Inference in first-order logic

Chapter 9
Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution
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., \( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \) yields:

- \( King(John) \land Greedy(John) \Rightarrow Evil(John) \)
- \( King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \)
- \( King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John)) \)
  
  .
  
  .
  
  .
Existential instantiation (EI)

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

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

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

  $$Crown(C_1) \land OnHead(C_1, John)$$

  provided $C_1$ is a new constant symbol, called a Skolem constant
Reduction to propositional inference

Suppose the KB contains just the following:

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

\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 \textit{propositionalized}: proposition symbols are

  \text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}), \text{etc.}
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., \textit{Father}(\textit{Father}(\textit{Father}(\textit{John})))
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 non-entailed sentence.)
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 \textit{Evil}(\textit{John}), but propositionalization produces lots of facts such as \textit{Greedy}(\textit{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 $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ 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>$\text{Knows}(\text{John}, \text{x})$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>_</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, \text{x})$</td>
<td>$\text{Knows}(\text{y}, \text{OJ})$</td>
<td>_</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, \text{x})$</td>
<td>$\text{Knows}(\text{y}, \text{Mother}(\text{y}))$</td>
<td>_</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, \text{x})$</td>
<td>$\text{Knows}(\text{x}, \text{OJ})$</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 $\text{King}(x)$ 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>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></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>

- Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17},OJ)$
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>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>${x/OJ, y/\text{John}}$</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>

- **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 $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

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

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

- **Standardizing apart** eliminates overlap of variables, e.g., Knows($z_{17}, \text{OJ}$)
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>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>${x/OJ,y/\text{John}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td>${y/\text{John}, x/\text{Mother(John)}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td>${\text{fail}}$</td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., Knows($z_17$,OJ)
Unification

- To unify \( \text{Knows}(John,x) \) and \( \text{Knows}(y,z) \), \( \theta = \{y/John, x/z\} \) 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. \( \text{MGU} = \{y/John, x/z\} \)
The unification algorithm

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
    inputs: x, a variable, constant, list, or compound
            y, a variable, constant, list, or compound
            θ, the substitution built up so far

    if θ = failure then return failure
    else if x = y then return θ
    else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
    else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
    else if COMPOUND?(x) and COMPOUND?(y) then
        return UNIFY(ARGS[x], ARGs[y], UNIFY(Op[x], Op[y], θ))
    else if LIST?(x) and LIST?(y) then
        return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
    else return failure
```
The unification algorithm

function UNIFY-VAR(var, x, θ) returns a substitution
inputs: var, a variable
         x, any expression
         θ, the substitution built up so far
if \{var/val\} ∈ θ then return UNIFY(val, x, θ)
else if \{x/val\} ∈ θ then return UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) then return failure
else return add \{var/x\} to θ
Generalized Modus Ponens (GMP)

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

\[
q\theta
\]

- \( p_1' \) is King(John)
- \( p_1 \) is King(x)
- \( p_2' \) is Greedy(y)
- \( p_2 \) is Greedy(x)
- \( \theta \) is \{x/John, y/John\}
- \( q \) 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
\[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \vdash q_\theta \]
provided that \( p_i'\theta = p_i\theta \) for all \( i \)

Lemma: For any sentence \( p \), we have \( p \vdash p_\theta \) by UI

1. \( (p_1 \land \ldots \land p_n \Rightarrow q) \vdash (p_1 \land \ldots \land p_n \Rightarrow q)_\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta) \)
2. \( p_1', \ldots, p_n' \vdash p_1' \land \ldots \land p_n' \vdash p_1'\theta \land \ldots \land p_n'\theta \)
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, 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
\[
\text{Missile}(x) \land \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono})
\]

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
        (p_1 ∧ ... ∧ p_n ⇒ q) ← STANDARDIZE-APART(r)
        for each θ such that (p_1 ∧ ... ∧ p_n)θ = (p'_1 ∧ ... ∧ p'_n)θ for some p'_1, ..., p'_n 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(West)
- Missile(M1)
- Owns(Nono,M1)
- Enemy(Nono,America)
Forward chaining proof
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
Efficiency of forward chaining

Incremental forward chaining: 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 positive literal

Matching itself can be expensive:

**Database indexing** allows O(1) retrieval of known facts

- e.g., query $Missile(x)$ retrieves $Missile(M_1)$

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{Colorable}() \text{ 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-Apart(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

SUBST(COMPOSE(θ₁, θ₂), p) = SUBST(θ₂, SUBST(θ₁, p))
Backward chaining example

Criminal

West
Backward chaining example
Backward chaining example

```
Criminal(West)

American(West)  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)

{ }

Missile(y)
```
Backward chaining example
Backward chaining example
Backward chaining example

Diagram showing the relationships between concepts such as Criminal, American, Weapon, Sells, Hostile, Missile, Owns, Enemy, with rules and conditions associated with each node.
Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - \( \Rightarrow \) fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - \( \Rightarrow \) 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., 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 alive(X) :- not dead(X).
  - alive(joe) succeeds if dead(joe) fails
Prolog

Appending two lists to produce a third:

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

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

answers: \quad A = [] \quad B = [1,2]
\quad A = [1,2] \quad B = []
Resolution: brief summary

- **Full first-order version:**
  \[
  \frac{\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n}{(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta}
  \]

  where \(\text{Unify}(\ell_i, \neg m_j) = \emptyset\).

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

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

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

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

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

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

2. Move \( \neg \) inwards:
\[ \neg \forall x \ p \equiv \exists x \ \neg p, \ \neg \exists x \ p \equiv \forall x \ \neg p \]
\[ \forall x [\exists y \ \neg(\neg\text{Animal}(y) \vee Loves(x,y))] \vee [\exists y Loves(y,x)] \]
\[ \forall x [\exists y \ \neg\text{Animal}(y) \land \neg\text{Loves}(x,y)] \vee [\exists y Loves(y,x)] \]
\[ \forall x [\exists y \text{Animal}(y) \land \neg\text{Loves}(x,y)] \vee [\exists y Loves(y,x)] \]
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 \lor over \land:
   \[ \left[ Animal(F(x)) \lor Loves(G(x),x) \right] \land \left[ \neg Loves(x,F(x)) \lor Loves(G(x),x) \right] \]
Resolution proof: definite clauses

\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor \neg Criminal(x)

\neg Criminal(West)

American(West)

\neg American(West) \lor \neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Missile(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

Missile(M1)

\neg Missile(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Sells(West,M1,z) \lor \neg Hostile(z)

\neg Sells(West,M1,z) \lor \neg Hostile(z)

\neg Missile(M1) \lor \neg Owns(Nono,x) \lor Sells(West,x,Nono)

\neg Sells(West,M1,z) \lor \neg Hostile(z)

Missile(M1)

\neg Missile(M1) \lor \neg Owns(Nono,x) \lor Sells(West,x,Nono)

\neg Owns(Nono,M1) \lor \neg Hostile(Nono)

Missile(M1)

\neg Missle(M1) \lor \neg Owns(Nono,M1) \lor \neg Hostile(Nono)

\neg Owns(Nono,M1) \lor \neg Hostile(Nono)

\neg Enemy(x,America) \lor \neg Hostile(x)

\neg Enemy(Nono,America)

\neg Enemy(Nono,America)

\neg Enemy(Nono,America)

\neg Hostile(Nono)

\neg Hostile(Nono)