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

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

• 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
Reduction to propositional inference

Suppose the KB contains just the following:
\[ \forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \]
King(John)
Greedy(John)
Brother(Richard, John)

- Instantiating the universal sentence in all possible ways, we have:
  King(John) \land \text{Greedy}(John) \Rightarrow \text{Evil}(John)
  King(Richard) \land \text{Greedy}(Richard) \Rightarrow \text{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))))
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:
  \( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \)
  King(John)
  \( \forall y \ Greedy(y) \)
  Brother(Richard,John)

• it seems obvious that \textit{Evil(John)}, but propositionalization produces lots of facts such as \textit{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 $\text{King}(x)$ 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>$\text{Knows(John,x)}$</td>
<td>$\text{Knows(John,Jane)}$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows(John,x)}$</td>
<td>$\text{Knows(y,OJ)}$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows(John,x)}$</td>
<td>$\text{Knows(y,Mother(y))}$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows(John,x)}$</td>
<td>$\text{Knows(x,OJ)}$</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 $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

$\theta = \{x/John, y/John\}$ 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., 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>${x/\text{Jane}}$</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>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td>${x/\text{OJ}, y/\text{John}}$</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., $\text{Knows}(z_{17}, \text{OJ})$
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>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td>${x/\text{OJ}, y/\text{John}}$</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., $\text{Knows}(z_{17}, \text{OJ})$
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 UnifyARGS[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

\begin{verbatim}
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
\end{verbatim}
Generalized Modus Ponens (GMP)

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

qθ

\[
p_1' \text{ is } King(John) \quad p_1 \text{ is } King(x)
\]

\[
p_2' \text{ is } Greedy(y) \quad p_2 \text{ is } Greedy(x)
\]

\[
\theta \text{ is } \{x/John, y/John\} \quad q \text{ is } Evil(x)
\]

q θ 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 sentence \( p \), 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:

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

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 …

\[
\text{Enemy}(\text{Nono},\text{America})
\]
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 \& \ldots \& pₙ \Rightarrow q) ← STANDARDIZE-APART(r)
        for each \( \theta \) such that \((p₁ \& \ldots \& pₙ)\theta = (p₁' \& \ldots \& pₙ')\theta\)
            for some \( p₁', \ldots, pₙ' \) in KB
                q' ← SUBST(\( \theta, q \))
                if q' is not a renaming of a sentence already in KB or new then do
                    add q' to new
                    \( \phi ← \text{UNIFY}(q', \alpha) \)
                    if \( \phi \) is not fail then return \( \phi \)
                add new to KB
        return false
Forward chaining proof

\text{American(West)} \quad \text{Missile(M1)} \quad \text{Owns(Nono,M1)} \quad \text{Enemy(Nono,America)}
Forward chaining proof

- American(West)
- Weapon(M1)
- Sells(West,M1,Nono)
- Owns(Nono,M1)
- Missile(M1)
- Hostile(Nono)
- Enemy(Nono,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 α 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 $k-1$

⇒ 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 \textit{Missile}(x) retrieves \textit{Missile}(M_1)

Forward chaining is widely used in deductive databases
Hard matching example

• **Colorable()** is inferred iff the CSP has a solution
• CSPs include 3SAT as a special case, hence matching is NP-hard

\[
\text{Diff}(\text{wa}, \text{nt}) \land \text{Diff}(\text{wa}, \text{sa}) \land \text{Diff}(\text{nt}, \text{q}) \land \\
\text{Diff}(\text{nt}, \text{sa}) \land \text{Diff}(\text{q}, \text{nsw}) \land \text{Diff}(\text{q}, \text{sa}) \land \\
\text{Diff}(\text{nsw}, \text{v}) \land \text{Diff}(\text{nsw}, \text{sa}) \land \text{Diff}(\text{v}, \text{sa}) \implies \\
\text{Colorable()}
\]

\[
\text{Diff(Red,Blue)} \quad \text{Diff (Red,Green)} \\
\text{Diff(Green,Red)} \quad \text{Diff(Green,Blue)} \\
\text{Diff(Blue,Red)} \quad \text{Diff(Blue,Green)}
\]
Backward chaining algorithm

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
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

- Criminal(West)
- American(West)
  - {}  
- Weapon(y)
- Sells(x,y,z)
- Hostile(z)
- Missile(y)
  - { y/ML }
Backward chaining example

```
Criminal(West)

American(West)  Weapon(y)  Sells(West,M1,z)  Hostile(z)
  \         /       \       /                \
  {}         y/M1     z/Nono     z/Nono

Missile(y)  Missile(M1)  Owns(Nono,M1)
  \                     /   /
  y/M1                M1
```
Backward chaining example

Diagram:

- Criminal(West)
  - American(West)
    - Missile(y)
  - Weapon(y)
  - Sells(West,M1,z)
    - Owns(Nono,M1)
  - Hostile(Nono)
    - Enemy(Nono,America)
Backward chaining example

```
Criminal(West)

American(West)
  [ ]

Weapon(y)

Sells(West, M1, z)
  [ z/Nono ]

Hostile(Nono)

  Missile(y)
  [ y/M1 ]

  Missile(M1)
  [ ]

  Owns(Nono, M1)
  [ ]

  Enemy(Nono, America)
  [ ]

{x/West, y/M1, z/Nono}
```
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., 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:
  
  \begin{align*}
  \text{append}([], Y, Y). \\
  \text{append}([X|L], Y, [X|Z]) & : \text{-- append}(L, Y, Z).
  \end{align*}

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

- answers: \begin{align*}
  A &= [] & B &= [1, 2] \\
  A &= [1, 2] & B &= []
  \end{align*}
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 \( \text{Unify}(\ell_i, \neg m_j) = \theta \).

• The two clauses are assumed to be standardized apart so that they share no variables.

• For example,

\[
\neg Rich(x) \lor Unhappy(x)
\]

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

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

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

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

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

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

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

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

\[ \neg \text{Criminal}(\text{West}) \]

\[ \neg \text{American}(\text{West}) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]

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

\[ \neg \text{Missile}(x) \lor \text{Weapon}(x) \]

\[ \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono},x) \lor \text{Sells}(\text{West},x,\text{Nono}) \]

\[ \neg \text{Missile}(\text{M1}) \]

\[ \neg \text{Missile}(\text{M1}) \lor \neg \text{Owns}(\text{Nono},\text{M1}) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Owns}(\text{Nono},\text{M1}) \]

\[ \neg \text{Enemy}(x,\text{America}) \lor \text{Hostile}(x) \]

\[ \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Hostile}(\text{Nono}) \]

\[ \text{Enemy}(\text{Nono},\text{America}) \]

\[ \text{Enemy}(\text{Nono},\text{America}) \]