Socrates' Example

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

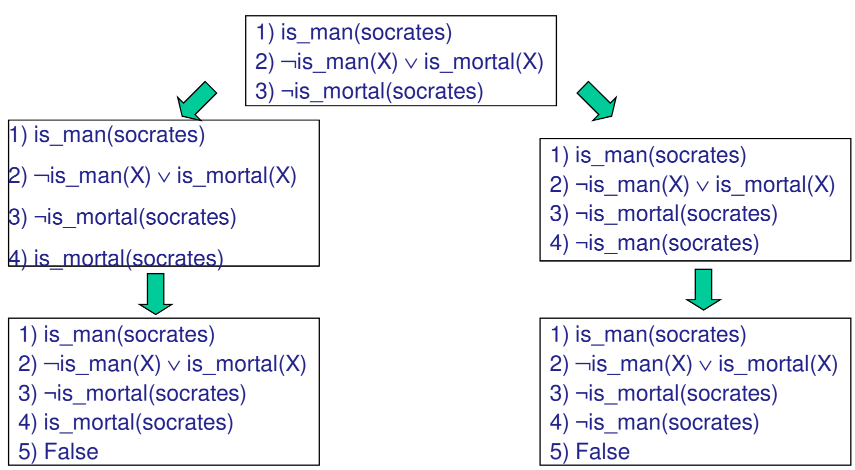
  1) is_man(socrates)

  2) ¬is_man(X) ∨ is_mortal(X)

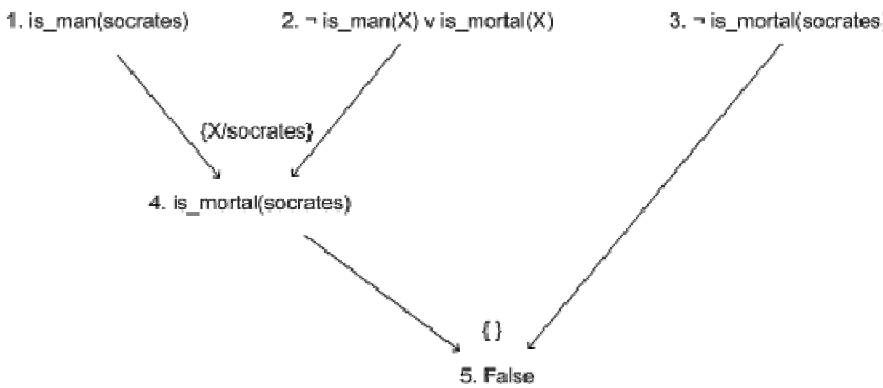
  3) ¬is_mortal(socrates)   (negation of theorem)
- Resolving (1) & (2) gives new state

  (1)-(3) & 4) is_mortal(socrates)

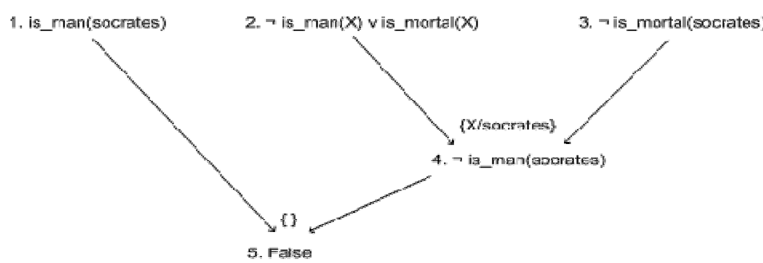
Aristotle's Example: Search Space



Resolution Proof Tree (Proof 1)



Resolution Proof Tree (Proof 2)



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.

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

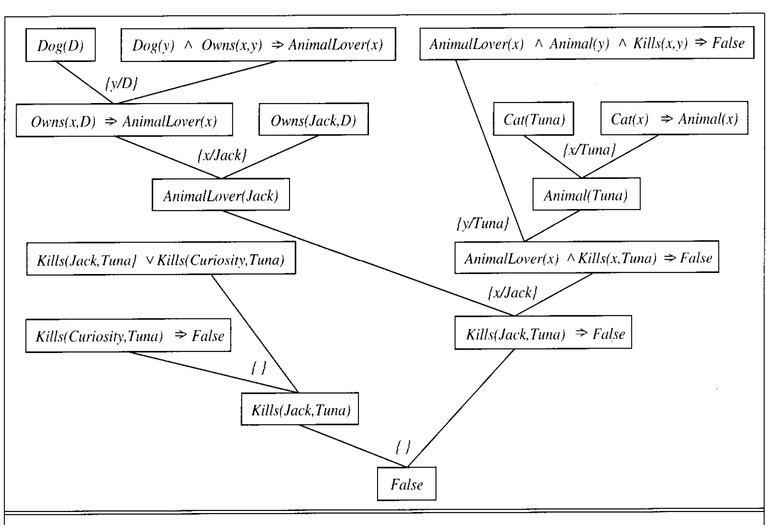
Jack owns a dog.
Every dog owner is an animal lover.
No animal lover kills an animal.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

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



Reduction to propositional inference

Suppose the KB contains just the following:

∀x King(x) ∧ Greedy(x) ⇒ Evil(x)

King(John)
Greedy(John)
Brother(Richard,John)

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

King(John) ∧ Greedy(John) ⇒ Evil(John)
King(Richard) ∧ Greedy(Richard) ⇒ 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.)

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 complete for a KB 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)

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 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
           ( p1 ∧ . . . ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
           for each θ such that (p1 ∧ . . . ∧ pn)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
```

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 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(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))