Inference in first-order logic

Chapter 9
Outline

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

• We know how to efficiently represent complex worlds in FOL
  – Quantification + variables → FOL much more flexible, compact

• We know how to do inference in propositional logics
  – Model-checking OR inference rules OR resolution
  – Can be arduous, but it’s fairly mechanical. Sound and complete
  – Size of problem becomes impossible as complexity of world grows.

• Would like to use FOL...and do inference in FOL.
  – Quantification + variables → make things much more complex.

• Question: How can we reason in FOL?
  – We can convert FOL to propositional → use propositional inferencing
  – We can develop new ideas to deal with instantiation/variables
    • Reasoning directly in FOL!
Universal Instantiation (UI)

• Goal: Get rid of universal quantifiers
• Plan: Logically equivalent replacement
  – Consider all facts that a universally quantified sentence could imply.
  – Replace the quantified sentence by all implied facts.

• Example. Consider:
  – ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)

• What are all the literal facts that this could imply?
  – King(John) ∧ Greedy(John) ⇒ Evil(John)
  – King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard)
  – King(Father(John)) ∧ Greedy(Father(John)) ⇒ Evil(Father(John))
  – Etc. etc. etc.  \( ∀x \) means “true for every possible literal substitution

• Every possible instantiation of a universally quantified sentence is entailed by it:
  \[
  \forall v \alpha \quad \frac{\Rightarrow}{\text{Subst}\{\{v/g\}, \alpha\}}
  \]
  for any variable \( v \) and ground term \( g \)

• So: for a given FOL fact base: substitute all \( ∀x \) with literal facts implied.
Existential Instantiation (EI)

- Goal: Get rid of existential quantifiers
- Plan: Logically equivalent replacement
- Example. Consider:
  - $\exists x \text{Crown}(x) \land \text{OnHead}(x, \text{John})$

- What exactly does this mean?
  - There is some literal in the world for which this is true.
  - But for which literal? What if we don’t know it yet? Or anonymous: mom(Kate)

- Answer: create a special unique anonymous literal: Skolem Constant

  $\exists v \alpha$ Subst({v/k}, $\alpha$) For any sentence $\alpha$, variable $v$, and constant symbol $k$

  that does not appear elsewhere in the knowledge base

  - So: Crown(C1) $\land$ OnHead(C1, John), where C1 is a new Skolem symbol

- So, to get rid of quantification:
  - 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 strictly equivalent to the old...but is satisfiable iff the old KB was satisfiable
Example: 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 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: King(John), Greedy(John), Evil(John), King(Richard) etc.
  – Note: not predicates! Consider them monolithic symbols like $B_{1,1}$ and $W_{3,2}$
Analysis: Reducing FOL to Propositional Inference

- Claim: a sentence $\alpha$ is entailed by new KB iff entailed by original KB
- Claim: every FOL KB can be propositionalized so as to preserve entailment
- Idea! Do inference in FOL KB by:
  1. Propositionalize KB and query
  2. apply propositional resolution, return result
- Problem: function symbols: for any input, yield an output entity
  - Can refer to infinitely many entities, e.g., Father(Father(Father(John)))
  - Propositionalized KB is essentially infinite!

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

- Yields new 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
- Basically the IDS concept applied to proving entailment

- Problem: works if $\alpha$ is entailed... But never stop looking if $\alpha$ is not entailed
  - Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable
Other practical problems with propositionalization

• Another problem: Propositionalization generates lots of irrelevant sentences.
  – E.g., from:
    • $\forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x)$
    • $King(John), \ Prince(Andrew), \ Lady(Di), \ Stodgy(Mum)$
    • $\forall y \ Greedy(y)$
    • $Brother(Richard, John)$
  – it seems instantly *obvious* to human readers that $Evil(John)$,
    ...but propositionalization produces *lots* of facts that are irrelevant:
    • $Greedy(Richard), \ Greedy(Andrew), \ Greedy(Di), \ Greedy(Mum), \ etc$

• With $p$ $k$-ary predicates and $n$ constants, there are $p \cdot n^k$ instantiations!
  – And with function symbols, it gets much much worse!

• We need a more straightforward way to inference directly in FOL !!!
Unification

• Observation:
  – Given previous KB, we can get the inference Evil(John) 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!

• Definition: \(\text{Unify}(\alpha, \beta) = \theta\) if \(\text{subst}(\theta, \alpha) = \text{subst}(\theta, \beta)\)
  – Two terms \(\alpha\) and \(\beta\) unify, if you can find a variable binding \(\theta\) that, when substituted in, makes the terms identical

• Example: Unify the following sentences:

<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>

• The last one fails...but shouldn’t. What’s going on?
  – The ‘x’ in Knows(John,x) is not necessarily same as ‘x’ in Knows(x,OJ)
    • Both are just variables...could be bound to anything.
  
  – **Standardize apart:** Variables in each clause made unique: Knows(John, x\(_{217}\))
Generalized Modus Ponens (GMP)

- We can leverage unification to generalize Modus Ponens Inference!
  - Reminder: Modus ponens was

\[
\frac{\alpha \Rightarrow \beta, \alpha}{\beta}
\]

Or, more generally:

\[
\frac{p_1, p_2, p_3...p_n, (p_1 \land p_2 \land \ldots \land p_n) \Rightarrow q}{q}
\]

- But now suppose that \( p_i \) and \( q \) are all parameterized by variables
- If there is some set of bindings \( \theta \) that makes all sentences in the premise of an implication identical to sentences already in the KB, then we can assert the conclusion.
- Or formally:

\[
\frac{p'_1, p'_2, p'_3...p'_n, (p'_1 \land p'_2 \land \ldots \land p'_n) \Rightarrow q}{\text{subst}(\theta,q)}
\]

where \( \text{subst}(\theta,p'_i) = \text{subst}(\theta,p_i) \) for all \( i \)

- So then:

\[
\frac{\text{King}(\text{John}), \text{Greedy}(y), (\text{King}(x) \land \text{Greedy}(y)) \Rightarrow \text{Evil}(x))}{\text{Evil}(\text{John})}
\]

for \( \theta = \{x/\text{John}, y/\text{John}\} \)

- Note that GMP is used with:
  - KB of definite clauses (exactly one positive term)
  - All variables are assumed to be universally quantified
Example: Putting it together

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: Putting it together (cont.)

... it is a crime for an American to sell weapons to hostile nations:
Example: Putting it together (cont.)

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

Nono . . . has some missiles
Example: Putting it together (cont.)

... 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
Example: Putting it together (cont.)

... 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., \( \exists x \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \) ...and applying EI
\[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]

... 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:
Example: Putting it together (cont.)

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

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
\[ \forall x \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono}) \]

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

An enemy of America counts as “hostile”: 
Example: Putting it together (cont.)

... 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., \( \exists x \text{Owns}(\text{Nono}, x) \land \text{Missile}(x): \)

\( \text{Owns}(\text{Nono}, M_1) \) 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 . . . 

\( \text{Enemy}(\text{Nono}, \text{America}) \)
Inference algorithms: Forward chaining

Similar to propositional logic, we can infer new facts using forward chaining.

```
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

First we have the known facts...

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

Forward chaining with:
- \(\forall x \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})\)
- \(\text{Missile}(x) \Rightarrow \text{Weapon}(x)\)
- \(\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)\)
Forward chaining proof

Continuing to infer with:
- \(\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)\)
Forward Chaining Proof

• **Properties** of Forward-chaining proof
  – Sound and complete for first-order definite clauses
    • Proof similar to propositional proof.
  – For Datalog (first-order definite clauses + no functions), FC terminates for Datalog in poly iterations
    • There at most $p \cdot n^k$ literals = upper bound on number of iterations.
    • May not terminate in general if $\alpha$ is not entailed. Fn’s generate infinite new facts.
    • This is unavoidable: entailment with definite clauses is semi-decidable

• **Efficiency** of forward chaining proof. Sources of inefficiency
  – Matching conjunctive premises to *all possible* literals
    • Can often *order* conjunctive clauses to quickly constrain literals.
  – Redundant matching of all rules on each cycle
    • Simple observation: no need to match a rule on iteration $k$ if a premise wasn’t added on iteration $k - 1$ $\Rightarrow$ match *only those rules* whose premise contains a newly added literal
  – Waste time generating facts irrelevant to goal
    • Inherent to forward chaining. Backward chaining more efficient for goal-directed.
    • Magic sets: Can dynamically rewrite rules to “specialize” them toward desired goal
Backward chaining algorithm

For query-oriented (goal-directed) proofs, backward chaining is more natural
  • Note that the algorithm is depth-first, recursive.

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 already applied)
  θ, the current substitution, initially the empty substitution { }
  local variables: answers, a set of substitutions, initially empty

if goals is empty then return {θ} ;; if I’ve proven everything, I’m done
q ← Subst(θ, First(goals)) ;; else grab the first goal in list and try to prove
for each sentence r in KB
  where Standardize-Apart(r) = ( p₁ ∧ . . . ∧ pₙ ⇒ q)
  - and θ' ← Unify(q, q') succeeds ;; unify goal with head of rule
new goals ← [ p₁, . . . , pₙ|Rest(goals)] ;; add RHS of rule to goal stack
;; now plug latest bindings made into goal stack and call recursively...
  answers ← FOL-BC-Ask(KB, new goals, Compose(θ', θ)) ∪ answers
return answers

Returns a set of substitutions
  • All possible sets of bindings for which entailment holds
  • Could also return an “iterator”: new set of bindings each time you ask
Backward chaining example

“Prove that Colonel West is a criminal”

$\text{Criminal}(\text{West})$

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

$\text{Owns}(\text{Nono}, M_1) \land \text{Missile}(M_1)$

$\forall x \quad \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(\text{Nono}, \text{America})$
Backward chaining example

\[
\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)
\]
\[
\text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1)
\]
\[
\forall x \quad \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})
\]
\[
\text{Missile}(x) \Rightarrow \text{Weapon}(x)
\]
\[
\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)
\]
\[
\text{American}(\text{West})
\]
\[
\text{Enemy}(\text{Nono, America})
\]
Backward chaining example

\[
\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \\
\text{Owns}(\text{Nono}, M_1) \land \text{Missile}(M_1) \\
\forall x \quad \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \\
\text{Missile}(x) \Rightarrow \text{Weapon}(x) \\
\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \\
\text{American}(\text{West}) \\
\text{Enemy}(\text{Nono, America})
\]
Backward chaining example

\[ \text{Criminal}(\text{West}) \]
\[ \{x/\text{West}\} \]

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

\[ \text{Weapon}(y) \]

\[ \text{Sells}(x, y, z) \]

\[ \text{Hostile}(z) \]

\[ \text{Missile}(y) \]

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]
\[ \text{American}(\text{West}) \]
\[ \text{Enemy}(\text{Nono, America}) \]

\[ \text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]
\[ \text{Owns}(\text{Nono, M}_1) \text{ and } \text{Missile}(M_1) \]
\[ \forall x \quad \text{Missile}(x) \wedge \text{Owns}(\text{Nono, x}) \Rightarrow \text{Sells}(\text{West, x, Nono}) \]

27
Backward chaining example

American(West) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal(x)
Owns(Nono, M₁) and Missile(M₁)
∀x Missle(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)
Missile(x) ⇒ Weapon(x)
Enemy(x, America) ⇒ Hostile(x)
American(West)
Enemy(Nono, America)
Backward chaining example

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

\[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]

\[ \forall x \quad \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

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

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

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

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

\[ \text{Owns} (\text{Nono}, M_1) \text{ and Missile} (M_1) \]

\[ \forall x \quad \text{Missile} (x) \land \text{Owns} (\text{Nono}, x) \Rightarrow \text{Sells} (\text{West}, x, \text{Nono}) \]

\[ \text{Missile} (x) \Rightarrow \text{Weapon} (x) \]

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

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

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

- **Properties** of Backward chaining proof
  - 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
    - Breaks loop...but may not solve incompleteness problem (inherent to DFS)
  - Inefficient due to repeated subgoals (both success and failure)
    - fix using caching of previous results (extra space!)
  - Widely used (with improvements!) for logic programming
Logic programming

Logic programming:
• is a way to solve problems declaratively
• Based directly on FOL... (using datalog semantics)

Conceptually: 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 Procedural 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 \( \text{Capital(} \text{NewYork, US} \text{)} \) 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.
  - e.g.: 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

- 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:
  
  \[\text{dfs}(X) :\text{= } \text{goal}(X).\]
  
  \[\text{dfs}(X) :\text{= successor}(X,S), \text{dfs}(S).\]
  
  – 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: \[\text{append}([2],[4,6,8], Z).\]
    • Yields (yawn): \[2,4,6,8\]

  – Query: \[\text{append}(A, B, [1,2]) \text{?}\]
    • WOW!! Much more powerful than procedural functions!
    • Provides all possible bindings of A, B that could (when appended) yield \[1,2\]
FOL Resolution: Brief summary

- Principle: Identical to propositional resolution

- Full first-order version:

\[
\begin{array}{c}
l_1 \lor \ldots \lor l_k, \ m_1 \lor \ldots \lor m_n \\
\hline
\text{Subst}(\theta, (l_1 \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n))
\end{array}
\]

- where \( \text{Unify}(l_i, \lnot m_j) = \theta \).

- So same as before: match complemented predicates
  - But now: that unify with some set of bindings to be identical

- For example,

\[
\begin{array}{c}
\lnot \text{Rich}(x) \lor \text{Unhappy}(x) \\
\text{Rich}(\text{Ken})
\end{array}
\]

\[
\begin{array}{c}
\hline
\text{Unhappy}(\text{Ken})
\end{array}
\]

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

- To show entailment of \( \alpha \): Apply resolution steps to CNF (KB \land \lnot \alpha)

- Is 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

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

2. Move \( \neg \) inwards:
   - Note that: \( \neg \forall x, p \equiv \exists x \neg p \)
   - And that: \( \neg \exists x, p \equiv \forall x \neg p \)

So:

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

3. Standardize variables: each quantifier should use a different one

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

4. Skolemize:
   - a more general form of existential instantiation.
   - Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
     - Could generate any possible literal as the one that “exists”

\[ \forall x \left[ \text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x)) \right] \lor \text{Loves}(G(x), x) \]

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 \( \land \) over \( \lor \):

\[ \left[ \text{Animal}(F(x)) \lor \text{Loves}(G(x), x) \right] \land \left[ \neg \text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x) \right] \]
Example: Resolution proof
Summary: Inference in FOL

• Fred Flintstone: Turn FOL into propositional logic; find entailments
  – Use of Universal Instantiation and Existential Instantiation
  – Works...but is slow for all but the smallest domains.

• Better plan: Use unification to identify possible bindings of variables!
  – Can be seen as a “goal-focused instantiation”
  – Then can just use generalized Modus Ponens for inference
  – Forward and Backward chaining algorithms implement this approach in two different ways

• Forward chaining is used in production systems (expert systems)
  – Forward chaining is complete for datalog → runs in polynomial time

• Backward chaining is best for query-oriented proof
  – Used in logic programming systems like Prolog
  – Suffers from typical DFS issues: infinite loops and incompleteness

• Prolog uses database semantics to implement a subset of FOL

• Generalized resolution proof provides a complete proof system for FOL
\( \alpha \beta \subseteq \neg \Rightarrow \models \land \lor \)