First Order Logic
(Predicate Calculus)

CPS 170

Ronald Parr

First Order Logic

- Propositional logic is very restrictive
  - Can't make global statements about objects in the world
  - Tends to have very large KBs
- First order logic is more expressive
  - Relations, quantification, functions
  - More expensive

First Order Syntax

- Sentences
- Atomic sentence predicate(term)
- Terms – functions, constants, variables
- Connectives
- Quantifiers
- Constants
- Variables

Relations

- Assert relationships between objects
- Examples
  - Loves(Harry, Sally)
  - Between(Canada, US, Mexico)
- Semantics
  - Object and predicate names are mnemonic only
  - Interpretation is imposed from outside

Functions

- Functions are specials cases of relations
- Suppose $R(x_1,x_2,\ldots,x_n,y)$ is such that for every value of $x_1,x_2,\ldots,x_n$ there is a unique y
- Then $R(x_1,x_2,\ldots,x_n)$ can be used as a shorthand for y
  - Crossed(Right_leg_of(Ron), Left_leg_of(Ron))
- Remember that the object identified by a function depends upon the interpretation

Quantification

- For all objects in the world…

$$\forall x \text{Tired}(x)$$

- For at least one object in the world…

$$\exists x \text{Tired}(x)$$

Examples

- Everybody loves somebody

$$\forall x \exists y Loves(x, y)$$

- Everybody loves everybody

$$\forall x \forall y Loves(x, y)$$

- Everybody loves Raymond

$$\forall x Loves(x, Raymond)$$

- Raymond loves everybody

$$\forall x Loves(Raymond, x)$$

What's Missing?

- There are many extensions to first order logic
- Higher order logics permit quantification over predicates:

$$\forall x, y(x = y) \Leftrightarrow (\forall p(p(x) \Leftrightarrow p(y)))$$

- Functional expressions (lambda calculus)
- Uniqueness
- Extensions typically replace a potentially long series of conjuncts with a single expression

Inference

- All rules of inference for propositional logic apply to first order logic
- We need extra rules to handle substitution for quantified variables

$$\text{SUBST}(\{x / Harry, y / Sally\}, Loves(x, y)) = Loves(Harry, Sally)$$

Inference Rules

- Universal Elimination

$$\frac{\forall v \alpha}{SUBST(\{v / g\}, \alpha)}$$

- How to read this:
  - We have a universally quantified variable v in α
  - Can substitute any g for v and α will still be true

Inference Rules

- Existential Elimination

$$\frac{\exists v \alpha}{\text{SUBST}(\{v / k\}, \alpha)}$$

- How to read this:
  - We have a universally quantified variable v in a
  - Can substitute any k for v and α will still be true
  - IMPORTANT: k must be a previously unused constant (*skolem* constant). Why is this OK?

Inference Rules

- Existential Introduction

$$\frac{\alpha}{\text{SUBST}(\{g / v\}, \exists v \alpha)}$$

- How to read this:
  - We know that the sentence α is true
  - Can substitute variable v for any constant g in α and (w/existential quantifier) and α will still be true
  - Why is this OK?

Inference Rules

- Generalized Modus Ponens
- Define a substitution such that:

$$\mathrm{SUBST}(\theta, p_i') = \mathrm{SUBST}(\theta, p_i) \forall i$$

- Then

$$\frac{p_1', p_2', \ldots p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\mathrm{SUBST}(\{\theta / q\})}$$

Generalized Modus Ponens

$$\mathrm{SUBST}(\theta, p_i') = \mathrm{SUBST}(\theta, p_i) \forall i$$

$$\frac{p_1', p_2', \ldots p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\mathrm{SUBST}(\{\theta / q\})}$$

- How to read this:
  - We have an implication which implies q
  - Any consistent substitution of variables on the LHS must yield a valid conclusion on the RHS

Unification

- Substitution is a non-trivial matter
- We need an algorithm unify:

  $$\mathrm{Unify}(p, q) = \theta : \mathrm{Subst}(\theta, p) = \mathrm{Subst}(\theta, q)$$

- Important: Unification replaces variables:

  $$\exists x \mathrm{Loves}(John, x), \exists x \mathrm{Hates}(John, x)$$

Unification Example

$$\forall x Knows(John, x) \Rightarrow Loves(John, x)$$
$$Knows(John, Jane)$$
$$\forall y Knows(y, Leonid)$$
$$\forall y Knows(y, Mother(y))$$
$$\forall x Knows(x, Elizabeth)$$

Note: All unquantified variables are assumed universal from here on.

$$\mathrm{Unify}(Knows(John, x), Knows(John, Jane)) =$$
$$\mathrm{Unify}(Knows(John, x), Knows(y, Leonid)) =$$
$$\mathrm{Unify}(Knows(John, x), Knows(y, Mother(y))) =$$
$$\mathrm{Unify}(Knows(John, x), Knows(x, Elizabeth)) =$$

Most General Unifier

- Unify(Knows(John,x),Knows(y,z))
  - {y/John,x/z}
  - {y/John,x/z,w/Freda}
  - {y/John,x/John,z/John)
- When in doubt, we should always return the most general unifier (MGU)
  - MGU makes least commitment about binding variables to constants

Proof Procedures

- Suppose we have a knowledge base: KB
- We want to prove q
- Forward Chaining
  - Like search: Keep proving new things and adding them to the KB until we are able to prove q
- Backward Chaining
  - Find $p_1 \ldots p_n$ s.t. knowing $p_1 \ldots p_n$ would prove q
  - Recursively try to prove $p_1 \ldots p_n$

Forward Chaining Example

$\forall x Knows(John, x) \Rightarrow Loves(John, x)$

$Knows(John, Jane)$

$\forall y Knows(y, Leonid)$

$\forall y Knows(y, Mother(y))$

$\forall x Knows(x, Elizabeth)$

Forward Chaining

```
Procedure Forward_Chain(KB,p)
If p is in KB then return
Add p to KB
For each (p_1 ^ … ^ p_n=>q) in KB such that for
some i,
Unify(p_i,p)=θ succeeds do
        Find_And_Infer(KB,[p_1,…,p_i-1,p_i+1,…,p_n],q,θ)
end
```

```
Procedure Find_and_Infer(KB,premises,conclusion,θ)
If premises=[] then
        Forward_Chain(KB,Subst(θ,conclusion))
Else for each p' in KB such that
Unify(p',Subst(θ,Head(premises)))=θ_2 do
        Find_And_Infer(KB,Tail(premises),conclusion,[θ,θ_2]))
end
```

Backward Chaining Example

$\forall x Knows(John, x) \Rightarrow Loves(John, x)$

$Knows(John, Jane)$

$\forall y Knows(y, Leonid)$

$\forall y Knows(y, Mother(y))$

$\forall x Knows(x, Elizabeth)$

Backward Chaining

```
Function Back_Chain(KB,q)
        Back_Chain_List(KB,[q],{})


Function Back_Chain_List(KB,qlist,θ)
If qlist=[] then return θ
q<-head(qlist)
For each q_i' in KB such that θ_i<-Unify(q,q_i') succeeds do
        Answers <- Answers + [θ,θ_i]
For each (p_i^…^p_n=>q_i')in KB: θ_i<-Unify(q,q_i') succeeds do
        Answers<- Answers+
                        Back_Chain_List(KB,Subst(q_i,[p_i…p_n]),[θ,θ_i]))
return union of Back_Chain_List(KB,Tail(qlist),θ) for each θ in answers
```

Completeness

$\forall x P(X) \Rightarrow Q(x)$

$\forall x \neg P(X) \Rightarrow R(x)$

$\forall x Q(x) \Rightarrow S(x)$

$\forall x R(x) \Rightarrow S(x)$

$S(A)???$

- Problem: Generalized Modus Ponens not complete
- Goal: A sound **and** complete inference procedure for first order logic

Generalized Resolution

$$\frac{(p_1 \vee \ldots p_j \ldots \vee p_m), (q_1 \vee \ldots q_k \ldots \vee q_n)}{\mathrm{SUBST}(\theta, (p_1 \vee \ldots p_{j-1} \vee p_{j+1} \ldots \vee p_m \vee q_1 \vee \ldots q_{k-1} \vee q_{k+1} \ldots \vee q_n))}$$

- How to read this:
  - Substitution: $\mathrm{Unify}(p_j, \neg q_k) = \theta$
  - If the same term appears in both positive and negative form in two disjunctions, they cancel out when disjunctions are combined

Resolution Properties

- Proof by refutation (asserting negation and resolving to nil) is sound and complete
- Resolution is not complete in a generative sense, only in a testing sense
- This is only part of the job
- To use resolution, we must convert everything to a canonical form

Canonical Form

- Eliminate Implications
- Move negation inwards
- Standardize (apart) variables
- Move quantifiers Left
- Skolemize
- Drop universal quantifiers
- Distribute AND over OR
- Flatten nested conjunctions and disjunctions
- Convert disjunctions to implications (optional)

Resolution Example

$(\neg P(x) \vee Q(x))$
$(P(x) \vee R(x))$
$(\neg Q(x) \vee S(x))$
$(\neg R(x) \vee S(x))$
$S(A)???$

Example on board…

Computational Properties

- Can we enumerate the set of all proofs?
- Can we check if a proof is valid?
- What if no valid proof exists?
- Inference in first order logic is semi-decidable

- Compare with halting problem

Gödel

- How do these soundness and completeness results relate to Gödel's incompleteness theorem?
- Incompleteness applies to mathematical systems
- You need numbers because you need a way of referring to proofs by number