Inference in first-order logic

Chapter 9

Outline

• Reducing first-order inference to propositional inference
• Unification
• Generalized Modus Ponens
• Forward chaining
• Backward chaining
• Resolution
Inference with Quantifiers

- **Universal Instantiation:**
  - Given $\forall X \text{ person}(X) \Rightarrow \text{likes}(X, \text{sun})$
  - Infer $\text{person}(\text{john}) \Rightarrow \text{likes}(\text{john}, \text{sun})$

- **Existential Instantiation:**
  - Given $\exists x \text{ likes}(x, \text{chocolate})$
  - Infer: $\text{likes}(\text{S1}, \text{chocolate})$
  - $\text{S1}$ is a “Skolem Constant” that is not found anywhere else in the KB and refers to (one of) the individuals that likes sun.

Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

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

for any variable $v$ and ground term $g$

- E.g., $\forall x \text{ King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields:
  - $\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
  - $\text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
  - $\text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$
  - ...
Existential instantiation (EI)

• For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\exists v \alpha$$
$$\text{Subst}\{\{v/k\}, \alpha\}$$

• E.g., $\exists x \ Crown(x) \land OnHead(x, John)$ yields:

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

provided $C_1$ is a new constant symbol, called a Skolem constant

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, we have:

  King(John) $\land$ Greedy(John) $\Rightarrow$ Evil(John)

  King(Richard) $\land$ Greedy(Richard) $\Rightarrow$ Evil(Richard)

  King(John)

  Greedy(John)

  Brother(Richard, John)

• The new KB is propositionalized: proposition symbols are

  King(John), Greedy(John), Evil(John), King(Richard), etc.
Reduction contd.

• Every FOL KB can be propositionalized so as to preserve entailment

• (A ground sentence is entailed by new KB iff entailed by original KB)

• 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)))

Reduction contd.

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 $n = 0$ to $\infty$ 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, loops if $\alpha$ is not entailed

Theorem: Turing (1936), Church (1936) Entailment for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)
Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

- E.g., from:
  \[ \forall x \ King(x) \land Greedy(x) \Rightarrow \text{Evil}(x) \]
  \[ \text{King}(\text{John}) \]
  \[ \forall y \ Greedy(y) \]
  \[ \text{Brother}((\text{Richard}, \text{John}) \]

- it seems obvious that \( \text{Evil}(\text{John}) \), but propositionalization produces lots of facts such as \( \text{Greedy}(\text{Richard}) \) that are irrelevant

- With \( p \, k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations.

Unification

- We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \)

\[ \theta = \{ x/\text{John}, y/\text{John} \} \text{ works} \]

- \( \text{Unify}(\alpha, \beta) = \theta \) if \( \alpha \theta = \beta \theta \)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>\theta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(\text{John},x)</td>
<td>Knows(\text{John},\text{Jane})</td>
<td></td>
</tr>
<tr>
<td>Knows(\text{John},x)</td>
<td>Knows(\text{y},OJ)</td>
<td></td>
</tr>
<tr>
<td>Knows(\text{John},x)</td>
<td>Knows(\text{y},\text{Mother(y)})</td>
<td></td>
</tr>
<tr>
<td>Knows(\text{John},x)</td>
<td>Knows(\text{x},OJ)</td>
<td></td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g.,
  \[ \text{Knows}(z_{17},\text{OJ}) \]
Unification

- We can get the inference immediately if we can find a substitution $\theta$ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

$\theta = \{x/John,y/John\}$ works

- $Unify(\alpha,\beta) = \theta$ if $\alpha\theta = \beta\theta$

\[
\begin{array}{ccc}
\text{p} & \text{q} & \theta \\
Knows(John,x) & Knows(John,Jane) & x/Jane} \\
Knows(John,x) & Knows(y,OJ) & x/OJ,y/John} \\
Knows(John,x) & Knows(y,Mother(y)) & \\
Knows(John,x) & Knows(x,OJ) & \\
\end{array}
\]

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17},OJ$)
Unification

- We can get the inference immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{OJ})$</td>
<td>${x/\text{OJ}, y/\text{John}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td>${y/\text{John}, x/\text{Mother}(\text{John})}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{OJ})$</td>
<td>${\text{fail}}$</td>
</tr>
</tbody>
</table>

- Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$
More on Standardizing apart

- `knows(john,X)`.
- `knows(X,elizabeth)`.
- These ought to unify, since john knows everyone, and everyone knows elizabeth.
- Rename variables to avoid such name clashes

  **Note:**
  
  All `X p(X) == all Y p(Y)`
  
  All `X (p(X) ^ q(X)) == All X p(X) ^ All Y p(Y)`

Unification

- To unify `Knows(John,x)` and `Knows(y,z)`,
  \[ \theta = \{ y/John, x/z \} \text{ or } \theta = \{ y/John, x/John, z/John \} \]

  - 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.

  \[ \text{MGU} = \{ y/John, x/z \} \]
Generalized Modus Ponens

- This is a general inference rule for FOL that does not require instantiation
- GMP “lifts” MP from propositional to first-order logic
- Key advantage of lifted inference rules over propositionalization is that they make only substitutions which are required to allow particular inferences to proceed

Example knowledge base

- 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 knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]

Nono ... has some missiles, i.e., \( \exists x \text{ Owns}(\text{Nono},x) \land \text{Missile}(x) \):
\[ \text{Owns}(\text{Nono},M_1) \land \text{Missile}(M_1) \]

... all of its missiles were sold to it by Colonel West
\[ \text{Missile}(x) \land \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono}) \]

Missiles are weapons:
\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

An enemy of America counts as "hostile":
\[ \text{Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x) \]

West, who is American ...
\[ \text{American}(\text{West}) \]

The country Nono, an enemy of America ...
\[ \text{Enemy}(\text{Nono},\text{America}) \]

Another Example

• \( \forall x,y,z \text{ parent}(x,y) \land \text{parent}(y,z) \Rightarrow \text{grandparent}(x,z) \)

• parent(james, john), parent(james, richard), parent(harry, james)

• We can derive:
  – Grandparent(harry, john), bindings: \{x/harry, y/james, z/john\}
  – Grandparent(harry, richard), bindings: \{x/harry, y/james, z/richard\}
Forward chaining algorithm

function FOL-FC-Ask(KB, α) returns a substitution or false
    repeat until new is empty
        new ← {}
        for each sentence r in KB do
            (p₁ ∧ ... ∧ pₙ ⇒ q) ← STANDARDIZE-APART(r)
            for each θ such that (p₁ ∧ ... ∧ pₙ)θ = (p₁' ∧ ... ∧ pₙ')θ
                for some p₁', ..., pₙ' in KB
                    q' ← SUBST(θ, q)
                    if q' is not a renaming of a sentence already in KB or new then do
                        add q' to new
                        φ ← UNIFY(q', α)
                        if φ is not fail then return φ
                    add new to KB
        return false

Forward chaining proof

American (New)  Missile (M1)  Own (Noise, M1)  Enemy (Nova, America)
Forward chaining proof
Properties of forward chaining

- Sound and complete for first-order definite clauses
- **Datalog** = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if $\alpha$ is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable
- Forward chaining is widely used in deductive databases

Backward chaining algorithm

```
function FOL-BC-Ask(KB, goals, $\theta$) returns a set of substitutions
inputs: KB, a knowledge base
        goals, a list of conjuncts forming a query
        $\theta$, the current substitution, initially the empty substitution {} 
local variables: ans, a set of substitutions, initially empty 
if goals is empty then return {} 
q' ← Subst($\theta$, First(goals)) 
for each r in KB where Standardize-Apart(r) = (p₁ \land \ldots \land pₙ \Rightarrow q) and $\theta'$ ← Unify(q, q') succeeds 
    ans ← FOL-BC-Ask(KB, p₁, \ldots, pₙ|Rest(goals)|, Compose($\theta$, $\theta'$)) \cup ans 
return ans
```
Backward chaining example

Backward chaining example

\begin{itemize}
\item American
\item Weapon
\item Sell
\item Missile
\end{itemize}
Backward chaining example

Backward chaining example
Backward chaining example
Backward chaining example

Backward chaining example
Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - ⇒ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - ⇒ fix using caching of previous results (extra space)
- Widely used for logic programming

Logic programming: Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques ⇒ 60 million LIPS
- Program = set of clauses = head :- literal₁, ..., literalₙ.
  criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., \( \text{X is Y*Z+3} \)
- Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
  - e.g., given \( \text{alive(X)} : \text{not dead(X)} \).
  - \( \text{alive(joe)} \) succeeds if \( \text{dead(joe)} \) fails
In full first-order logic, you would have to be able to infer "~neighbor(canada,india)" for "neighbor(canada,india)" to be false.

Resolution: brief summary

- Full first-order version:
  \[
  \frac{\ell_1 \lor \cdots \lor \ell_k, \ m_1 \lor \cdots \lor m_n}{(\ell_1 \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_n) \theta}
  \]
  where \( \text{Unify}(\ell_i, \neg m_j) = \theta \).

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,
  \[
  \frac{\neg \text{Rich}(x) \lor \text{Unhappy}(x)}{\text{Rich}(\text{Ken})}
  \]
  with \( \theta = \{x/\text{Ken}\} \)

- Apply resolution steps to CNF(KB ∧ ¬α); complete for FOL
Conversion to CNF

• Everyone who loves all animals is loved by someone:
  \( \forall x [\forall y \text{Animal}(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y Loves(y,x)] \)

• 1. Eliminate biconditionals and implications
  \( \forall x [\neg\forall y \neg\text{Animal}(y) \lor Loves(x,y)] \lor [\exists y Loves(y,x)] \)

• 2. Move \( \neg \) inwards:
  \( \neg\forall x p \equiv \exists x \neg p, \quad \neg \exists x p \equiv \forall x \neg p \)
  \( \forall x [\exists y \neg(\neg\text{Animal}(y) \lor Loves(x,y))] \lor [\exists y Loves(y,x)] \)
  \( \forall x [\exists y \neg\text{Animal}(y) \land \neg\text{Loves}(x,y)] \lor [\exists y Loves(y,x)] \)
  \( \forall x [\exists y \text{Animal}(y) \land \neg\text{Loves}(x,y)] \lor [\exists y Loves(y,x)] \)

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one
  \( \forall x [\exists y \text{Animal}(y) \land \neg\text{Loves}(x,y)] \lor [\exists z Loves(z,x)] \)

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing
   universally quantified variables:
   \( \forall x [\text{Animal}(F(x)) \land \neg\text{Loves}(x,F(x))] \lor Loves(G(x),x) \)

5. Drop universal quantifiers:
   \( [\text{Animal}(F(x)) \land \neg\text{Loves}(x,F(x))] \lor Loves(G(x),x) \)

6. Distribute \( \lor \) over \( \land \):
   \( [\text{Animal}(F(x)) \lor Loves(G(x),x)] \land [\neg\text{Loves}(x,F(x)) \lor Loves(G(x),x)] \)
Resolution proof: definite clauses

- Resolution is complete. If you don’t want to take this on faith, study pp. 300-303
- Strategies (heuristics) for efficient resolution exist, see details in text if interested
  - Unit preference. If a clause has only one literal, use it first.
  - Set of support. Identify “useful” rules and ignore the rest. (p. 305)
  - Input resolution. Intermediately generated sentences can only be combined with original inputs or original rules.
  - Subsumption. Prune unnecessary facts from the database.
Inference Methods (Review)

- Unification (prerequisite)
- Forward Chaining
  - Production Systems
- Backward Chaining
  - Logic Programming (Prolog)
- Resolution
  - Transform to CNF
  - Generalization of Prop. Logic resolution

Pages to skip/skim

- Storage and Retrieval (pp. 278-279)
- Efficient forward chaining (starts p. 283) through Irrelevant facts (ends p. 287)
- Efficient implementation of logic programs (starts p. 290) through Constraint logic programming (ends p. 295)
- Completeness of resolution (pp. 300-303)
- Extending Prolog (pp. 307-308)