First Order Logic
(Predicate Calculus)

CPS 270
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...
  \( \forall x \text{happy}(x) \)
- For at least one object in the world...
  \( \exists x \text{happy}(x) \)
Examples

• Everybody loves somebody
• Everybody loves everybody
• Everybody loves Raymond
• Raymond loves everybody

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

\[
\text{SUBST} \left( \{ x / \text{Harry, } y / \text{Sally} \}, \text{Loves}(x, y) \right) = \text{Loves(Harry, Sally)}
\]

Inference Rules

• Universal Elimination

\[
\forall \forall \alpha \\
\text{SUBST} \left( \{ v / k \}, \alpha \right)
\]

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

Skolemization within Quantifiers

• Skolemizing with universal quantifier is tricky
• Everybody loves somebody

\[
\forall x \exists y : \text{loves}(x, y)
\]

• With Skolem constants, becomes:

\[
\forall x : \text{loves}(x, \text{object}34752)
\]

• Why is this wrong?
• Need to use skolem functions:

\[
\forall x : \text{loves}(x, \text{personlovedby}(x))
\]
Inference Rules

• Existential Introduction

\[ \alpha \]
\[ \text{SUBST}((g / v), \exists v \alpha) \]

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

Inference Rules

• Generalized Modus Ponens
  • Define a substitution such that:
  \[ \text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i) \forall i \]
  • Then
  \[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]
  \[ \text{SUBST}((\theta / q)) \]

Generalized Modus Ponens

\[ \text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i) \forall i \]
\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]
\[ \text{SUBST}((\theta / q)) \]

• 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:
    
    \[ \text{Unify}(p, q) = \theta : \text{Subst}(\theta, p) = \text{Subst}(\theta, q) \]
  • Important: Unification replaces variables:
    
    \[ \exists x \text{Loves}(John, x), \exists x \text{Hates}(John, x) \]

Unification Example

\[ \forall x \text{Knows}(John, x) \Rightarrow \text{Loves}(John, x) \]
\[ \text{Knows}(John, Jane) \]
\[ \forall y \text{Knows}(y, Leonid) \]
\[ \forall y \text{Knows}(y, Mother(y)) \]
\[ \forall x \text{Knows}(x, Elizabeth) \]

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

\[ \text{Unify}((\text{Knows}(John, x), \text{Knows}(John, Jane))) = \]
\[ \text{Unify}((\text{Knows}(John, x), \text{Knows}(y, Leonid))) = \]
\[ \text{Unify}((\text{Knows}(John, x), \text{Knows}(y, Mother(y)))) = \]
\[ \text{Unify}((\text{Knows}(John, x), \text{Knows}(x, Elizabeth))) = \]

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 p₁…pₙ such that knowing p₁…pₙ would prove q
  – Recursively try to prove p₁…pₙ

Forward Chaining Example

∀xKnows(x, John) ⇒ Loves(John, x)
Knows(John, Jane)
∀yKnows(y, Leonid)
∀yKnows(y, Mother(y))
∀xKnows(x, Elizabeth)

Forward Chaining

Procedure Forward_Chain(KB, p)
If p is in KB then return
Add p to KB
For each (p₁ ^ … ^ pₙ ⇒ q) in KB such that for some i,
  Unify(p, pᵢ) = 0 succeeds do
    Find_And_Infer(KB, [p₁, …, pᵢ−₁, pᵢ₊₁, …, pₙ], q, θ)
  end
Procedure Find_and_Infer(KB, premises, conclusion, θ)
If premises = [] then
  Forward_Chain(KB, Subst(θ, conclusion))
Else for each pᵢ in KB such that
  Unify(pᵢ, Subst(θ, Head(premises))) = θᵢ do
  Find_And_Infer(KB, Tail(premises), conclusion, [θ, θᵢ])
end

Backward Chaining Example

∀xKnows(x, John) ⇒ Loves(John, x)
Knows(John, Jane)
∀yKnows(y, Leonid)
∀yKnows(y, Mother(y))
∀xKnows(x, Elizabeth)

Backward Chaining

Function Back_Chain(KB, q)
  Back_Chain_List(KB, [q], [])

Function Back_Chain_List(KB, q list, θ)
If q list = [] then return θ
If q list is head of q list
  For each qᵢ in KB such that θᵢ • Unify(q, qᵢ) succeeds do
    Answers ← Answers + [θᵢ]
  end
  For each (p₁ ^ … ^ pₙ ⇒ q) in KB
    If Unify(qᵢ, q) succeeds do
      Answers ← Answers
      Back_Chain_List(KB, Subst(q, [p₁, …, pᵢ−₁, pᵢ₊₁, …, pₙ]), [θ, θᵢ])
    end
return union of Back_Chain_List(KB, Tail(q list), θ) for each θ in answers

Completeness

∀xP(x) ⇒ Q(x)
∀x¬P(X) ⇒ R(x)
∀xQ(x) ⇒ S(x)
∀xR(x) ⇒ S(x)
S(A)???

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

\[(p_1 \lor \cdots \lor p_n) \land (q_1 \lor \cdots \lor q_m)\]

\[\text{SUBST}(\theta, (p_1 \lor \cdots \lor p_n) \land (q_1 \lor \cdots \lor q_m))\]

- How to read this:
  - Substitution: \(\text{Unify}(p, \neg q) = \theta\)
  - 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 \(\land\) over \(\lor\)
- Flatten nested conjunctions and disjunctions
- Convert disjunctions to implications (optional)

Resolution Example

\[\neg P(x) \lor Q(x)\]
\[P(x) \lor R(x)\]
\[\neg Q(x) \lor S(x)\]
\[\neg R(x) \lor S(x)\]

\[S(A)??\]

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