Inference in first-order logic

Russell and Norvig Chapter 9

Outline
- Reducing first-order inference to propositional inference
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution

November 3, 2014

FOL to PL
- First order inference can be done by converting the knowledge base to PL and using propositional inference.
  - How to convert universal quantifiers?
    - Replace variable by ground term.
  - How to convert existential quantifiers?
    - Skolemization.

November 3, 2014

Universal Instantiation (UI)
Every instantiation of a universally quantified sentence is entailed by it:

\[ \forall v \alpha \Rightarrow \text{Subst}([v/g], \alpha) \]

for any variable \( v \) and ground term \( g \)

E.g., \( \forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \) yields:

\[
\begin{align*}
\text{King}(\text{John}) \land \text{Greedy}(\text{John}) & \Rightarrow \text{Evil}(\text{John}) \\
\text{King}((\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John}))) & \Rightarrow \text{Evil}(\text{Father}(\text{John}))
\end{align*}
\]

November 3, 2014

Existential Instantiation (EI)
For any sentence \( \alpha \), variable \( v \), and constant symbol \( k \) that does not appear elsewhere in the knowledge base:

\[ \exists v \alpha \Rightarrow \text{Subst}([v/k], \alpha) \]

E.g., \( \exists x \text{Crown}(x) \land \text{OnHead}(x, \text{John}) \) yields:

\[
\text{Crown}(C_1) \land \text{OnHead}(C_1, \text{John})
\]

provided \( C_1 \) is a new constant symbol, called a Skolem constant.

November 3, 2014

EI versus UI
- UI can be applied several times to add new sentences; the new KB is logically equivalent to the old.
- EI can be applied once to replace the existential sentence.

November 3, 2014
Reduction to propositional inference

- Suppose the KB contains just the following:
  \( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \)
  King(John)
  Greedy(John)
  Brother(Richard, John)

- Instantiating the universal sentence in all possible ways:
  King(John) \land Greedy(John) \Rightarrow Evil(John)
  King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)

- The new KB is propositionalized

Reduction (cont)

- CLAIM: A ground sentence is entailed by the new KB if entailed by the original KB.
- CLAIM: Every FOL KB can be propositionalized so as to preserve entailment.
- IDEA: propositionalize KB and query, apply resolution, return result
- PROBLEM: with function symbols, there are infinitely many ground terms,
  e.g., Father(Father(Father(John)))
  Instantiating the universal sentence in all possible ways:

  The question of entailment for FOL is semi-decidable: no algorithm
  exists that says no to every nonentailed sentence.

Reduction (cont)

- THEOREM: Herbrand (1930). If a sentence \( \alpha \) is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB.
- IDEA: For \( n \geq 0 \) to do:
  - create a propositional KB by instantiating with depth-\( n \) terms
  - see if \( \alpha \) is entailed by this KB
- PROBLEM: works if \( \alpha \) is entailed, does not halt if \( \alpha \) is not entailed
- \( \alpha \) is entailed if \( \alpha \) is entailed.
- With \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations!

Is there another way?

- Instead of translating the knowledge base to PL, we can make the inference rules work in FOL.
- For example, given
  \( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \)
  King(John)
  \( \forall y \ Greedy(y) \Rightarrow Evil(y) \)
  It is intuitively clear that we can substitute \( \{x/\text{John}, y/\text{John}\} \) and obtain Evil(John)

Unification

- We can make the inference if we can find a substitution such that King(x) and Greedy(x) match King(John) and Greedy(y)
- \( \{x/\text{John}, y/\text{John}\} \) works
- Uniify(\( \alpha, \beta \)) = 0 if Subst(\( \alpha, \beta \)) = Subst(\( \alpha, \beta \))


<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>Subst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John,x)</td>
<td>Knows(John, Jane)</td>
<td>Knows(John, Jane)</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y, OJ)</td>
<td>Knows(y, OJ)</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y, Mother(y))</td>
<td>Knows(y, Mother(y))</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x, OJ)</td>
<td>Knows(x, OJ)</td>
</tr>
</tbody>
</table>

Fall 2013
Unification

Unify(\alpha, \beta) = \emptyset if Subst(\emptyset, \alpha) = Subst(\emptyset, \beta)

<table>
<thead>
<tr>
<th>\alpha</th>
<th>\beta</th>
<th>Subst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, x)</td>
<td>Knows(John, Jane)</td>
<td>{x/Jane}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, OJ)</td>
<td>{y/OJ, y/John}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mother(y))</td>
<td>{y/John, x/Mother(John)}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(x, OJ)</td>
<td>{fail}</td>
</tr>
</tbody>
</table>

Unifiers of Knows(John, x) and Knows(y, z) are
\{y/John, x/z\} or \{y/John, x/John, z/John\}

The first unifier is more general than the second.

There is a single most general unifier (MGU) that is unique up to renaming of variables.

MGU = \{y/John, x/z\}

The unification algorithm

Function Unify(x, y, \emptyset) returns a substitution to make x and y identical

\[ x \] a variable, constant, list, or compound
\[ y \] a variable, constant, list, or compound
\[
\text{if } x = y \text{ then return } \emptyset
\]
\[
\text{else if } x \text{ a variable then return } \text{Unify-Vars}(x, y, \emptyset)
\]
\[
\text{else if } \text{Variable}(x) \text{ then return } \text{Unify-Vars}(x, y, \emptyset)
\]
\[
\text{else if } \text{Compound}(x) \text{ and Compound}(y) \text{ then return } \text{Unify-Vars}(x, y, \emptyset)
\]
\[
\text{else if } \text{List}(x) \text{ and List}(y) \text{ then return } \text{Unify-List}(x, y, \emptyset)
\]
\[
\text{else if } \text{List}(x) \text{ and List}(y) \text{ then return } \text{Unify-List}(x, y, \emptyset)
\]
\[
\text{else if } \text{Compound}(x) \text{ and Compound}(y) \text{ then return } \text{Unify-Compound}(x, y, \emptyset)
\]
\[
\text{else if } \text{List}(x) \text{ and List}(y) \text{ then return } \text{Unify-List}(x, y, \emptyset)
\]
\[
\text{else if } \text{Compound}(x) \text{ and Compound}(y) \text{ then return } \text{Unify-Compound}(x, y, \emptyset)
\]
\[
\text{else return } \text{fail}
\]
Generalized Modus Ponens (GMP)

Suppose that $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$ for all $i$ then:

$p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)$

$\text{Subst}(\theta, q)$

$p_1'$ is King($\text{John}$)
$p_2'$ is Greedy($y$)
$q$ is Evil($x$)

- All variables assumed universally quantified.

Example

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove that Col. West is a criminal

Example

... it is a crime for an American to sell weapons to hostile nations:

$\text{American}(x) \land \text{Weapon}(y) \land Sells(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

Example

... it is a crime for an American to sell weapons to hostile nations:

$\text{American}(x) \land \text{Weapon}(y) \land Sells(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

Example

... it is a crime for an American to sell weapons to hostile nations:

$\text{American}(x) \land \text{Weapon}(y) \land Sells(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

Example

... it is a crime for an American to sell weapons to hostile nations:

$\text{American}(x) \land \text{Weapon}(y) \land Sells(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

Nono has some missiles, i.e., $\exists x \text{Owns}(\text{Nono},x) \land \text{Missile}(x)$.
Example

... it is a crime for an American to sell weapons to hostile nations:
American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) → Criminal(x)
Nono ... has some missiles, i.e., 3x Owns(Nono,x) ∧ Missile(x)
... all of its missiles were sold to it by Colonel West
Missile(x) ∧ Owns(Nono,x) → Sells(West,x,Nono)
Missiles are weapons:
Missile(x) → Weapon(x)

An enemy of America counts as "hostile":
Enemy(x,America) → Hostile(x)

November 3, 2014
Example

... it is a crime for an American to sell weapons to hostile nations:
American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

NoNo ... has some missiles, i.e., ∃x Owns(NoNo,x) ∧ Missile(x):

Owns(NoNo,M1) ∧ Missile(M1)

... all of its missiles were sold to it by Colonel West
Missile(x) ∧ Owns(NoNo,x) ⇒ Sells(West,x,NoNo)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile":

Enemy(x,America) ⇒ Hostile(x)

West, who is American ...

American(West)

Enemy(NoNo,America)

November 3, 2014

Forward chaining algorithm

Function FOL-FC: Ax(E, R, A) returns a substitution or false
repeat until x is empty
for each sentence r in KB do
if r is not a variable then return false
if r is a sentence in the form: p₁ ... pₙ ψ where ψ is a standard-parse
if there are no collisions between p₁, ..., pₙ, ψ and x
if ψ is a standard-parse of a sentence already in KB then add ψ to KB
if ψ is a standard-parse of a sentence already in KB then return false
add new to KB
return false
Forward chaining example

Forward chaining example

Forward chaining for FOL

Backward chaining

Backward chaining example
Backward chaining example

\[
\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)
\]

Backward chaining example

\[
\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})
\]

Backward chaining algorithm

```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
return FOL-BC-OR(KB, query, {})

generator FOL-BC-OR(KB, goal, θ) yields a substitution
for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs, rhs) ← STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ' in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
        yield θ'

else do
    for each θ in FOL-BC-AND(KB, goal, UNIFY(lhs, rhs, θ)) do
        yield θ
```

There may be multiple relevant substitutions, so the functions are generators.
Properties of backward chaining

- Depth-first recursive proof search
- Space is linear in size of proof.
- Incomplete due to infinite loops
  - Fixable
- Inefficient due to repeated subgoals
  - Fixable by caching previous results (extra space!!)
- Widely used for logic programming: problem solving by inference.
  - Example: Prolog

Logic programming: Prolog

- BASIS: backward chaining with Horn clauses + bells & whistles
- Program = set of clauses of the form
  \[ \text{head} :- \text{literal}_1, \ldots, \text{literal}_n \]
  - Example: Prolog

Prolog is a "declarative" language

- Clauses are statements about what is true about a problem, instead of instructions how to accomplish the solution.
- The Prolog system uses the clauses to work out how to accomplish the solution by searching through the space of possible solutions.

Example

- Prolog program consists of facts and rules.
  - animal(lion).
  - animal(sparrow).
  - hasfeathers(sparrow).
  - bird(X) :- animal(X), hasfeathers(X).
- "Run" by asking questions or queries. Or (using logic terminology) by setting a goal for Prolog to try to prove:
  \[ ?- \text{bird(sparrow)}. \]
- Or to find a value of a variable that makes it true:
  \[ ?- \text{bird(What)}. \]
  - What = sparrow

Example

- Appending two lists to produce a third:
  - append([], Y, Y).
  - append([X|L], Y, [X|Z]) :- append(L, Y, Z).
- query: append(A, B, [1, 2]).
- answers:
  - A=[]  B=[1,2]
  - A=[1,2]  B=[]

Example

- ancestor(X, Y) :- parent(X, Y).
  - (X is an ancestor of Y if X is a parent of Y.)
- ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
  - (X is an ancestor of Y if X is a parent of an ancestor of Y.)
Permutations

- Permutation(X, Y) is true whenever Y is a permutation of X.

Conversion to CNF

- Need to convert KB to CNF, eliminating existential quantifiers
- Consider the sentence: ∀x Loves(x, y)

Resolution

- Full first-order version:

Conversion to CNF

- Standardize variables: each quantifier should use a different one:

Conversion to CNF

- Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables: ∀x (Animal(F(x)) ⇒ Loves(x, y)) ⇒ Loves(Animal(x), y)

Conversion to CNF

- Distribute v over ∧:

Conversion to CNF

- Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables: ∀x (Animal(F(x)) ⇒ Loves(x, y)) ⇒ Loves(Animal(x), y)

Conversion to CNF

- Standardize variables: each quantifier should use a different one: ∀x (Animal(F(x)) ⇒ Loves(x, y)) ⇒ Loves(Animal(x), y)

Conversion to CNF

- Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables: ∀x (Animal(F(x)) ⇒ Loves(x, y)) ⇒ Loves(Animal(x), y)
Resolution proof

Connection with CSP

Theorem provers

- Theorem prover – a system that does full first order logic inference (using resolution)
- Uses:
  - Prove mathematical theorems (known successes!)
  - Hardware/software verification
    - Verify that circuit/software produces correct output for all possible inputs (RSA algorithm was verified this way)