Inference in first-order logic

Chapter 9, Chapter 10 Sections 2–3, AIMA2e 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

450 B.C. Stoics propositional logic, inference (maybe)
322 B.C. Aristotle “syllogisms” (inference rules), quantifiers
1565 Cardano probability theory (propositional logic + uncertainty)
1847 Boole propositional logic (again)
1879 Frege first-order logic
1922 Wittgenstein proof by truth tables
1930 Gödel $\exists$ complete algorithm for FOL
1930 Herbrand complete algorithm for FOL (reduce to propositional)
1931 Gödel $\neg\exists$ complete algorithm for arithmetic
1960 Davis/Putnam “practical” algorithm for propositional logic
1965 Robinson “practical” algorithm for FOL—resolution

Universal instantiation (UI)

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

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

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

E.g., $\forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x)$ yields

$$King(John) \land Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \land Greedy(Father(John))$$
$$\Rightarrow Evil(Father(John))$$
$$\vdots$$
Existential instantiation (EI)

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

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

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

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.

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; 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:
\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]

\[ King(John) \]

\[ Greedy(John) \]

\[ Brother(Richard, John) \]

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.

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,

\[ \text{e.g., Father(Father(Father(John)))} \]

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 \( \infty \) 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 in FOL is semidecidable
## Problems with propositionalization

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

\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]

\[ King(John) \]

\[ \forall y \ Greedy(y) \]

\[ Brother(Richard, John) \]

it seems obvious that \( Evil(John) \), but propositionalization produces lots of facts such as \( Greedy(Richard) \) 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 \( King(x) \) and \( Greedy(x) \) match \( King(John) \) and \( Greedy(y) \)

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

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

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

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></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{OJ})$</td>
<td></td>
</tr>
</tbody>
</table>
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

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

<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>

---

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

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

<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.,

$\text{Knows}(z_{17}, \text{OJ})$
Generalized Modus Ponens (GMP)

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

where \( p_i' \theta = p_i \theta \) for all \( i \)

\[ q \theta \]

\( p_1' \) is King(John) \hspace{1cm} p_1 \) is King(x)

\( p_2' \) is Greedy(y) \hspace{1cm} p_2 \) is Greedy(x)

\( \theta \) is \( \{x/John, y/John\} \) \hspace{1cm} q is Evil(x)

\( q \theta \) is Evil(John)

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

Soundness of GMP

Need to show that

\[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \models q \theta \]

provided that \( p_i' \theta = p_i \theta \) for all \( i \)

Lemma: For any definite clause \( p \), we have \( p \models p \theta \) 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. \( p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1' \theta \land \ldots \land p_n' \theta \)

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:
Example knowledge base contd.

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

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

Nono ... has some missiles, i.e., \( \exists x \ Owns(Nono, x) \land Missile(x) : Owns(Nono, M_1) \) and \( Missile(M_1) \)

... 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:
\[ American(x) \land Weapon(y) \land Sells(x, y, z) \land\]
\[ Hostile(z) \Rightarrow Criminal(x) \]
Nono ... has some missiles, i.e., \( \exists x \, Owns(Nono, x) \land Missile(x) : \)
\[ Owns(Nono, M_1) \land Missile(M_1) \]
... all of its missiles were sold to it by Colonel West
\[ \forall x \, Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono) \]
Missiles are weapons:

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[ American(x) \land Weapon(y) \land Sells(x, y, z) \land\]
\[ Hostile(z) \Rightarrow Criminal(x) \]
Nono ... has some missiles, i.e., \( \exists x \, Owns(Nono, x) \land Missile(x) : \)
\[ Owns(Nono, M_1) \land Missile(M_1) \]
... all of its missiles were sold to it by Colonel West
\[ \forall x \, Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono) \]
Missiles are weapons:
\[ Missile(x) \Rightarrow Weapon(x) \]
An enemy of America counts as “hostile”: 
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., \( \exists x \ O\!w\!n\!s(Nono, x) \land \text{Missile}(x) \):
\[ O\!w\!n\!s(Nono, M_1) \land \text{Missile}(M_1) \]
... all of its missiles were sold to it by Colonel West
\[ \forall x \ \text{Missile}(x) \land O\!w\!n\!s(Nono, x) \Rightarrow \text{Sells(West, x, Nono)} \]

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 ...
\[ \text{American(West)} \]

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

---

Forward chaining algorithm

```plaintext
function FOL-FC-ASK(KB, \alpha) returns a substitution or false

repeat until new is empty
    new \leftarrow \emptyset
    for each sentence r in KB do
        \( (p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r) \)
        for each \( \theta \) such that \( (p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta \)
            for some \( p'_1, \ldots, p'_n \) 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)

Forward chaining proof

Weapon(M1)  Sells(West,M1,Nono)  Hostile(Nono)

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
Forward chaining proof

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 $Missile(x)$ retrieves $Missile(M_1)$

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

---

Hard matching example

$$Diff(wa, nt) \land Diff(wa, sa) \land$$

$$Diff(nt, q)Diff(nt, sa) \land$$

$$Diff(q, nsw) \land Diff(q, sa) \land$$

$$Diff(nsw, v) \land Diff(nsw, sa) \land$$

$$Diff(v, sa) \Rightarrow Colorable()$$

```
Diff(Red, Blue)  Diff(Red, Green)
Diff(Green, Red)  Diff(Green, Blue)
Diff(Blue, Red)  Diff(Blue, Green)
```

*Colorable()* is inferred iff the CSP has a solution

CSPs include 3SAT as a special case, hence matching is NP-hard
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions

inputs: KB, a knowledge base

goals, a list of conjuncts forming a query
θ, the current substitution, initially the empty substitution ⌀

local variables: ans, a set of substitutions, initially empty

if goals is empty then return {θ}

q' ← SUBST(θ, FIRST(goals))

for each r in KB where STANDARDIZE-A PART(r) = ( p₁ ∧ . . . ∧ pₙ ⇒ q)
and θ' ← UNIFY(q, q') succeeds

ans ← FOL-BC-ASK(KB, [p₁, . . . , pₙ|REST(goals)], COMPOSE(θ', θ)) ∪ ans

return ans

Backward chaining example

Criminal(West)
Backward chaining example

Criminal(West)

{x/West}

American(x)  Weapon(y)  Sells(x,y,z)  Hostile(z)

Backward chaining example

Criminal(West)

{x/West}

American(West)  Weapon(y)  Sells(x,y,z)  Hostile(z)
Backward chaining example

Backward chaining example

35

36
Backward chaining example

Backward chaining example

37

38


**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

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem instance as facts</td>
<td>Encode problem instance as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

Should be easier to debug $\text{Capital(NewYork, US)}$ 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 ⇒ 60 million 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

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:
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=[]
Resolution: brief summary

Full first-order version:

\[ \ell_1 \lor \cdots \lor \ell_k, \ m_1 \lor \cdots \lor m_n \]

\[
(l_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

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

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

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

with \( \theta = \{x/\text{Ken}\} \)

Apply resolution steps to \( \text{CNF}(KB \land \neg \alpha) \); complete for FOL

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: \( \forall x, p \equiv \exists x \ \neg p, \quad \neg \exists x, p \equiv \forall x \ \neg p \):

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

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

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

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

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

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

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

Resolution proof: definite clauses

[Diagram showing the resolution proof with definite clauses]