First Order Logics
Logics and Databases

G. Falquet
Technologies du Web Sémantique

First order logic

Language vocabulary

- constant symbols
- variable symbols
- predicate symbols (with $=$)
- function symbols
- $\wedge$ $\vee$ $\exists$ $\forall$ $\neg$ $\rightarrow$ ( )

Terms: Constant, Variable, Func(T, ..., T)

Atoms: Pred(T, ..., T)

Formulae: Atom | F$\wedge$F | F$\vee$F | $\neg$F | $\exists$ Var F | $\forall$ Var F

Example

constants: a, b, c, d

predicate: knows, $\leq$, $=$

functions: age

formulae

- knows(a, b)
- age(a) $\leq$ age(c)
- $\exists$ x knows(x, c)
- $\forall$ y $\forall$ z (knows(y, z) $\rightarrow$ knows(z, y))
- $\forall$ y (knows(y, c) $\bigvee \exists$ z (age(z) $\leq$ age(y)))

Interpretation

a domain (universe) U

a function from constant symbols to U

$$c \longmapsto c^I$$

a function from predicate symbols to relations

$$p \longmapsto p^I \subseteq U^n,$$

for an n-ary predicate

a function from function symbols to functions

$$f \longmapsto f^I \subseteq U^n \rightarrow U$$

Extension to terms

Ground terms (no variable)

$$(f(t_1, \ldots, t_n))^I = f^I(t_1{}^I, \ldots, t_n{}^I)$$

Terms with variables

we need a value assignment μ for each variable

for a constant: $c^{I,\mu} = c^I$

for a variable: $x^{I,\mu} = \mu(x)$

for a term: $f(t_1, \ldots, t_n)^{I,\mu} = f^I(t_1{}^{I,\mu}, \ldots, t_n{}^{I,\mu})$

Example

$U = \{\text{alice, bob, chris, dora, } 0, 1, \ldots, 150\ \}$

$a^I = \text{alice}, b^I = \text{bob}, \ldots$

$\text{age}^I = \{\text{alice} \mapsto 33, \text{bob} \mapsto 12, \text{chris} \mapsto 44, \text{dora} \mapsto 27\}$

$\text{knows}^I = \{(\text{alice, bob}), (\text{bob, alice}), (\text{dora, chris})\}$

$\text{age}(x)^{I, [x=\text{bob}]} = \text{age}^I(\text{bob}) = 12$

Satisfaction of a formula

$I \models P(t_1, \ldots, t_n)[\mu]$ iff

$$(t_1^{I,\mu}, \ldots, t_n^{I,\mu}) \in P^I$$

$I \models \exists x\, F\, [\mu]$ iff

there is some $u \in U$ such that

$$I \models F\, [\mu/x{=}u]$$

etc.

Example

$U = \{\text{alice, bob, chris, dora, } 0, 1, \ldots, 150\ \}$

$a^I = \text{alice}, b^I = \text{bob}, \ldots$

$\text{age}^I = \{\text{alice} \mapsto 33, \text{bob} \mapsto 28, \text{chris} \mapsto 11, \text{dora} \mapsto 27\}$

$\text{knows}^I = \{(\text{alice, bob}), (\text{bob, alice}), (\text{dora, bob}), (\text{chris, bob})\}$

$I \models \text{knows}(a, b)$

$I \models \text{knows}(b, x)[x{=}\text{alice}]$

$I \models \forall y \forall z\ (\text{knows}(y, z) \rightarrow (\text{knows}(z, y) \vee \text{age}(y) \leq \text{age}(z)))$

Model and Implication

I is a model of F1, ..., Fk iff

$$I \models F1 \text{ and } \ldots \text{ and } I \models Fk$$

Logical implication, Logical consequence, Entailment

$$F \models G$$
$$\text{iff}$$
$$I \models F \Rightarrow I \models G \text{ (for all I)}$$

the satisfaction of F entails the satisfaction of G

when F is true G is necessarily true

Computability

- Implication is not computable (recursive)
  - there is no algorithm that
    - answers YES if $F \models G$
    - answers NO if $F \not\models G$
- Implication is recursively enumerable
  - there is an algorithm that
    - answers YES iff $F \models G$
    - answers NO or never halts if $F \not\models G$

Proof systems

- to prove $F \models G$ by the application of formal rules
- Gödel completeness thm

Compactness

Φ a set (finite or infinite) of formulae

If $\Phi \models G$
there is a finite subset $\Phi' \subseteq \Phi$ such that $\Phi' \models G$

or

If each finite subset of Φ is satisfiable then Φ is satisfiable

Graph connectedness cannot be expressed in FOL

- Φ a formula that expresses "a and b are connected"
  - $I \models \Phi$ iff a and b are connected
- $S = \{s_i \mid i > 0\}$ (s_i expresses "there is not path shorter than i from a to b")
- Every finite subset of $\Phi \cup S$ is sat.
  - $\Rightarrow \Phi \cup S$ is sat.
  - $\Rightarrow$ contradiction $\Rightarrow \Phi$ does not exist

Logic and databases

- Databases as interpretations (mode theory)
- Databases as logical theories

Datalog

Approach based on model theory: every predicate symbol is associated to a relation (its interpretation)

There is a database with stored relations: the extensional database (EDB)

- A predicate with a relation in EDB is extensionally defined
- A predicate defined by logical rules is a deduced predicate. Its relation belongs to the deduced database (DDB).

- *Goal: compute the deduced database*

Datalog "program"

- Facts
  - atomic formulae
    - friend(bob, max)

- Rules
  - horn clauses: atom :- atom, atom, ..., atom

    grandparent(X, Y) :- parent(X, Z), parent(Z, Y)

    in standard FOL: $\forall X \forall Y \forall Z$ (parent(X, Z), parent(Z, Y))$\rightarrow$grandparent(X, Y)

Predefined predicates

$<, \leq, >, \geq, =, \neq$. are predefined

- their interpretations are infinite relations

A rule is unsafe if it produces an infinite relation

- r(X, Y) :– X > Y

To avoid unsafe rules, every variable must appear in a non-predefined relation

- greaterSalary(X, Y) :– X>Y, Salary (P,X), Salary(Q,Y).

Relation for a rule

A relation whose attributes are the variables appearing in the rule

The join of the relations corresponding to the atoms in the rule body + selections.

(r) cousin(X, Y) :– parent(X, Xp) , parent(Y, Yp) , sibling(Xp, Yp).

==>

Cousin(X, Xp, Y, Yp) = Parent(X, Xp) $\bowtie$ Parent(Y, Yp) $\bowtie$ Sibling(Xp, Yp).

notation: EVAL-RULE(r, Parent, Parent, Sibling).

(r') sibling(X, Y) :– parent(X, Z) , parent(Y, Z) , X ≠ Y.

EVAL-RULE(r', Parent, Parent) =

$$\text{sibling}(X, Y) = \pi_{X,Y}\sigma_{X \neq Y}(\text{Parent}(X, Z) \bowtie \text{Parent}(Y, Z))$$

- There is a straightforward algorithm to transform a rule into a relational expression.

Evaluation non-recursive rules

Definition. p depends on q if there is a rule of the form

p(...) :– ... , q(...) , ...

A predicate that belongs to no cycle in the dependency graph is non-recursive.

Algorithm for non-recursive program

1. Order the rules according to the dependencies
2. Compute the relation of each rule
3. Take the union of the relations corresponding to the same predicate

Properties

- deduced facts (tuples is computed relations) are logical consequences of the rules and initial facts
- deduced relations form a minimal model

Recursive rules

- p(X, Y) :– q(X, Y).
- p(X, Y) :– p(X, Z) , q(Z, Y).

Problem: there is no evaluation order for the rules

Derivation of all possible fact

R_1, ..., R_n, the extensional database
P_1, ..., P_m the relations to compute (deduce)

Define EVAL(p_i, R_1, ..., R_n, P_1, ..., P_m) as
the union of the EVAL-RULE for the rules defining p_i.

Derivation of all possible fact

repeat
  for each P_i
    $P_i \leftarrow$ EVAL(p_i, R_1, ..., R_n, P_1, ..., P_m)
until nothing new is produced

- will necessarily stop because EVAL is monotonic and no new constant is created
- will eventually produce all the deductible facts
- the obtained relations will satisfy the equations

$$P_i = \text{EVAL}(p_i, R_1, ..., R_n, P_1, ..., P_m)$$

- = computes the minimal fixpoint of the equations
- computes the unique minimal model of the rules

Example

- path(X, Y) :– segment(X, Y).
- path(X, Y) :- path(X, Z) , path(Z, Y).

Equation:

$$\text{Path}(X, Y) = \text{Segment}(X, Y) \cup \pi_{X,Y}(\text{Path}(X, Z) \bowtie \text{Path}(Z, Y))$$

Negations

Problem: there is no least fixpoint P.ex.

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

- Two models that are fixpoints

1. P = ∅, Q = {1}, R = {1}.
2. P = {1}, Q = ∅, R = {1}.

Stratification

Technique to choose a "good" minimal fixpoint.

A program is stratified if

- if there is a rule p :– ... ¬q ...
- then there is no path from p to q in the dependency graph (q does not depend on p)

- *S* is a stratification if

  p :– ... ¬q ... ⇒ S(p) > S(q)

  p :– ... q ... ⇒ S(p) ≥ S(q)

- *S* provides an evaluation order for the predicates

Try it!

Datalog Educational System (DES)

http://www.fdi.ucm.es/profesor/fernan/des/