CS 520: Introduction to Artificial Intelligence

Prof. Louis Steinberg

Lecture 10:

First-order Logic

Representing a problem domain

Proof Algorithms

Review

- **A propositional agent cannot use rules generalized over objects or times**
- **Need First-Order Logic (FOL) to express generalization**
- **FOL: environment is made up of objects that can take part in relations.**
  - **Relation is a set of tuples**
  - **Special classes of relation:**
    - **Properties: unary relations, e.g. green(box1)**
    - **Functions: mother-of(John)**

Review

Quantifiers: ∀x, ∃x

- **Nested quantifiers**

  ∀x ∃y P(x, y) vs ∃y ∀x P(x, y)

- **¬ and Quantifiers**

  ∀x ¬P(x) ≡ ¬ ∃ x P(x)

  ∃x ¬ P(x) ≡ ¬ ∀ x P(x)

Review

- **English into first-order logic**
- **First-order logic into English**
  - **Ambiguity of English quantifiers**
  - **Implicit assumptions**
    - **Every dog is fed by its owner**
      ∀d dog(d) => ∃o (owns(o, d) ^ feeds(o, d))
      ∀d dog(d) => feeds(owner(d), d)
    - **Everyone loves Raymond**
      ∀x loves(x, Raymond)
      ∀x person(x) => loves(x, Raymond)

Some Extensions

- Higher-Order Logics allow quantification over functions and relations in addition to objects.
- The lambda-operator builds predicates and functions from simpler components.
- The uniqueness quantifier∃! means ``there is exactly one''.

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.
- For example, we can capture the domain of sets with
  - EmptySet (constant),
  - Member (predicate), Subset (predicate), Set (predicate),
  - Intersection (function),
  - Union (function), Adjoin (function)

Axioms for Sets

- **Sets are empty or made by adjoining something**

$$\forall s\ set(s) \Leftrightarrow (s = EmptySet \vee \exists x,s_1\ s = adjoin(x,s_1))$$

- **The empty set has no elements**

$$\neg\exists x,s\ adjoin(x,s) = EmptySet$$

- **Adjoining an element already there has no effect**

$$\forall x,s\ member(x,s) \Leftrightarrow s = adjoin(x,s)$$

- **The only members of a set are elements adjoined to it**

$$\forall x,s\ member(x,s) \Leftrightarrow \exists s_1\ s = adjoin(x,s_1) \wedge \neg member(x,s_1)$$

Prove $\neg\exists x\ member(x,EmptySet)$

An FOL Agent

- **Executive tells KB what sensors perceive**

  currentPercept(stench, nobreeze, noglitter, nobump)

- **Executive asks KB what it should do**

  ∃ action doNow(action) ?

  – **When proving an existential, KB finds an object that satisfies the existential**

  doNow(turnLeft)

A simple Reflex FOL Agent

- **A reflex agent has rules that map percepts to acts**

  currentPercept(_, _, glitter, _) -> doNow(grab)
- **FOL rules can also abstract perceptual input:**

  currentPercept(_, _, glitter, _) -> goldHere()
- **so that direct connections are made from abstract percepts to actions:**

  goldHere()-> doNow(grab)

Problems with Reflex Agents

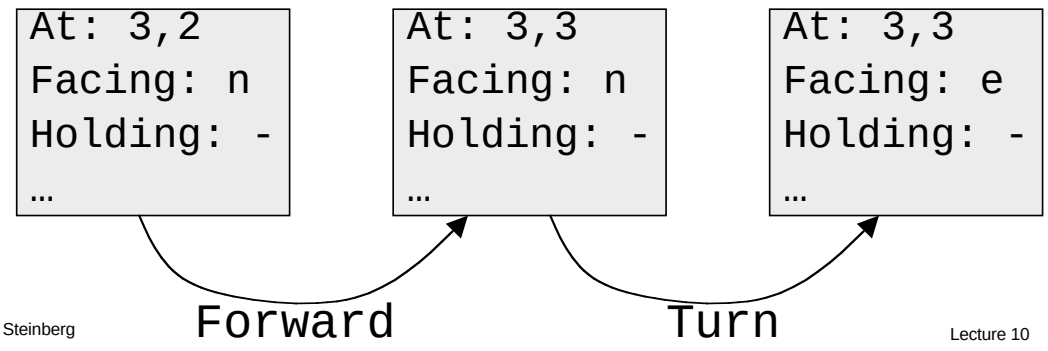
- **Because reflex agents do not maintain an internal representation of the world, even an FOL reflex agent, for example, never knows if it has the gold or is at the start state.**
- **Because it can't know where it came from, it cannot avoid getting trapped in infinite loops.**

How Do We Represent a Dynamic World?

- As new percept sentences are acquired by the agent (forming the percept sequence), how do we maintain our internal model of the world?
  - We could replace incorrect facts with new facts as they are updated in time. For example, At(Agent,[1,1]) could be changed to At(Agent,[1,2]).
  - This would mean only reasoning about the current state
- Solution: situation calculus

Situation Calculus

- How to represent changeable facts in FOL
  - Eg, location of agent
- Basic idea: time is a sequence of *states*
  - When the agent does an action, it transforms the world into a new state



States and Predicates

- **Every changeable predicate gets an extra parameter: the state.**
  - **Eg, at(loc(X, Y), *State)*,**
    facing(Direction, *State)*
- **A state can be some specific constant**
  facing(n, initialState)
- **Or a state can be the result of an action**
  facing(e, resultof(turn, initialState))

Successor-State Axioms

- **The effect of actions on properties of the world can be defined by axioms that refer to a state and its successor**

facing(Dir, resultof(Act, State)) <=>
Act = turn ∧ facing(OldDir, State) ∧
turnof (OldDir, Dir).

Effect and frame axioms

- **Effect axioms describe actions by defining their effects:**

  ∀x,s Present(x,s) ^ Portable(x) =>
  Holding(x,Result(Grab,s))

  ∀x,s ¬Holding(x,Result(Release,s))

  – **But effect axioms don't keep track of things that don't change**

- **frame axioms describe how the world stays the same:**

  – **∀act, x, s Holding(x,s) ^ ¬(act = Release) => Holding(x,Result(a,s))**

Successor state axioms

- **effect and frame axioms can be combined to form successor-state axioms :**

  ∀act,x,s Holding(x,Result(act,s)) <=>
  [(act = Grab ^ Present(x,s) ^ Portable(x))
  ∨ (Holding(x,s) ^ ¬(act= Release))]

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**
- **The key problem in FOL inference is handling universals**

  ∀x even(x) <=> ∃y x=2*y

  => even(324) <=> ∃y 324=2*y

One General-Purpose Rule

$$\frac{P_1' \quad P_2' \quad P_n' \quad P_1 \wedge P_2 \wedge \ldots P_n \Rightarrow Q}{Q'}$$

Where P's and Q's are atomic and
There is a substitution θ such that

$SUBST(\theta, P_i) = SUBST(\theta, P_i')$

$SUBST(\theta, Q) = SUBST(\theta, Q')$

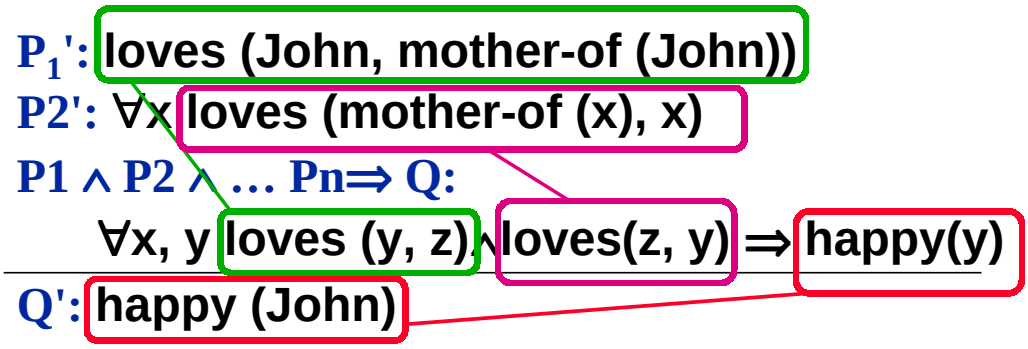
Example

John loves his mother

Everyone's mother loves them

To love one who loves you is to be happy

John is happy



P'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 will discuss two proof methods involving two different inference rules:**
  - **Backwards chaining with 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)**

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

Implicative Normal Form

$A(x) \wedge B(x) \wedge \ldots \wedge C(x) \Rightarrow D(x)$

- **Remove ¬ via DeMorgan**
- **What happened to quantifiers?**
  - **∃: replace by Skolem constant or function**

    ∃ x f(x) -> f(Sam)

    ∀ y ∃ x g(x, y) -> ∀ y ∃ x g(foo(y), y)
  - **∀: drop (implicit)**
- **What about ∨ ?**
  - **Distribute:**

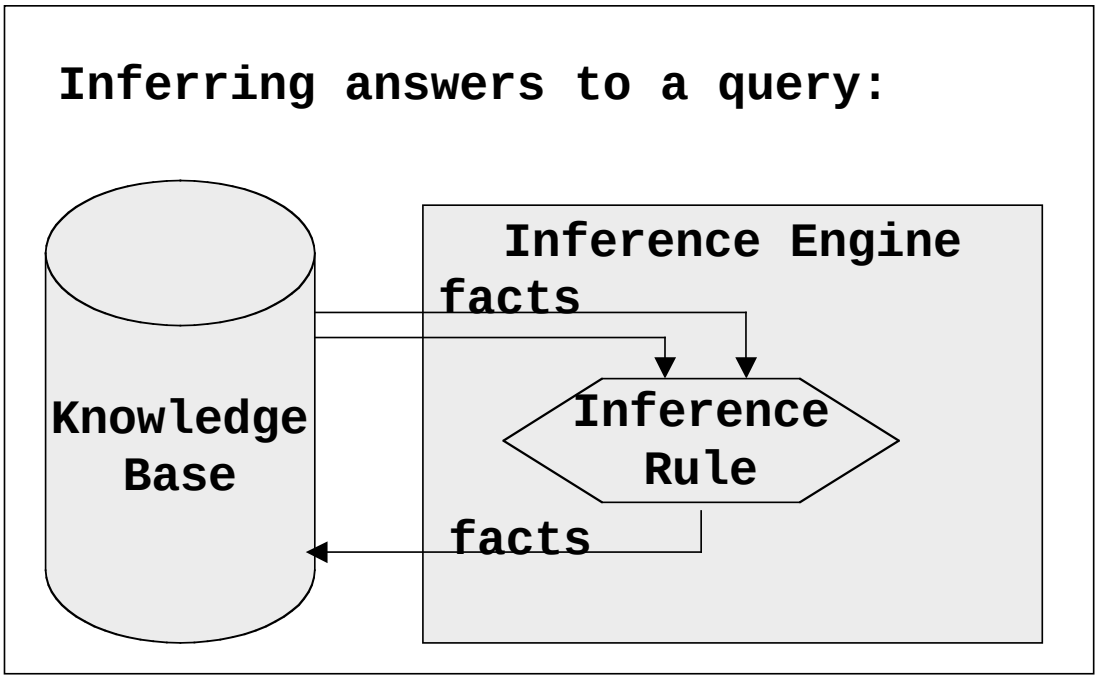
    $(A \vee B) \wedge C \Rightarrow D \rightarrow \begin{cases} A \wedge C \Rightarrow D \\ B \wedge C \Rightarrow D \end{cases}$

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

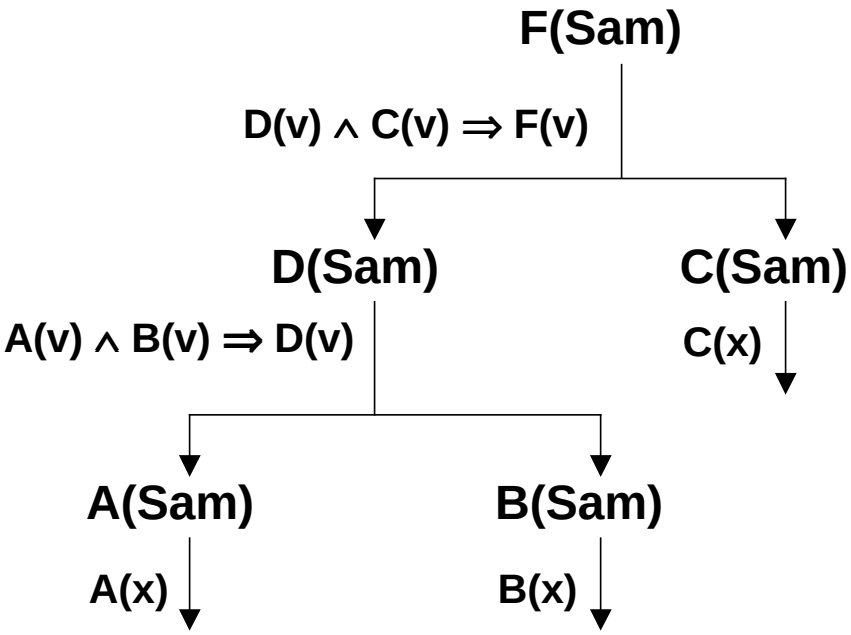
Backward:

- 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

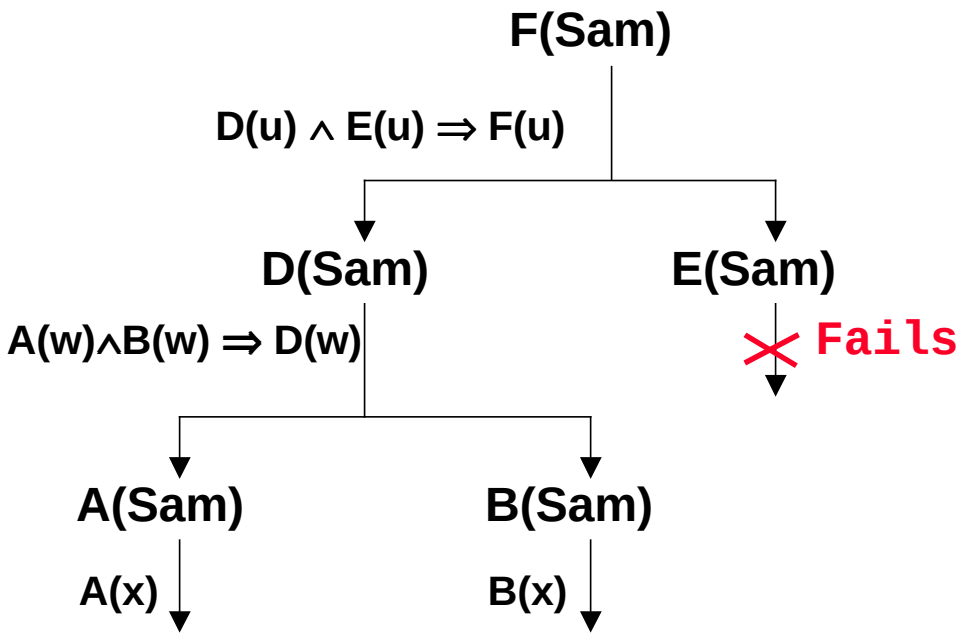


An Example With a Choice

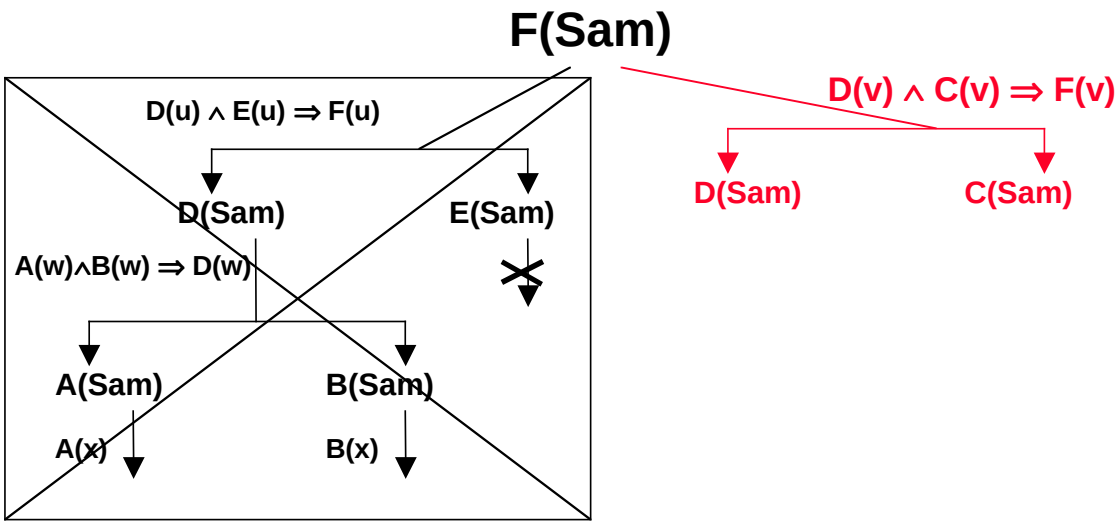
- **KB:**
  - A(x)
  - B(y)
  - C(z)
  - A(w) ∧ B(w) ⇒ D(w)
  - D(u) ∧ E(u) ⇒ F(u)
  - D(v) ∧ C(v) ⇒ F(v)

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

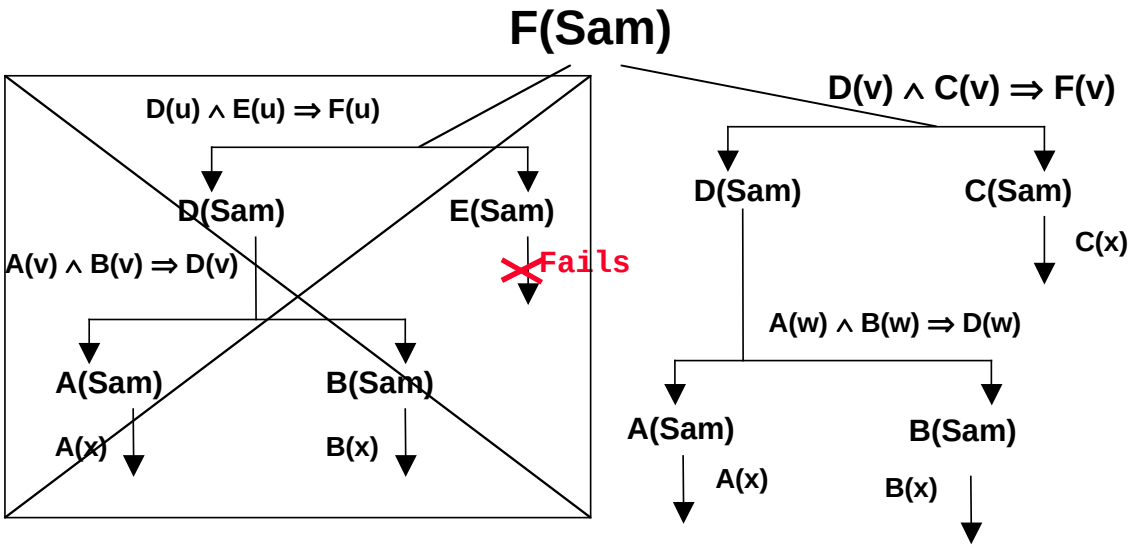
Searching for a Proof Tree



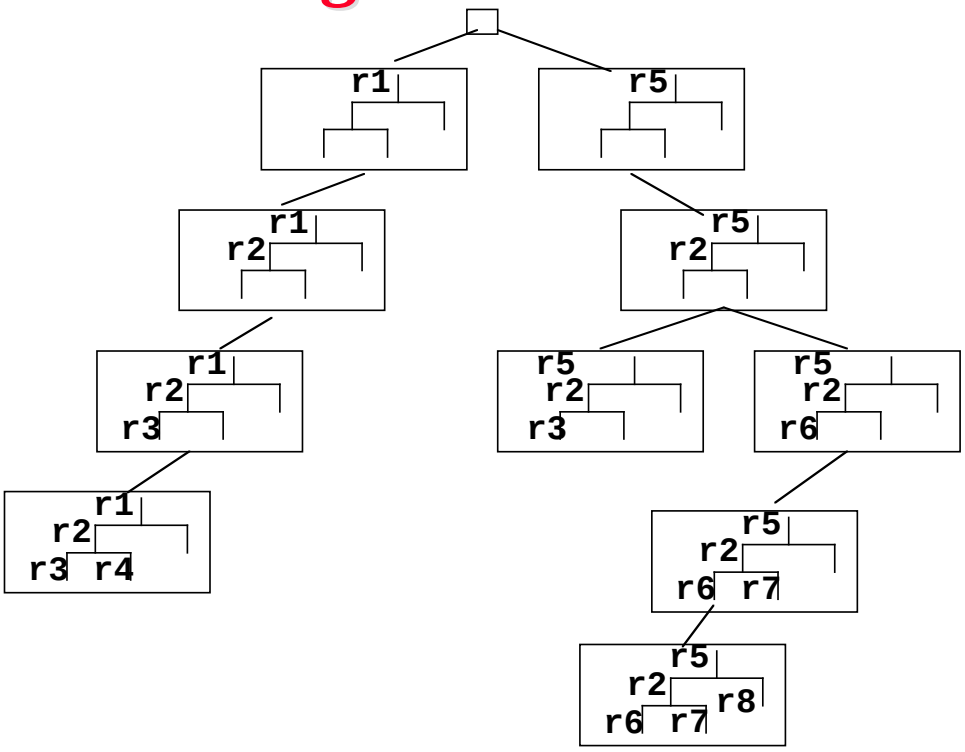
Find a Choice Point



And Continue



Searching a Tree of Search Trees



A Complete Inference Method

- **Resolution Refutation is complete for all of FOL.**
  - **Assume KB plus not(query)**
  - **Generate new sentences via resolution**
  - **Prove false**