CS 520: Introduction to Artificial Intelligence

Prof. Louis Steinberg

Lecture 11:

First-order Logic

Proof Algorithms

Review

Representing a Domain in FOL

- **A domain is a portion of the world about which we wish to express knowledge.**
- **We can define a set of basic predicates from which all other predicates can be defined.**

An FOL Agent

- **Executive tells KB what sensors perceive**
- **Executive asks KB what it should do**

Review - Types of FOL Agents

- **Reflex agent**
  - **Non-FOL: percept --rule--> action**
  - **FOL:**

  percept --inference--> abstract percept --inference--> action
- **FOL agents with state**

  percept ------inference--> world state --inference--> action

  world state

Review - Situation Calculus

- **How to represent changeable facts in FOL**
- **Basic idea: time is a sequence of *states***

States and Predicates

- **Every changeable predicate gets an extra parameter: the state.**
- **Effect, frame, and successor state axioms**

Inference in FOL

- **Inference in FOL is semi-decidable**
  - **There are algorithms such that:**
    - **If an inference can be proven, the algorithm will produce a proof in finite time**
    - **If the inference cannot be proven, the algorithm may run forever**
  - **And this is the best we can do**
- **One such algorithm is Resolution Refutation**

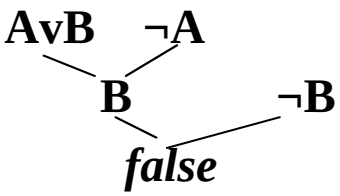
The key problem in FOL inference is handling universals

∀x even(x) <=> ∃y x=2*y

=> even(324) <=> ∃y 324=2*y

Resolution Refutation

- **Refutation proof:**
  - **to prove that KB entails α, add ¬α to KB and infer *false***
- **Eg, propositionally,**
  - **KB: A v B, ¬A. Prove: B**
  - **KB + ¬α : A v B, ¬A, ¬B**



Generalized Modus Ponens

- **Generalized Modus Ponens (GMP):**

$$\frac{A_1, \ldots, A_n, \quad A_1 \wedge \ldots \wedge A_n \Rightarrow B}{B}$$

- **Equivalent Resolution Refutation:**

$$\frac{A_1, \ldots, A_n, \quad \neg A_1 \vee \ldots \vee \neg A_n \vee B, \quad \neg B}{\textit{false}}$$

GMP and Quantifiers

- **We can get rid of ∃ by Skolemization**
  - **If we have ∃x P(x), make up a name for the object that makes this true.**

    ∃x ∀y loves(y, x) ----> ∀y loves(y, s)
  - **Must be a new name.**

    ∀y loves(y, ~~Raymond~~)
  - **If there are ∀ outside the ∃, use a function**

∀y ∃x loves(y, x) ----> ∀y loves(y, s(y))

∀w ∀y ∃x P(w, x) ^ Q(x, y) -------->

∀w ∀y P(w, s(w, y)) ^ Q(s(w, y), y)

GMP and ∀

Problem:

Anything a cow eats is a plant	**∀x eats(cow, x)=>plant(x)**
Everything eats lettuce	**∀y eats(y, lettuce)**
Lettuce is a plant	**plant(lettuce)**

These are not identical!

But if we substitute lettuce for y and cow for x

eats(cow, lettuce)=>plant(lettuce)

eats(cow, lettuce)

plant(lettuce)

GMP and ∀

If there is a substitution θ such that

SUBST(θ , P_i) = SUBST(θ , P_i')

SUBST(θ , Q) = SUBST(θ , Q')

Then $\frac{P'_1, \ldots, P'_n,\ P_1 \wedge \ldots \wedge P_n \Rightarrow Q}{Q'}$

Example

John loves his mother

Everyone's mother loves them

To love one who loves you is to be happy

John is happy

P_1': loves (John, mother-of (John))

P2': ∀x loves (mother-of (x), x)

P1 ∧ P2 ∧ … Pn⇒ Q:

∀x, y loves (y, z) ∧ loves(z, y) ⇒ happy(y)

Q': happy (John)

P's and Q's and θ

P_1' = loves (John, mother-of (John))
P_1 = loves (y, z)

P_2' = loves (mother-of (x), x)
P_2 = loves (z, y)

How?

θ = {y/John, z/mother-of(John), x/John)

Q' = happy(john)
Q = happy(y)

Unification

- Make variables in each sentence unique

  Already ok.

- For each pair of atomic sentences:

  loves (John, mother-of (John))
  loves (y, z)

⇒ Predicates must be the same

⇒ Unify each pair of corresponding terms

Unifying Terms

- **If either term is a variable:**

  John y

  add the substitution variable / other-term to θ :

  { y / John }

Unifying Terms

- **If either term is a variable:**

  John y

  add the substitution variable / other-term to θ :

  { y / John }

  apply to all sentences:

  loves (John, mother-of (John))

  loves (y, z)

  loves (mother-of (x), x)

  loves (z, y)

Unifying Terms

- **If either term is a variable:**

  John y

  add the substitution variable / other-term to θ :

  { y / John }

  apply to all sentences:

  loves (John, mother-of (John))
  loves (John, z)
  loves (mother-of (x), x)
  loves (z, John)

Again

- **Unify next pair of terms**

  mother-of (John)) z

  θ = { y / John, z / mother-of(John)}

  loves (John, mother-of (John))
  loves (John, mother-of (John))
  loves (mother-of (x), x)
  loves (mother-of (John), John)

Next Pair of Atomic Sentences

loves (mother-of (x), x)
loves (mother-of (John), John)

- **Predicates match**
  loves loves
- **Match terms, first**
  mother-of (x)
  mother-of (John)

To match Function Terms:

mother-of (x)
mother-of (John)

- **Functions must be the same**
  mother-of mother-of
- **Then match arguments**
  x
  John
  θ = {y / John, x / mother-of(John), x / John}
  loves (John, mother-of (John))
  loves (John, mother-of (John))
  loves (mother-of (John), John)
  loves (mother-of (John), John)

Final Pair of Terms

John

John

- **Two constants only match if they are the same.**

Final θ

θ = { y / John,
z / mother-of(John),
x / John}

- **Note: "Occurs check"**
  - **Can't unify x with f(x)**

Summary of Unification

- **Unify-Sentences(A, B)**
  - **Unify corresponding atomic sentences**
- **Unify-Atomic Sentences (A, B)**
  - **If either A or B is a variable, add substitution var / other term to θ and apply to A and B**
  - **If predicates differ, fail**
  - **Unify corresponding terms**
- **Unify-terms**
  - **If both A and B are constants**
    if A=B succeed else fail
  - **If both A and B are function terms**
    - **If not same function, fail**
    - **Unify-terms on corresponding arguments**

Normal Forms

- **We have discussed two proof methods involving two different inference rules:**
  - **Generalized Modus Ponens**
  - **Resolution refutation**
- **Each requires the KB in a certain form**
  - **GMP: implicative normal form (inf)**

    $A(x) \wedge B(x) \wedge \ldots \wedge C(x) \Rightarrow D(x)$
  - **RR: conjunctive normal form (cnf)**

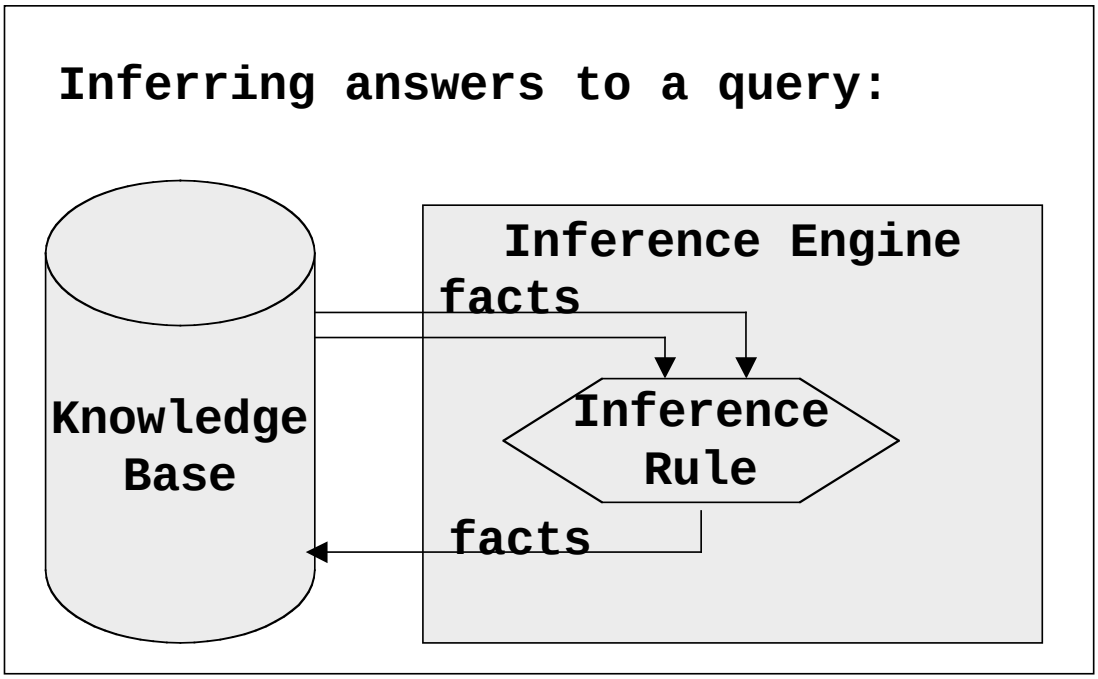
    $A(x) \vee B(x) \vee \ldots \vee D(x)$

Conjunctive Normal Form

- Eliminate implications
- Move ¬ inwards
- Standardize variables
- Move quantifiers left
- Skolemize ∃
- Distribute ∧

(For INF: gather negative literals and convert to implication)

What's Wrong With This Picture?



Forward vs Backward Chaining

Forward:

- I have
  - P_1'
  - P_2'
  - $P_1 \wedge P_2 \Rightarrow Q$
- I get Q

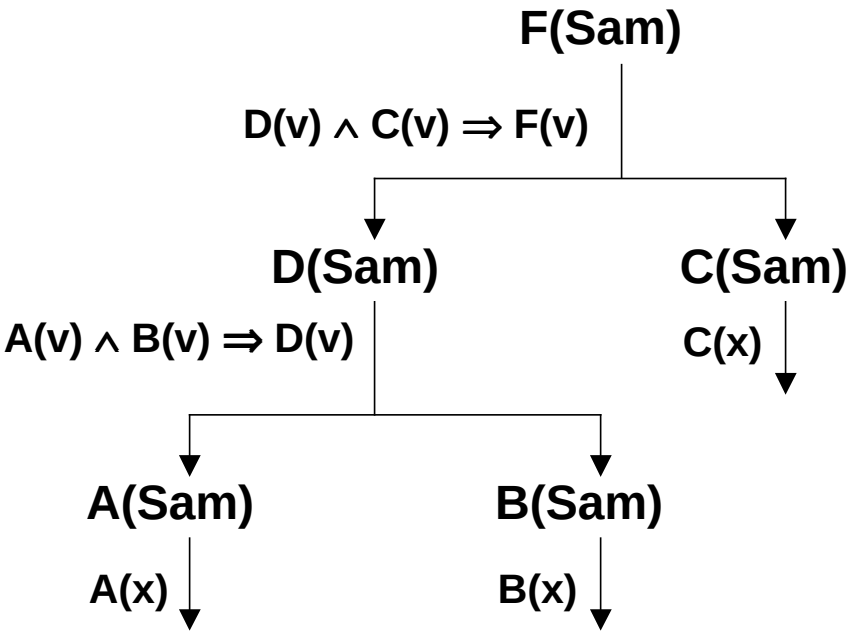
Backward:

- I want Q'
- I have
  - $P_1 \wedge P_2 \Rightarrow Q$
- So now I want
  - P_1'
  - P_2'

Example

- **KB:**
  - A(x)
  - B(y)
  - C(z)
  - $A(w) \wedge B(w) \Rightarrow D(w)$
  - $D(v) \wedge C(v) \Rightarrow F(v)$
- **Query:**
  - F(Sam)

A Proof Tree

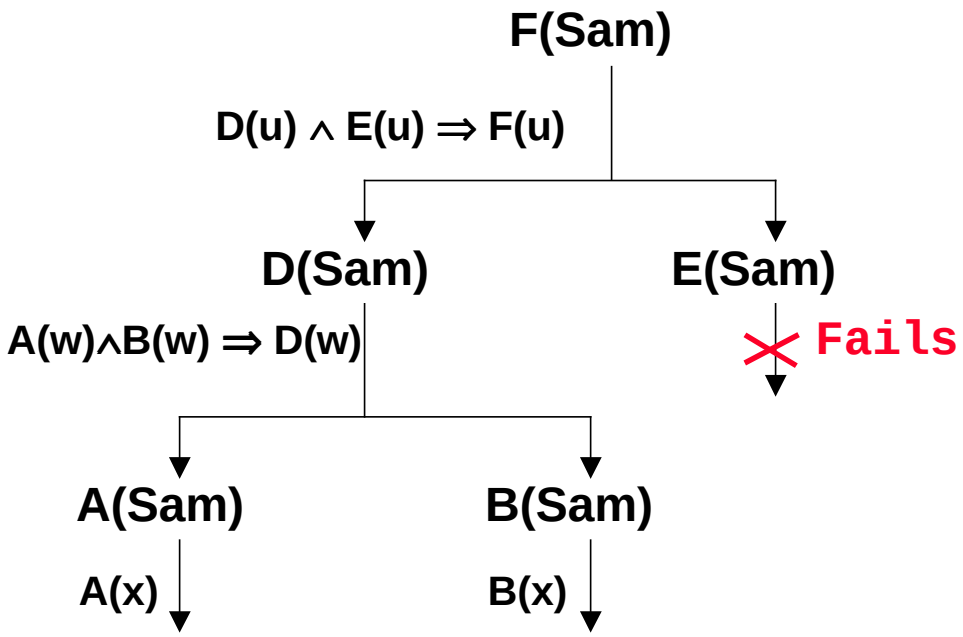


An Example With a Choice

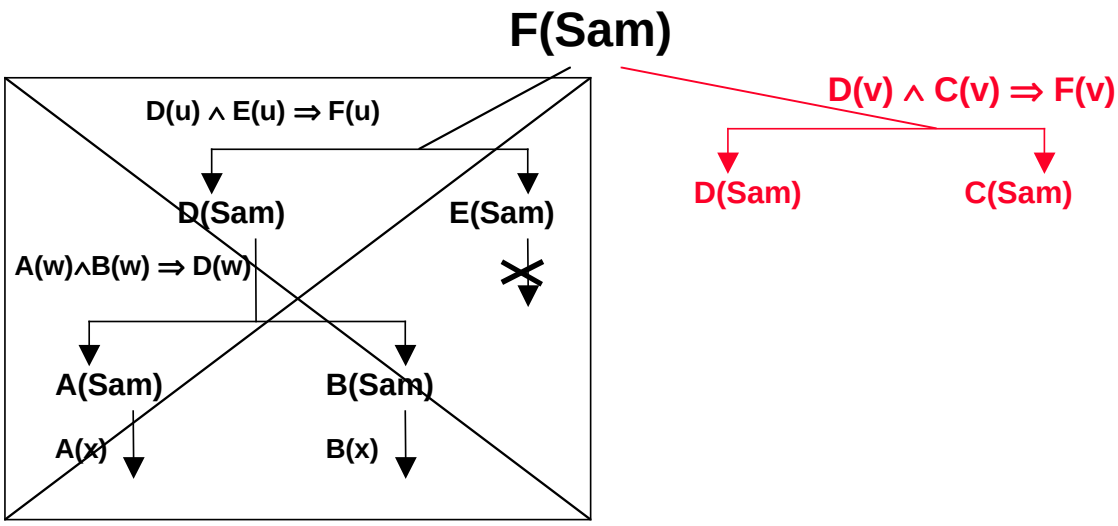
- **KB:**
  - A(x)
  - B(y)
  - C(z)
  - A(w) ∧ B(w) ⇒ D(w)
  - D(u) ∧ E(u) ⇒ F(u)
  - D(v) ∧ C(v) ⇒ F(v)

- **Query:**
  - F(Sam)

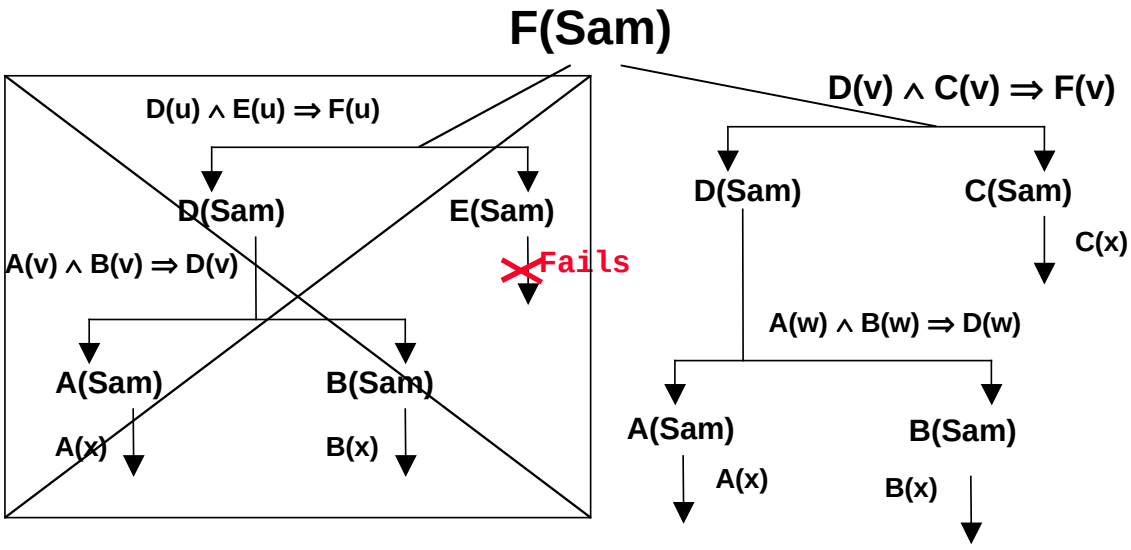
Searching for a Proof Tree



Find a Choice Point



And Continue



Searching a Tree of Search Trees