Artificial Intelligence

Topic 14

Inference in first-order logic

Reading: Russell and Norvig, Chapter 9

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution

A brief history of reasoning

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>450B.C.</td>
<td>Stoics</td>
<td>propositional logic, inference (maybe)</td>
</tr>
<tr>
<td>322B.C.</td>
<td>Aristotle</td>
<td>&quot;syllogisms&quot; (inference rules), quantifiers</td>
</tr>
<tr>
<td>1565</td>
<td>Cardano</td>
<td>probability theory (propositional logic + uncertainty)</td>
</tr>
<tr>
<td>1847</td>
<td>Boole</td>
<td>propositional logic (again)</td>
</tr>
<tr>
<td>1879</td>
<td>Frege</td>
<td>first-order logic</td>
</tr>
<tr>
<td>1922</td>
<td>Wittgenstein</td>
<td>proof by truth tables</td>
</tr>
<tr>
<td>1930</td>
<td>Gödel</td>
<td>∃ complete algorithm for FOL</td>
</tr>
<tr>
<td>1930</td>
<td>Herbrand</td>
<td>complete algorithm for FOL (reduce to propositional)</td>
</tr>
<tr>
<td>1931</td>
<td>Gödel</td>
<td>¬∃ complete algorithm for arithmetic</td>
</tr>
<tr>
<td>1960</td>
<td>Davis/Putnam</td>
<td>&quot;practical&quot; algorithm for propositional logic</td>
</tr>
<tr>
<td>1965</td>
<td>Robinson</td>
<td>&quot;practical&quot; algorithm for FOL—resolution</td>
</tr>
</tbody>
</table>

Universal instantiation (UI)

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

\[ \forall v \alpha \]

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

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:

$$\exists v \, \alpha$$

Subst($\{v/k\}, \alpha$)

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

$\text{Crown}(\mathcal{C}_1) \land \text{OnHead}(\mathcal{C}_1, \text{John})$

provided $\mathcal{C}_1$ is a new constant symbol, called a *Skolem constant*

Another example: from $\exists x \, d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided $e$ is a new constant symbol

---

**Existential instantiation contd.**

EI 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; the new KB is *not* equivalent to the old, but is satisfiable iff the old KB was satisfiable

---

**Reduction to propositional inference**

Suppose the KB contains just the following:

$$\forall x \, \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

Instantiating the universal sentence in all possible ways, we have

$$\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$

$$\text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

$$\text{King}(\text{John})$$

$$\text{Greedy}(\text{John})$$

$\text{Brother}(\text{Richard}, \text{John})$

The new KB is *propositionalized*: proposition symbols are

$\text{King}(\text{John})$, $\text{Greedy}(\text{John})$, $\text{Evil}(\text{John})$, $\text{King}(\text{Richard})$ etc.

---

**Reduction contd.**

Claim: a ground sentence is entailed by new KB iff entailed by 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., $\text{Father}(\text{Father}(\text{Father}(\text{John})))$

Theorem: Herbrand (1930). If a sentence $\alpha$ is entailed by an FOL KB, it is entailed by a *finite* subset of the propositional KB

Idea: For $n = 0 \to \infty$: create a propositional KB by instantiating with depth-$n$ terms and 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 in FOL is *semidecidable*
Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences. E.g., from

\[
\forall x \: \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)
\]

\[
\text{King}(\text{John})
\]

\[
\forall y \: \text{Greedy}(y)
\]

\[
\text{Brother}(\text{Richard}, \text{John})
\]

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.

With function symbols, it gets much much worse!

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

\[\text{UNIFY}(\alpha, \beta) = \theta \text{ if } \alpha\theta = \beta\theta\]

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

\[
\begin{array}{c|c|c}
 p & q & \theta \\
\hline
 \text{Knows}(\text{John}, x) & \text{Knows}(\text{John}, \text{Jane}) & \{x/\text{Jane}\} \\
 \text{Knows}(\text{John}, x) & \text{Knows}(y, \text{OJ}) & \{x/\text{OJ}, y/\text{John}\} \\
 \text{Knows}(\text{John}, x) & \text{Knows}(y, \text{Mother}(y)) & \\
 \text{Knows}(\text{John}, x) & \text{Knows}(x, \text{OJ}) & \\
\end{array}
\]
**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>${x/\text{OJ}, y/\text{John}}$</td>
</tr>
</tbody>
</table>

<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>fail</td>
</tr>
</tbody>
</table>

**Generalized Modus Ponens (GMP)**

\[
p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)_{\theta} \quad \text{where} \quad p_i' \theta = p_i \theta \text{ for all } i \]

$p_1'$ is $\text{King}(\text{John})$\hfill $p_1$ is $\text{King}(x)$$p_2'$ is $\text{Greedy}(y)$\hfill $p_2$ is $\text{Greedy}(x)$$\theta = \{x/\text{John}, y/\text{John}\}$\hfill $q$ is $\text{Evil}(x)$$q \theta$ is $\text{Evil}(\text{John})$

GMP used with KB of definite clauses (exactly one positive literal)\hfill All variables assumed universally quantified

**Soundness of GMP**

Need to show that \[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q)_{\theta} = q \theta \]

provided that $p_i \theta = p_i$ for all $i$

Lemma: For any definite clause $p_i$, we have $p_i \models p_i \theta$ by UI

1. $(p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)_{\theta} = (p_1 \theta \land \ldots \land p_n \theta \Rightarrow q \theta)$
2. $p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' = p_1' \theta \land \ldots \land p_n' \theta$
3. From 1 and 2, $q \theta$ follows by ordinary Modus Ponens
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

Example knowledge base contd.

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
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., \[ \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \]
\[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]
... all of its missiles were sold to it by Colonel West
\[ \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]
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 ...
\[ \text{Enemy}(\text{Nono}, \text{America}) \]

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., \[ \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \]
\[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]
... all of its missiles were sold to it by Colonel West
\[ \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]
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 ...
\[ \text{Enemy}(\text{Nono}, \text{America}) \]

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
        if r ← STANDARDIZE-APART(r)
            for each θ such that \( (p_1 \land \ldots \land p_n) \theta = (p'_1 \land \ldots \land p'_m) \theta \)
                for some \( p'_1, \ldots, p'_m \) in KB
                    \( q' \leftarrow \text{SUBST}(\theta, q) \)
            if \( q' \) is not a renaming of a sentence already in KB or new then
                do
                    add \( q' \) to new
                    \( \phi \leftarrow \text{UNIFY}(q', \alpha) \)
                    if \( \phi \) is not fail then return \( \phi \)
                add new to KB
            return false
Forward chaining proof

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)

Chapter 9

Forward chaining proof

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)

Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

Datalog = first-order definite clauses + no functions (e.g., crime KB)
FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

May not terminate in general if $\alpha$ is not entailed
This is unavoidable: entailment with definite clauses is semidecidable
Efficiency of forward chaining

Simple observation: no need to match a rule on iteration $k$ if a premise wasn’t added on iteration $k - 1$

$\Rightarrow$ match each rule whose premise contains a newly added literal

Matching itself can be expensive

Database indexing allows $O(1)$ retrieval of known facts e.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

Hard matching example

$\text{Diff}(\text{wa}, \text{nt}) \land \text{Diff}(\text{wa}, \text{sa}) \land$

$\text{Diff}(\text{nt}, \text{q}) \land \text{Diff}(\text{nt}, \text{sa}) \land$

$\text{Diff}(\text{q}, \text{nsw}) \land \text{Diff}(\text{q}, \text{sa}) \land$

$\text{Diff}(\text{nsw}, \text{v}) \land \text{Diff}(\text{nsw}, \text{sa}) \land$

$\text{Diff}(\text{v}, \text{sa}) \Rightarrow \text{Colorable()}$

$\text{Diff}(\text{Red}, \text{Blue}) \Rightarrow \text{Colorable()}$

$\text{Diff}(\text{Green}, \text{Red}) \Rightarrow \text{Colorable()}$

$\text{Diff}(\text{Blue}, \text{Red}) \Rightarrow \text{Colorable()}$

$\text{Colorable()}$ is inferred iff the CSP has a solution

CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining algorithm

function $\text{FOL-BC-Ask}(KB, \text{goals}, \theta)$ returns a set of substitutions

inputs: $KB$, a knowledge base $\text{goals}$, a list of conjuncts forming a query ($\theta$ already applied) $\theta'$, the current substitution, initially the empty substitution $\{\}$

local variables: $\text{answers}$, a set of substitutions, initially empty

if $\text{goals}$ is empty then return $\theta'$

$\theta' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\text{goals}))$

for each sentence $r$ in $KB$

$\text{where STANDARDIZE-APART}(r) = \{p_1 \land \ldots \land p_k \Rightarrow q\}$

and $\theta'' \leftarrow \text{UNIFY}(q, \theta')$ succeeds

$\text{new_goals} = \{p_1, \ldots, p_k, \text{REST}(\text{goals})\}$

$\text{answers} \leftarrow \text{FOL-BC-Ask}(KB, \text{new_goals}, \text{COMPOSE}(\theta', \theta)) \cup$

return $\text{answers}$

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 (without improvements!) for logic programming

Logic programming

Sound bite: computation as inference on logical KBs

Logic programming
1. Identify problem
2. Assemble information
3. Tea break
4. Encode information in KB
5. Encode problem instance as facts
6. Ask queries
7. Find false facts

Ordinary programming
1. Identify problem
2. Assemble information
3. Figure out solution
4. Program solution
5. Encode problem instance as data
6. Apply program to data
7. Debug procedural errors

Should be easier to debug Capital(NewYork,US) than $x := x + 2$ !
Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ approaching a billion LIPs
Program = set of clauses = head :- literal₁, ..., literalₙ.

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).

Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
Closed-world assumption ("negation as failure")
e.g., given alive(X) :- not dead(X).
alive(joe) succeeds if dead(joe) fails

Resolution: brief summary

Full first-order version:

\[ \ell₁ \lor \cdots \lor \ellₖ \land m₁ \lor \cdots \lor mₙ \]

\[ \frac{\ell₁ \lor \cdots \lor \ellₖ \land m₁ \lor \cdots \lor mₙ}{\text{UNIFY}(\ellᵢ, -mⱼ) = \theta} \]

For example,

\[ \neg \text{Rich}(x) \lor \text{Unhappy}(x) \]

\[ \frac{\text{Rich(Ken)}}{\text{Unhappy(Ken)}} \]

with \( \theta = \{x/Ken\} \)
Apply resolution steps to CNF(\(KB \land \neg \alpha\)); complete for FOL

Prolog examples

Depth-first search from a start state X:

dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).

No need to loop over S: successor succeeds for each

Appending two lists to produce a third:

\[ \text{append}([],Y,Y) \]
\[ \text{append}([X|L],Y,[X|Z]) :\text{ append}(L,Y,Z). \]

query: \text{append}(A,B,[1,2]) ?

answers: \(A=\[]\ B=[1,2]\)
\(A=[1] \ B=\[]\)
\(A=[1,2] \ B=\[]\)

Conversion to CNF

Everyone who loves all animals is loved by someone:

\[ \forall x \ [ \forall y \ (\text{Animal}(y) \Rightarrow \text{Loves}(x,y)) \Rightarrow [\exists y \ \text{Loves}(y,x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg \forall y \ (\neg \text{Animal}(y) \lor \text{Loves}(x,y)) \lor [\exists y \ \text{Loves}(y,x)] \]

2. Move \( \neg \) inwards:

\[ \forall x \ [\neg \exists x, p \equiv \exists x \neg p, \neg \exists x, p \equiv \forall x \neg p; \]

\[ \forall x \ [\exists y \ (\neg \text{Animal}(y) \lor \text{Loves}(x,y))) \lor [\exists y \ \text{Loves}(y,x)] \]
\[ \forall x \ [\exists y \ (\neg \text{Animal}(y) \land \neg \text{Loves}(x,y)) \lor [\exists y \ \text{Loves}(y,x)] \]
\[ \forall x \ [\exists y \ (\text{Animal}(y) \land \neg \text{Loves}(x,y)) \lor [\exists y \ \text{Loves}(y,x)] \]
Conversion to CNF contd.

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

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

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

6. Distribute \( \land \) over \( \lor \):
\[ \left[ Animal(F(x)) \lor Loves(G(x), x) \right] \land \left[ \neg Loves(x, F(x)) \lor Loves(G(x), x) \right] \]