Quick Quiz
10 minutes

Put in CNF

$(A \vee B) \Rightarrow (C \wedge D)$.

Reduction to propositional inference

Suppose the KB contains just the following:

$\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

King(John)

Greedy(John)

Brother(Richard,John)

- Instantiating the universal sentence in all possible ways, we have:

  King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)

  King(Richard) $\wedge$ 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 to propositional inference

- 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 in PC, return result
- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., *Father*(*Father*(*Father*(*John*)))

Reduction to propositional inference

Theorem: Herbrand (1930). If a sentence α is entailed by a FOL KB, it is entailed by a finite subset of the propositionalized KB

Problem: works if α is entailed, loops if α 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.)

→ Resolution won't always give an answer since entailment is only semidecidable

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
- E.g., from:
- $\forall x$ King(x) $\wedge$ 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

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

p_1' is ***King*(*John*)** p_1 is ***King*(*x*)**

p_2' is ***Greedy*(*y*)** p_2 is ***Greedy*(*x*)**

θ is {x/John,y/John} q is ***Evil*(*x*)**

q θ is ***Evil*(*John*)**

Soundness and Completeness of GMP

- GMP is sound
  - Only derives sentences that are logically entailed (proof on p276 in text)
  - GMP is not complete for FOL
  - Generalized Modus Ponens ***is*** complete for KBs consisting of definite clauses
  - Complete: derives all sentences that are entailed
  - OR…answers every query whose answers are entailed by such a KB
  - Definite clause: disjunction of literals of which exactly one is positive,
  - e.g., King(x) AND Greedy(x) -> Evil(x)

    NOT(King(x)) OR NOT(Greedy(x)) OR Evil(x)

Conjunction Normal Form (CNF)

We like to prove: $KB \models \alpha$

equivalent to : $KB \wedge \neg\alpha$ *unsatifiable*

We first rewrite $KB \wedge \neg\alpha$ into conjunctive normal form (CNF).

A "conjunction of disjunctions"

$(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Clause — Clause — literals

In theory
- Any KB can be converted into CNF.
- In fact, any KB can be converted into CNF-3, i.e. using clauses with at most 3 literals.

Example: Conversion to CNF (PC)

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Move $\neg$ inwards using de Morgan's rules and double-negation: $\neg(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta$

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Apply distributive law ($\wedge$ over $\vee$) and flatten:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution Algorithm in FOPC

1) Convert sentences in the KB to CNF (clausal form)
2) Take the negation of the proposed query, convert it to CNF, and add it to the KB.
3) Repeatedly apply the resolution rule to derive new clauses.
4) If the empty clause (False) is eventually derived, stop and conclude that the proposed theorem is true.

Procedure:

✓Eliminate implications and biconditionals
✓Move ¬ inward
✓Standardize variables
✓Move quantifiers left
✓Skolemize: replace each existentially quantified variable with a Skolem constant or Skolem function
✓Distribute $\wedge$ over $\vee$ to convert to conjunctions of clauses
✓Convert clauses to implications if desired for readability
$(\neg a \vee \neg b \vee c \vee d)$ To $a \vee b \Rightarrow c \vee d$

Conversion to CNF

- Everyone who loves all animals is loved by someone:

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

  1. Eliminate biconditionals and implications

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

2. Move ¬ inwards:"$\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\exists x\ p \equiv \forall x\ \neg p$"

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

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

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

Conversion to CNF contd.

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

   $\forall x([\exists y\ \textit{Animal}(y) \wedge \neg\textit{Loves}(x,y)] \vee [\exists z\ \textit{Loves}(z,x)])$

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

   $\forall x([\textit{Animal}(F(x)) \wedge \neg\textit{Loves}(x,F(x))] \vee \textit{Loves}(G(x),x))$

5. Drop universal quantifiers:

   $[\textit{Animal}(F(x)) \wedge \neg\textit{Loves}(x,F(x))] \vee \textit{Loves}(G(x),x)$

6. Distribute ∨ over ∧ :

   $[\textit{Animal}(F(x)) \vee \textit{Loves}(G(x),x)] \wedge [\neg\textit{Loves}(x,F(x)) \vee \textit{Loves}(G(x),x)]$

Resolution in PC

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

- Resolution inference rule (for CNF):

$$\frac{l_i \vee \ldots \vee l_k, \qquad m_1 \vee \ldots \vee m_n}{l_i \vee \ldots \vee l_{s-1} \vee l_{s+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where l_s and m_j are complementary literals.

E.g., $$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic

Resolution in FOL

- Full first-order version:

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\mathtt{Unify}(l_i, \neg m_j) = \theta$.

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

- For example,

$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

A More Compact Version

$$\frac{\bigvee_{i \in A} L_i \qquad \bigvee_{i \in B} L_i}{\bigvee_{i \in C} Subst(\theta, L_i)} \qquad \begin{array}{l} Unify(L_j, \neg L_k) \\ j \in A, k \in B \\ C = (A \cup B) \backslash \{j, k\} \end{array}$$

E.g. for A = {1, 2, 7} first clause is $L_1 \vee L_2 \vee L_7$

Empty Clause means False

- Resolution theorem proving ends
  - When the resolved clause has no literals (empty)
- This can only be because:
  - Two **unit clauses** were resolved
    - One was the negation of the other (after substitution)
  - Example: q(X) and ¬q(X)   or: p(X) and ¬p(bob)
- Hence if we see the empty clause
  - This was because there was an inconsistency
  - Hence the proof by refutation

Resolution as Search

- **Initial State**: Knowledge base (KB) of axioms and negated theorem in CNF
- **Operators**: Resolution rule picks 2 clauses and adds new clause
- **Goal Test**: Does KB contain the empty clause?

- Search space of KB states

Socrates' Example

- KB: ***Socrates is a man and all men are mortal Therefore Socrates is mortal***
- Initial state

  1) is_man(socrates)

  2) ¬is_man(X) ∨ is_mortal(X)

  3) ¬is_mortal(socrates) (negation of theorem)
- Resolving (1) & (2) gives new state

  4) is_mortal(socrates)

  Resolving (3) & (4) gives new state

  empty

Aristotle's Example: Search Space

1) is_man(socrates)
2) ¬is_man(X) ∨ is_mortal(X)
3) ¬is_mortal(socrates)

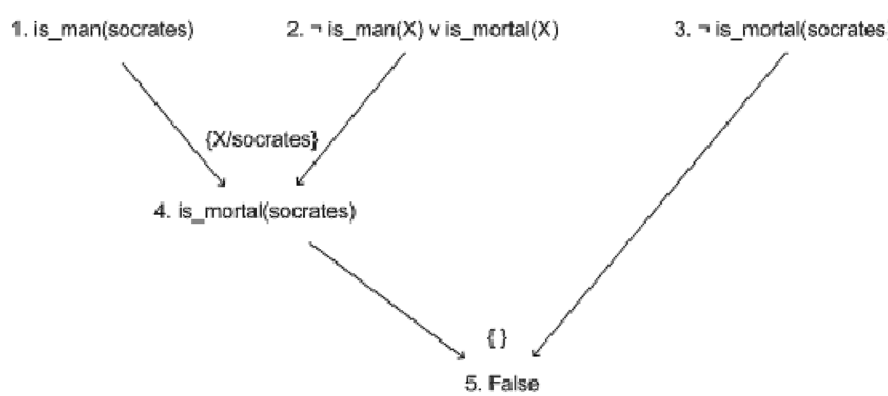
1) is_man(socrates)
2) ¬is_man(X) ∨ is_mortal(X)
3) ¬is_mortal(socrates)
4) is_mortal(socrates)

1) is_man(socrates)
2) ¬is_man(X) ∨ is_mortal(X)
3) ¬is_mortal(socrates)
4) is_mortal(socrates)
5) False

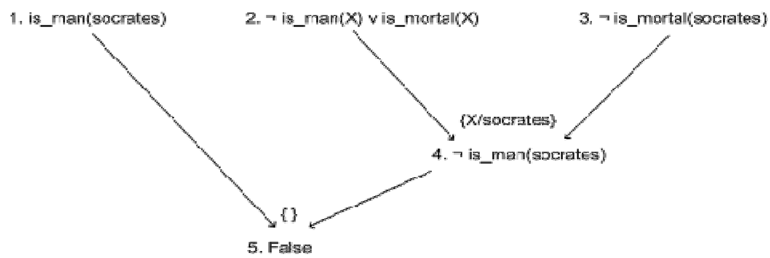
1) is_man(socrates)
2) ¬is_man(X) ∨ is_mortal(X)
3) ¬is_mortal(socrates)
4) ¬is_man(socrates)

1) is_man(socrates)
2) ¬is_man(X) ∨ is_mortal(X)
3) ¬is_mortal(socrates)
4) ¬is_man(socrates)
5) False

Resolution Proof Tree (Proof 1)



Resolution Proof Tree (Proof 2)

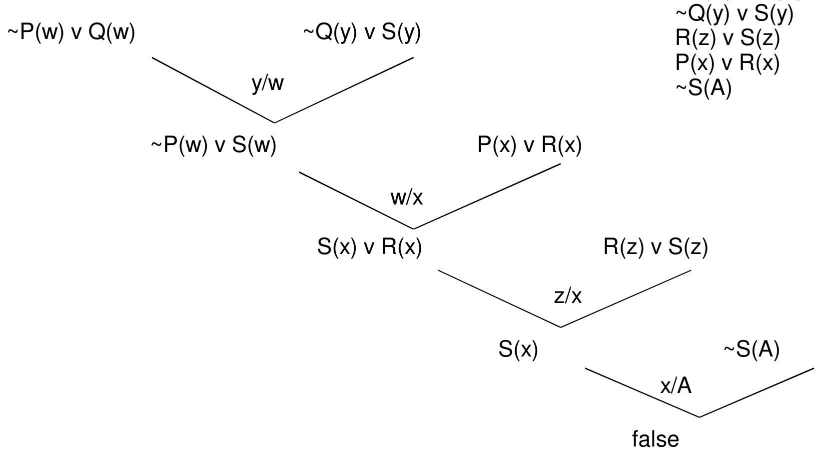


Read as:

You said that all men were mortal. That means that for all things X, either X is not a man, or X is mortal. If we assume that Socrates is not mortal, then, given your previous statement, this means Socrates is not a man. But you said that Socrates ***is*** a man, which means that our assumption was false, so Socrates must be mortal.

Building Refutation resolution proof tree

KB
~P(w) v Q(w)
~Q(y) v S(y)
R(z) v S(z)
P(x) v R(x)
~S(A)



Resolution Algorithm in FOPC

1) Convert sentences in the KB to CNF (clausal form)
2) Take the negation of the proposed query, convert it to CNF, and add it to the KB.
3) Repeatedly apply the resolution rule to derive new clauses.
4) If the empty clause (False) is eventually derived, stop and conclude that the proposed theorem is true.

Procedure:

✓Eliminate implications and biconditionals
✓Move ¬ inward
✓Standardize variables
✓Move quantifiers left
✓Skolemize: replace each existentially quantified variable with a Skolem constant or Skolem function
✓Distribute ∧ over ∨ to convert to conjunctions of clauses
✓Convert clauses to implications if desired for readability
(¬ a ∨ ¬ b ∨ c ∨ d) To a ∨ b => c ∨ d

Conversion to CNF

- Everyone who loves all animals is loved by someone:

$\forall x([\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)])$

1. Eliminate biconditionals and implications

$\forall x([\neg\forall y\ (\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)])$

2. Move ¬ inwards:"$\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg \exists x\ p \equiv \forall x\ \neg p$"

$\forall x\ ([\exists y\ (\neg(\neg Animal(y) \vee Loves(x,y)))] \vee [\exists y\ Loves(y,x)]\ )$

$\forall x\ ([\exists y\ (\neg\neg Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y\ Loves(y,x)]\ )$

$\forall x([\exists y\ (Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y\ Loves(y,x)]\ )$

Conversion to CNF contd.

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

   $\forall x([\exists y\ \textit{Animal}(y) \wedge \neg \textit{Loves}(x,y)] \vee [\exists z\ \textit{Loves}(z,x)])$

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

   $\forall x([\textit{Animal}(F(x)) \wedge \neg \textit{Loves}(x,F(x))] \vee \textit{Loves}(G(x),x))$

5. Drop universal quantifiers:

   $[\textit{Animal}(F(x)) \wedge \neg \textit{Loves}(x,F(x))] \vee \textit{Loves}(G(x),x)$

6. Distribute $\vee$ over $\wedge$:

   $[\textit{Animal}(F(x)) \vee \textit{Loves}(G(x),x)] \wedge [\neg \textit{Loves}(x,F(x)) \vee \textit{Loves}(G(x),x)]$

Example: KB

Jack owns a dog.
Every dog owner is an animal lover.
No animal lover kills an animal.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

Example: KB

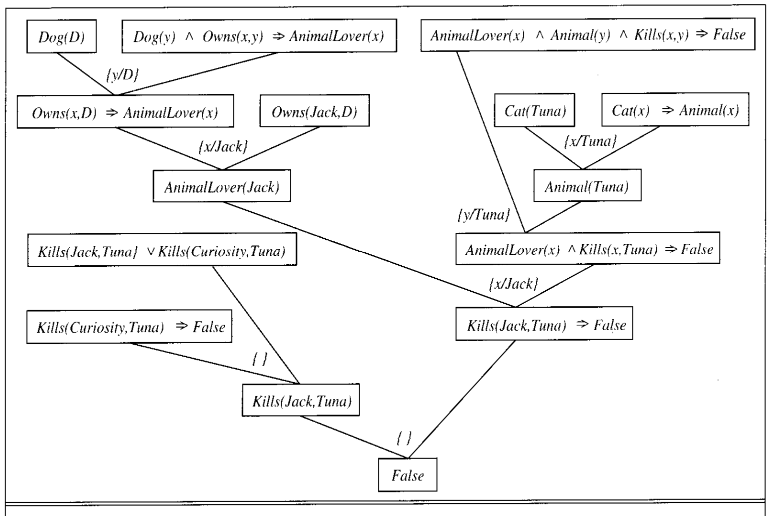
Jack owns a dog.
Every dog owner is an animal lover.
No animal lover kills an animal.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

A. $\exists x \; Dog(x) \wedge Owns(Jack, x)$
B. $\forall x \; (\exists y \; Dog(y) \wedge Owns(x, y)) \Rightarrow AnimalLover(x)$
C. $\forall x \; AnimalLover(x) \Rightarrow \forall y \; Animal(y) \Rightarrow \neg Kills(x, y)$
D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $\forall x \; Cat(x) \Rightarrow Animal(x)$

Example: (CNF)

A1. $Dog(D)$
A2. $Owns(Jack, D)$
B. $Dog(y) \wedge Owns(x, y) \Rightarrow AnimalLover(x)$
C. $AnimalLover(x) \wedge Animal(y) \wedge Kills(x, y) \Rightarrow False$
D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $Cat(x) \Rightarrow Animal(x)$

Example: Proof Tree



Forward chaining

- FC: "Idea" fire any rule whose premises are satisfied in the ***KB***, add its conclusion to the ***KB***, until query is found
- Deduce new facts from axioms
- Hopefully end up deducing the theorem statement
- ❖ Can take a long time: not using the goal to direct search
- 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

Forward Chaining

- Use modus ponens to always derive all consequences from new information
- To avoid looping and duplicated effort, must prevent addition of a sentence to the KB which is the same as one already present.

Problems with Forward Chaining

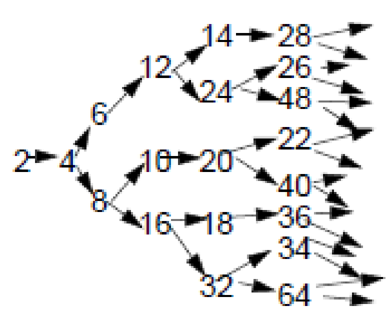
- Inference can explode forward and may never terminate.

Even(x) ➔ Even(plus(x,2))

Integer(x) ➔ Even(times(2,x))

Even(x) ➔ Integer(x)

Even(2)



Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
 $new \leftarrow \{\}$
 for each sentence r **in** KB **do**
  $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
  for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
   for some $p'_1, \ldots, p'_n$ in KB
   $q' \leftarrow$ SUBST(θ, q)
   if q' is not a renaming of a sentence already in KB or *new* **then do**
    add q' to *new*
    $\phi \leftarrow$ UNIFY(q', α)
    if ϕ is not *fail* **then return** ϕ
 add *new* to KB
return *false*

Backward chaining

- BC: "Idea" work backwards from the query ***q in (p→q)***
  - check if ***q*** is already known, or
  - prove by BC all premises of some rule concluding ***q***
- Start with the conclusion and work backwards
  - Hope to end up at the facts from KB
- Widely used for logic programming
- PROLOG is backward chaining

Remarks:

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal has already been proved true, or has already failed

Backward Chaining

- Start from a query or atomic sentence to be proven and look for ways to prove it
- Query can contain variables
- Inference process should return all sets of variables hat satisfy the query
- First try to answer query by unifying it to all possible facts in the KB
- Next to tries to prove it using a rule whose consequent unifies with the query and try to prove all its antecedents recursively

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
```