->50000].
dept[mngr=>empl].
```

F-Logic Example (cont'd)

- Rule:

```
E[boss->M] :- E:empl, D:dept,
              E[affiliation->D[mngr->M:empl]].
```

Translating RDF to F-Logic

- Translate triples to attribute molecules
- $\langle A,B,C \rangle \mapsto$ `A[B -> C]`
- Axiomatize RDF semantics
  - Add facts for axiomatic triples
  - Add rules for entailment rules
- Treat bNodes as skolem constants
  - "rigid" bNodes
  - anonymous resources

bNodes as Anonymous Resources

- Unnumbered anonymous resource: `_#`
  - Globally unique constant identifier
  - Every occurrence is **fresh** identifier
- Numbered anonymous resource: `_#1, _#2, ...`
  - Resource has **scope**: formula
  - Within scope, different occurrences denote the same resource
- Anonymous resources are **not** existential variables
- Thus, semantics of bNodes cannot be captured completely

Reducing F-Logic to LP

- F-Logic Programming can be reduced to Logic Programming
- Thus, F-Logic is syntactic sugar; it does not add anything in expressiveness
- Reducing terms:
  - Map object IDs to constants
  - Map constructed terms to functions
  - Map variables to variables

  Thus, terms look exactly the same!
- Map atoms to atoms

Reducing F-Logic to LP (2)

- Each type of molecule is mapped to an atom:
  - `A:B` maps to `_member(A,B)`
  - `A::B` maps to `_subclass(A,B)`
  - `A[B->C]` maps to `_svatt(A,B,C)`
  - `A[B->>C]` maps to `_mvatt(A,B,C)`
  - `A[B=>C]` maps to `_svattsig(A,B,C)`
  - `A[B=>>C]` maps to `_mvattsig(A,B,C)`

  Note that the name of the predicates does not really matter.
- You obtain LP rules by simply replacing terms, atoms and molecules as specified.
- Axiomatize the semantics of the F-Logic is-a molecules:

```
_subclass(X,Z) :- _subclass(X,Y),_subclass(Y,Z).
 _member(X,Z) :- _member(X,Y),_subclass(Y,Z).
```

Summary

Required reading

- Michael Kifer: Rules and Ontologies in F-Logic. Reasoning Web 2005: 22-34

Further reading

- Michael Kifer, Georg Lausen, James Wu: Logical Foundations of Object-Oriented and Frame-Based Languages. J. ACM 42(4): 741-843 (1995)
- Guizhen Yang, Michael Kifer: Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. J. Data Semantics 1: 69-97 (2003)
- Guizhen Yang, Michael Kifer: Well-Founded Optimism: Inheritance in Frame-Based Knowledge Bases. CoopIS/DOA/ODBASE 2002: 1013-1032