Inference in first-order logic

Outline

- Inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution

Model checking

- Given KB, does sentence S hold?
- Basically generate and test:
  - Generate all the possible models
  - Consider the models M in which KB is TRUE
  - If $\forall$M S , then S is **provably true**
  - If $\forall$M $\neg$S, then S is **provably false**
  - Otherwise ($\exists$M1 S $\wedge$ $\exists$M2 $\neg$S): S is **satisfiable** but neither provably true or provably false

Inference rules for FOL

- Inference rules for propositional logic apply to FOL as well
  - Modus Ponens, And-Introduction, And-Elimination, …
- New (sound) inference rules for use with quantifiers:
  - Universal elimination
  - Existential introduction
  - Existential elimination
  - Generalized Modus Ponens (GMP)

Quantified inference rules

- Universal instantiation
  - $\forall x\ P(x) \therefore P(A)$
- Universal generalization
  - $P(A) \wedge P(B) \ldots \therefore \forall x\ P(x)$
- Existential instantiation
  - $\exists x\ P(x) \therefore P(F)$ $\leftarrow$ **skolem constant F**
- Existential generalization
  - $P(A) \therefore \exists x\ P(x)$

Universal instantiation (universal elimination)

- If $(\forall x)\ P(x)$ is true, then $P(C)$ is true, where C is *any* constant in the domain of x
- Example:

  $(\forall x)\ \text{eats}(\text{Ziggy}, x) \Rightarrow \text{eats}(\text{Ziggy}, \text{IceCream})$
- The variable symbol can be replaced by any ground term, i.e., any constant symbol or function symbol applied to ground terms only

Existential instantiation (existential elimination)

- From $(\exists x)\ P(x)$ infer $P(c)$
- Example:
  - $(\exists x)$ eats(Ziggy, x) $\rightarrow$ eats(Ziggy, Stuff)
- Note that the variable is replaced by a **brand-new constant** not occurring in this or any other sentence in the KB
- Also known as skolemization; constant is a **skolem constant**
- In other words, we don't want to accidentally draw other inferences about it by introducing the constant
- Convenient to use this to reason about the unknown object, rather than constantly manipulating the existential quantifier

Existential generalization (existential introduction)

- If P(c) is true, then $(\exists x)\ P(x)$ is inferred.
- Example

  $\text{eats(Ziggy, IceCream)} \Rightarrow (\exists x)\ \text{eats(Ziggy, x)}$
- All instances of the given constant symbol are replaced by the new variable symbol
- Note that the variable symbol cannot already exist anywhere in the expression

Generalized Modus Ponens (GMP)

- Apply modus ponens reasoning to generalized rules
- Combines And-Introduction, Universal-Elimination, and Modus Ponens
  - From *P(c)* and *Q(c)* and *("x)(P(x) Ù Q(x)) ® R(x)* derive *R(c)*
- General case: **Given**
  - **atomic sentences** $P_1, P_2, ..., P_N$
  - **implication sentence** $(Q_1 \wedge Q_2 \wedge ... \wedge Q_N) \rightarrow R$
    - $Q_1, ..., Q_N$ and R are atomic sentences
  - **substitution** $subst(\theta, P_i) = subst(\theta, Q_i)$ for i=1,...,N
  - **Derive new sentence: subst(θ, R)**
- Substitutions
  - subst(θ, α) denotes the result of applying a set of substitutions defined by θ to the sentence α
  - A substitution list $\theta = \{v_1/t_1, v_2/t_2, ..., v_n/t_n\}$ means to replace all occurrences of variable symbol v_i by term t_i
  - Substitutions are made in left-to-right order in the list
  - subst({x/IceCream, y/Ziggy}, eats(y,x)) = eats(Ziggy, IceCream)

Generalized Modus Ponens (GMP)

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

where $p_i'\theta = p_i\,\theta$ for all i

p_1' is *King*(*John*)
p_2' is *Greedy*(*y*)
θ is {x/John,y/John}
q θ is *Evil*(*John*)

p_1 is *King*(*x*)
p_2 is *Greedy*(*x*)
q is *Evil*(*x*)

- All variables assumed universally quantified

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	{fail}

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,
  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

- The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.
  MGU = { y/John, x/z }

Unification

- Unification is a **“pattern-matching”** procedure
  - Takes two atomic sentences, called literals, as input
  - Returns “Failure” if they do not match and a substitution list, θ, if they do
- That is, *unify(p,q) = θ* means *subst(θ, p) = subst(θ, q)* for two atomic sentences, *p* and *q*
- **θ** is called the **most general unifier** (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally quantified) variables by terms

Unification algorithm

```
procedure unify(p, q, θ)
    Scan p and q left-to-right and find the first corresponding
      terms where p and q "disagree" (i.e., p and q not equal)
    If there is no disagreement, return θ  (success!)
    Let r and s be the terms in p and q, respectively,
      where disagreement first occurs
    If variable(r) then {
      Let θ = union(θ, {r/s})
      Return unify(subst(θ, p), subst(θ, q), θ)
    } else if variable(s) then {
      Let θ = union(θ, {s/r})
      Return unify(subst(θ, p), subst(θ, q), θ)
    } else return "Failure"
  end
```

Unification: Remarks

- *Unify* is a linear-time algorithm that returns the most general unifier (mgu), i.e., the shortest-length substitution list that makes the two literals match.
- In general, there is not a **unique** minimum-length substitution list, but unify returns one of minimum length
- A variable can never be replaced by a term containing that variable

  Example: x/f(x) is illegal.
- This “occurs check” should be done in the above pseudo-code before making the recursive calls

Unification examples

- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(Bill), y)
  - {x/Bill, y/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(y), z)
  - {x/Bill, y/Bill, z/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Jane))
  - parents(Bill, father(y), mother(y))
  - Failure

The unification algorithm

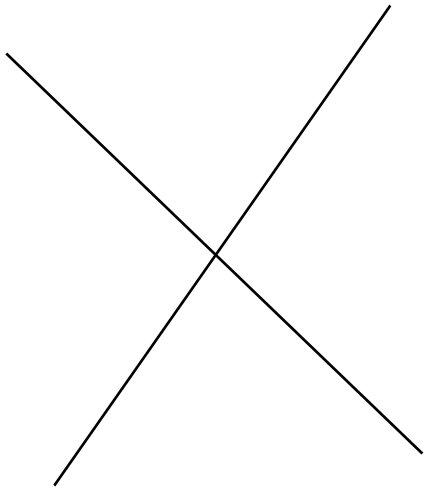
{ P[A,f(y),B], P[x,f(B),B]}	P[A,f(B),B]
{P(x),P(A)}	P(A)
{P[f(x),y,g(y)], P[f(x),z,g(x)]}	P[f(x),x,g(x)]
{P[f(x,g(A,y)),g(A,y)], P[f(x,z),z]}	P[f(x,g(A,y)),g(A,y)]

The unification algorithm

{P(x,x), P(f(z),z)}

{P(f(z), f(z)), P(f(z),z)}

{P(f(f(z)), f(f(z))), P(f(f(z)), f(z))}



Converting sentences to CNF

1. Eliminate all $\leftrightarrow$ connectives

   $(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

2. Eliminate all $\rightarrow$ connectives

   $(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate

   $\neg\neg P \Rightarrow P$

   $\neg(P \vee Q) \Rightarrow \neg P \wedge \neg Q$

   $\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$

   $\neg(\forall x)P \Rightarrow (\exists x)\neg P$

   $\neg(\exists x)P \Rightarrow (\forall x)\neg P$

4. Standardize variables: rename all variables so that each quantifier has its own unique variable name

Converting sentences to clausal form

Skolem constants and functions

5. Eliminate existential quantification by introducing Skolem constants/functions

   $(\exists x)P(x) \Rightarrow P(c)$

   c is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

   $(\forall x)(\exists y)P(x,y) \Rightarrow (\forall x)P(x, f(x))$

   since $\exists$ is within the scope of a universally quantified variable, use a **Skolem function f** to construct a new value that **depends on** the universally quantified variable

   f must be a brand-new function name not occurring in any other sentence in the KB.

   E.g., $(\forall x)(\exists y)\text{loves}(x,y) \Rightarrow (\forall x)\text{loves}(x,f(x))$

   In this case, f(x) specifies the person that x loves

Converting sentences to clausal form

6. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the “prefix” part

   Ex: $(\forall x)P(x) \Rightarrow P(x)$

7. Put into conjunctive normal form (conjunction of disjunctions) using distributive and associative laws

   $(P \wedge Q) \vee R \Rightarrow (P \vee R) \wedge (Q \vee R)$

   $(P \vee Q) \vee R \Rightarrow (P \vee Q \vee R)$

8. Split conjuncts into separate clauses
9. Standardize variables so each clause contains only variable names that do not occur in any other clause

An example

(" x)(P(x) ® ((" y)(P(y) ® P(f(x,y))) Ù Ø(" y)(Q(x,y) ® P(y))))

1. Eliminate $\rightarrow$

   $(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge \neg(\forall y)(\neg Q(x,y) \vee P(y))))$

2. Reduce scope of negation

   $(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge(\exists y)(Q(x,y) \wedge \neg P(y))))$

3. Standardize variables

   $(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge(\exists z)(Q(x,z) \wedge \neg P(z))))$

4. Eliminate existential quantification

   $(\forall x)(\neg P(x) \vee((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge(Q(x,g(x)) \wedge \neg P(g(x)))))$

5. Drop universal quantification symbols

   $(\neg P(x) \vee ((\neg P(y) \vee P(f(x,y))) \wedge(Q(x,g(x)) \wedge \neg P(g(x)))))$

Example

6. Convert to conjunction of disjunctions

$$(\neg P(x) \vee \neg P(y) \vee P(f(x,y))) \wedge (\neg P(x) \vee Q(x,g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))$$

7. Create separate clauses

$$\neg P(x) \vee \neg P(y) \vee P(f(x,y))$$

$$\neg P(x) \vee Q(x,g(x))$$

$$\neg P(x) \vee \neg P(g(x))$$

8. Standardize variables

$$\neg P(x) \vee \neg P(y) \vee P(f(x,y))$$

$$\neg P(z) \vee Q(z,g(z))$$

$$\neg P(w) \vee \neg P(g(w))$$

Conversion to CNF

- Everyone who loves all animals is loved by someone:

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

1. Eliminate biconditionals and implications

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

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

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

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

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

Conversion to CNF contd.

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

   "x [$y Animal(y) Ù ØLoves(x,y)] Ú [$z 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:

   "x [Animal(F(x)) Ù ØLoves(x,F(x))] Ú Loves(G(x),x)

5. *Drop universal quantifiers:*

   [Animal(F(x)) Ù ØLoves(x,F(x))] Ú Loves(G(x),x)

6. *Distribute Ú over Ù :*

   [Animal(F(x)) Ú Loves(G(x),x)] Ù [ØLoves(x,F(x)) Ú Loves(G(x),x)]

Skolem functions

[" "wQ(w)]É ("x){("y){($z)[P(x,y,z)É("u)R(x,y,u,z]}}
[" "wQ(w)]É ("x){("y) [P(x,y,g(x,y))É("u)R(x,y,u,g(x,y))]}

("x){ØP(x)Ú{("y)[ØP(y)ÚP(f(x,y))] Ù ($w)[Q(x,w)ÙØP(w)]}}
("x){ØP(x)Ú{("y)[ØP(y)ÚP(f(x,y))] Ù [Q(x,h(x))ÙØP(h(x)]}}

Resolution in first-order logic

- Given sentences

  $P_1 \vee ... \vee P_n$

  $Q_1 \vee ... \vee Q_m$
- in *conjunctive normal form:*
  - each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,
- if P_j and $\neg Q_k$ **unify** with substitution list θ, then derive the resolvent sentence:

  subst(θ, $P_1 \vee ... \vee P_{j-1} \vee P_{j+1} ... P_n \vee Q_1 \vee \ldots Q_{k-1} \vee Q_{k+1} \vee ... \vee Q_m$)
- Example
  - from clause — **P(x, f(a))** Ú **P(x, f(y))** Ú **Q(y)**
  - and clause — Ø**P(z, f(a))** Ú Ø**Q(z)**
  - derive resolvent — **P(z, f(y))** Ú **Q(y)** Ú Ø**Q(z)**
  - using — **θ = {x/z}**

Resolution refutation

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- **Proof by contradiction:** Add ¬Q to KB and try to prove false.

  i.e., (KB |- Q) ↔ (KB ∨ ¬Q |- False)
- Resolution is **refutation complete:** it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences
- Also, it cannot be used to prove that Q is **not entailed** by KB.
- Resolution **won't always give an answer** since entailment is only semidecidable
  - And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one

Example

(" x,y){[Package(x)ÙPackage(y)ÙInroom(x,27)ÙInroom(y,28)]ÉSmaller(x,y)}

P(A)

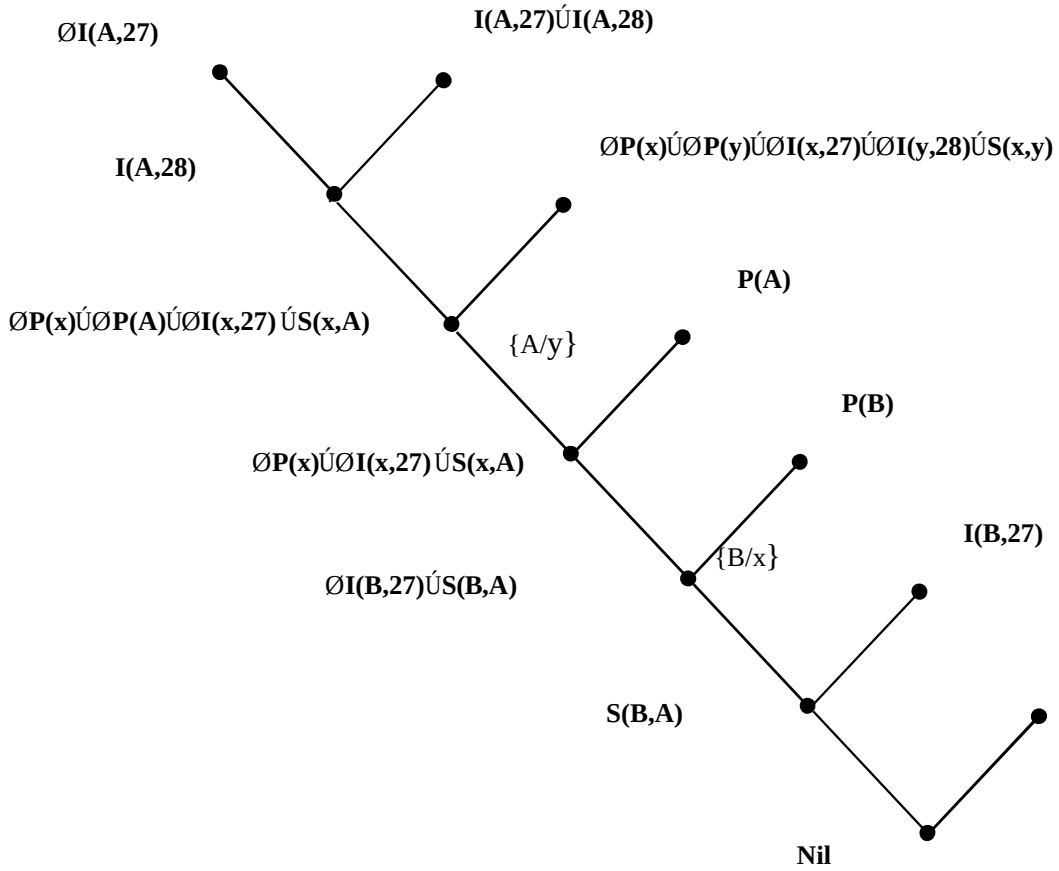
P(B)

I(A,27)ÚI(A,28)

ØS(B,A)

Prove I(A,27)

Example

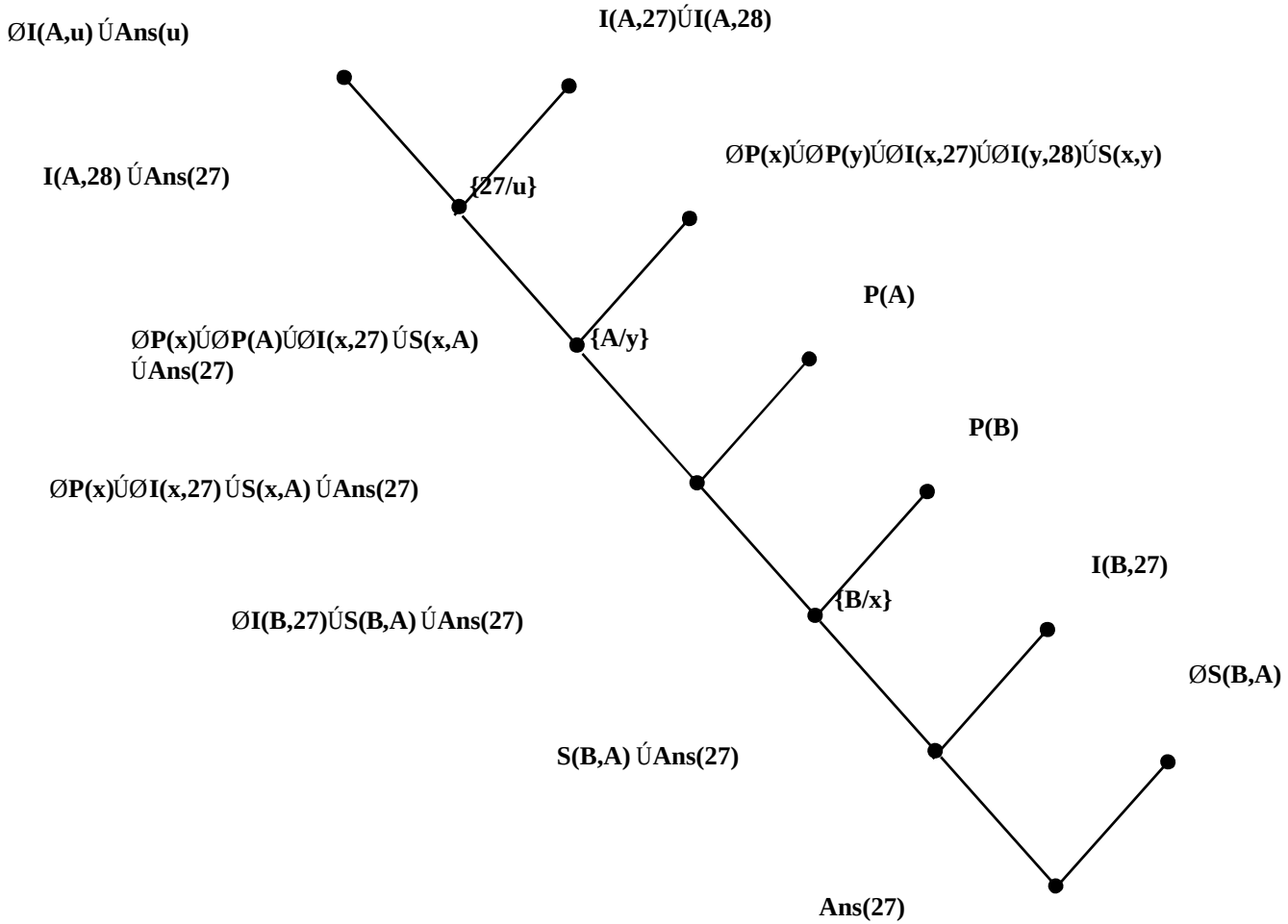

ØI(A,27)
I(A,27)ÚI(A,28)
I(A,28)
ØP(x)ÚØP(y)ÚØI(x,27)ÚØI(y,28)ÚS(x,y)
P(A)
ØP(x)ÚØP(A)ÚØI(x,27) ÚS(x,A)
{A/y}
P(B)
ØP(x)ÚØI(x,27) ÚS(x,A)
I(B,27)
{B/x}
ØI(B,27)ÚS(B,A)
S(B,A)
Nil

Answer Extraction

($u) I(A,u)

ØI(A,u) ÚAns(u)

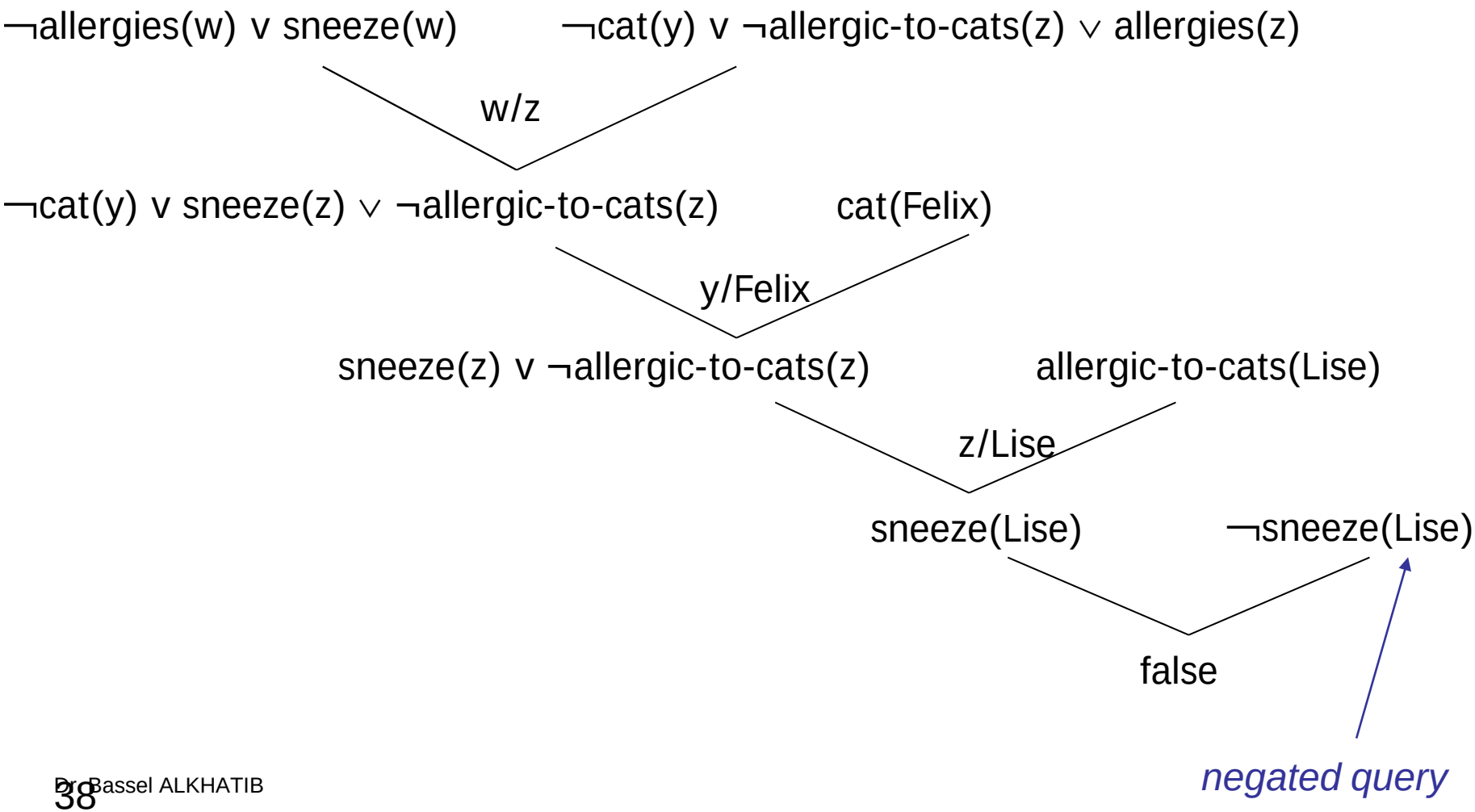
Answer Extraction



Example

- KB:
  - allergies(X) $\rightarrow$ sneeze(X)
  - cat(Y) $\wedge$ allergic-to-cats(X) $\rightarrow$ allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)
- Goal:
  - sneeze(Lise)

Refutation resolution proof tree



Practice example

Did Curiosity kill the cat

- 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?

Practice example

Did Curiosity kill the cat

- 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?
- These can be represented as follows:

  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. $(\forall x)\ Cat(x) \rightarrow Animal(x)$

  G. $Kills(Curiosity, Tuna)$ ⟵ GOAL

- **Convert to clause form**

  A1. (Dog(D)) ← D is a skolem constant

  A2. (Owns(Jack,D))

  B. (¬Dog(y), ¬Owns(x, y), AnimalLover(x))

  C. (¬AnimalLover(a), ¬Animal(b), ¬Kills(a,b))

  D. (Kills(Jack,Tuna), Kills(Curiosity,Tuna))

  E. Cat(Tuna)

  F. (¬Cat(z), Animal(z))

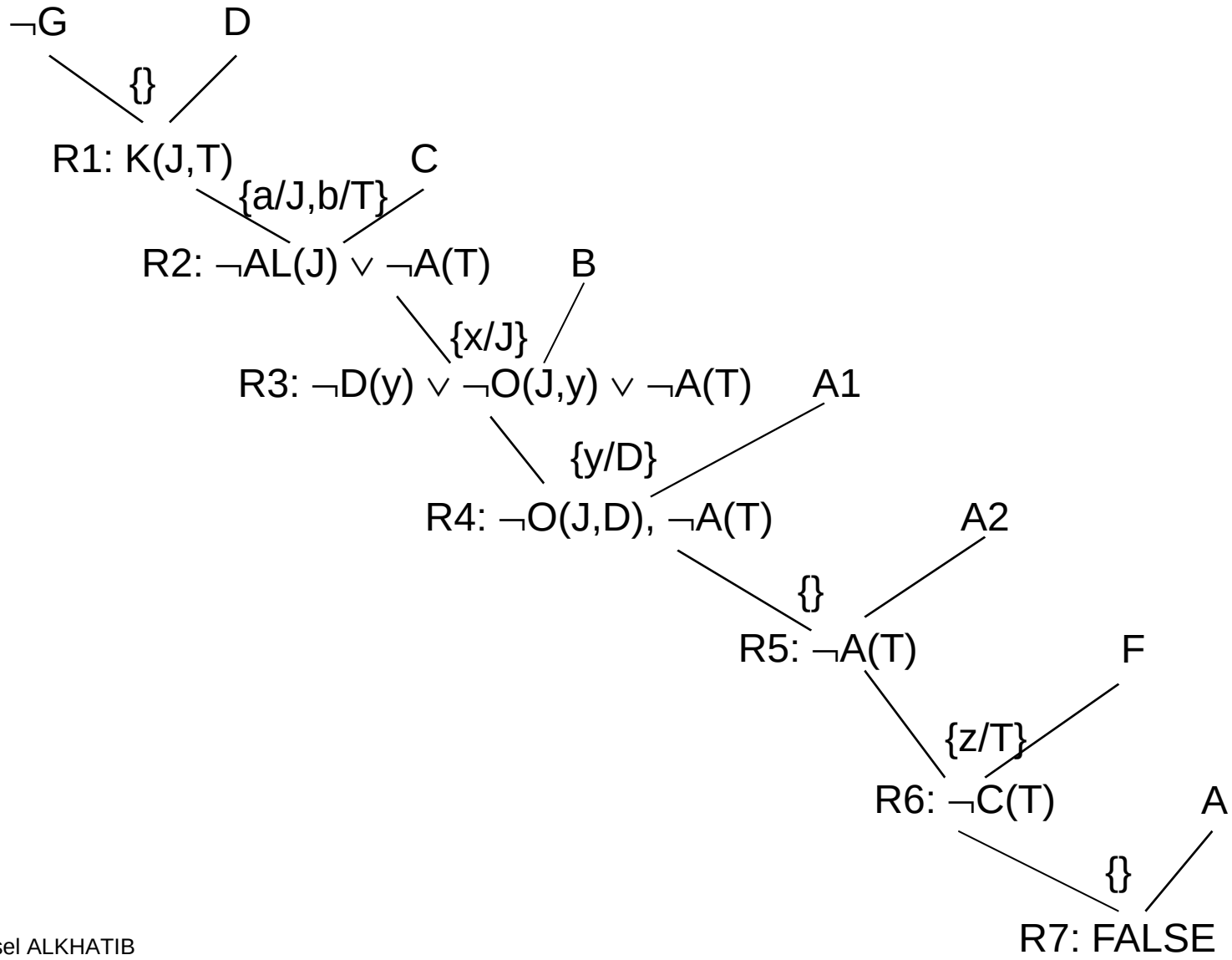
- **Add the negation of query:**

  ¬G: (¬Kills(Curiosity, Tuna))

- **The resolution refutation proof**

R1: ¬G, D, {}	(Kills(Jack, Tuna))
R2: R1, C, {a/Jack, b/Tuna}	(~AnimalLover(Jack), ~Animal(Tuna))
R3: R2, B, {x/Jack}	(~Dog(y), ~Owns(Jack, y),~Animal(Tuna))
R4: R3, A1, {y/D}	(~Owns(Jack, D), ~Animal(Tuna))
R5: R4, A2, {}	(~Animal(Tuna))
R6: R5, F, {z/Tuna}	(~Cat(Tuna))
R7: R6, E, {}	FALSE

- **The proof tree**



Exercise

Convert to CNF

(" x){ØP(x)Ú{(" y)[ØP(y)ÚP(f(x,y))] Ù ($w)[Q(x,w)ÙØP(w)]}}

Exercise

- **ØP(x1)ÚØP(y)ÚP[f(x1,y)]**
- **ØP(x2)ÚQ[x2,h(x2)]**
- **ØP(x3)ÚØP[h(x3)]**

Example knowledge base

- 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 knowledge base

... it is a crime for an American to sell weapons to hostile nations:

American(x) Ù Weapon(y) Ù Sells(x,y,z) Ù Hostile(z) Þ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M1) Ù Missile(M1)

… all of its missiles were sold to it by Colonel West

Missile(x) Ù Owns(Nono,x) Þ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) Þ Weapon(x)

An enemy of America counts as "hostile“:

Enemy(x,America) Þ Hostile(x)

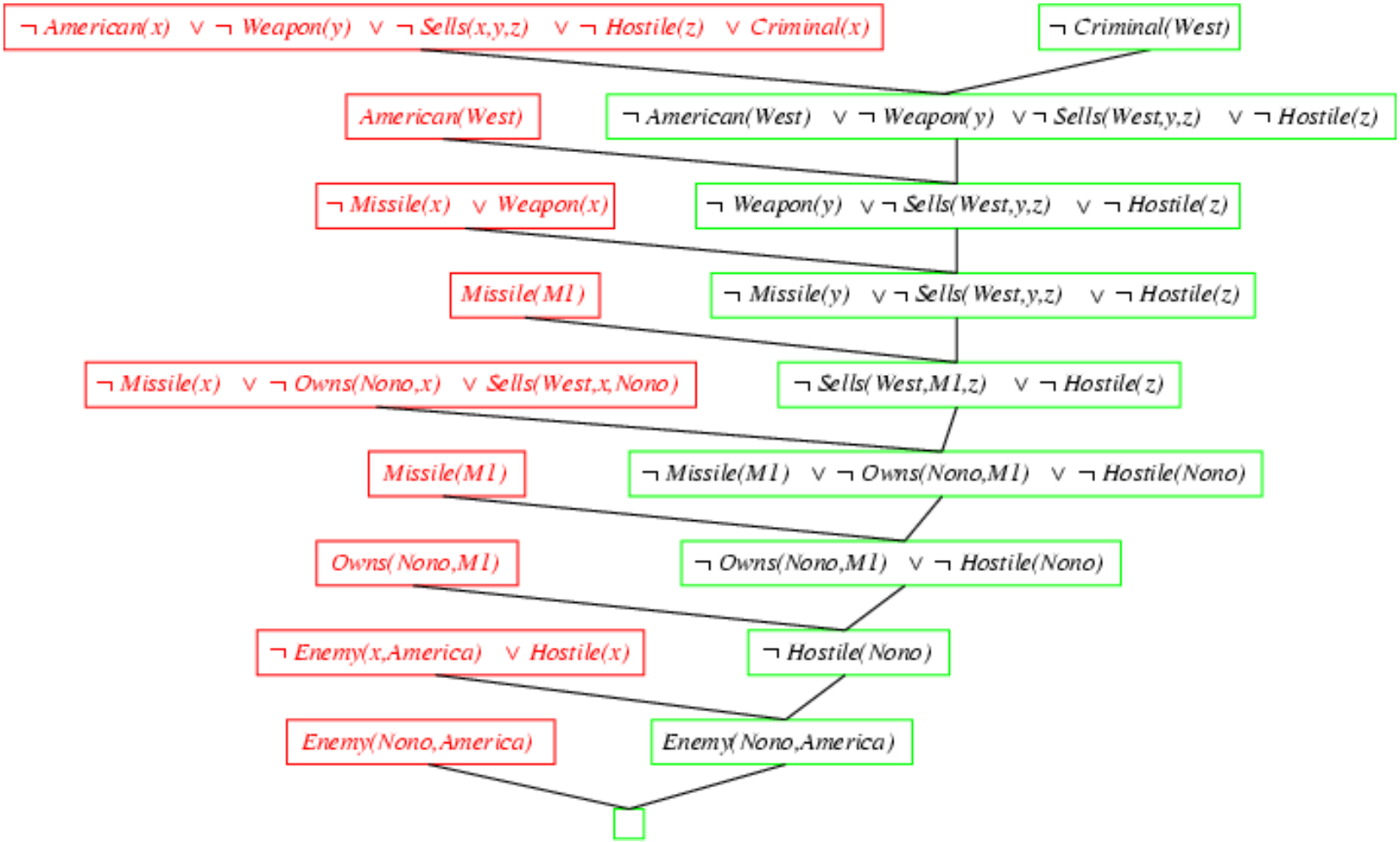
West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Resolution proof: definite clauses

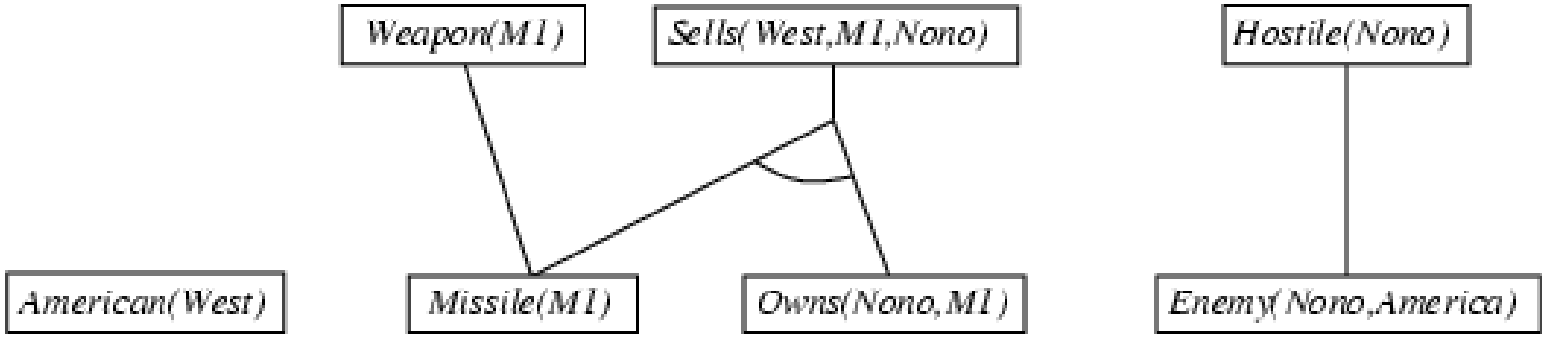


Forward chaining

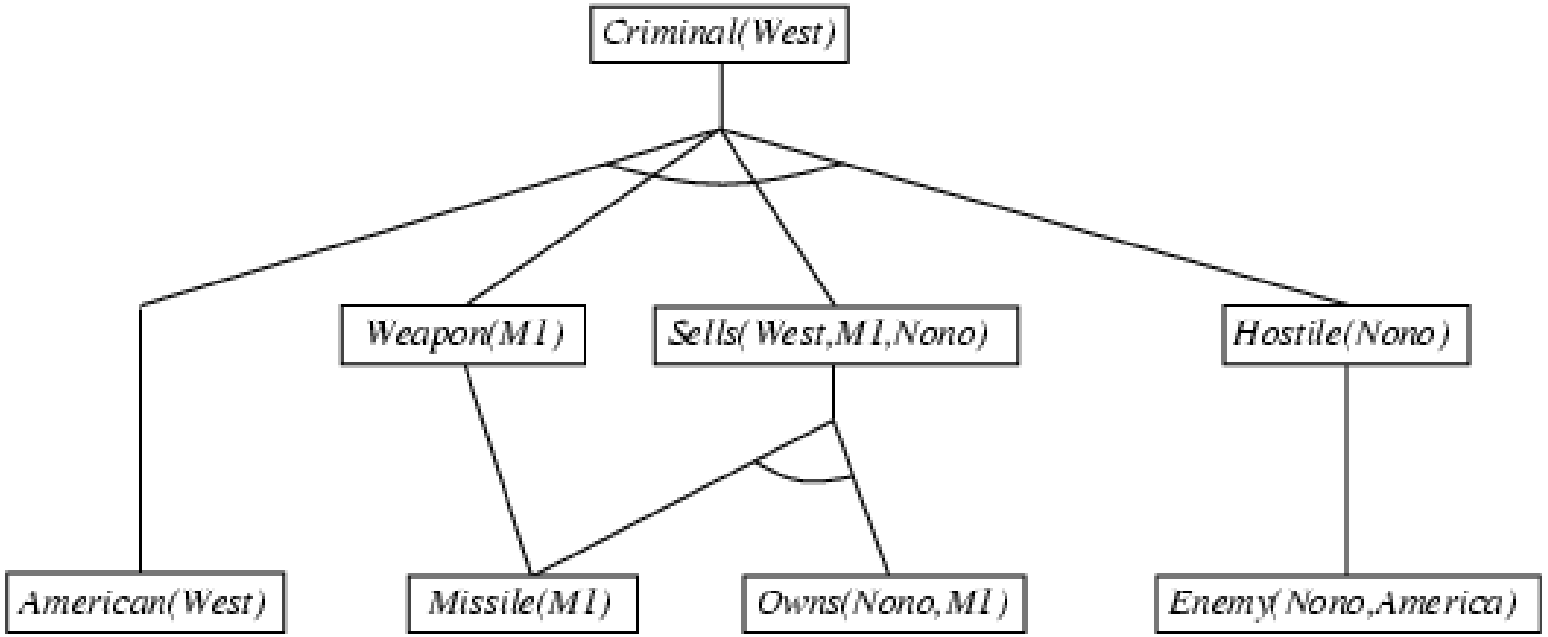
- Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived
- This defines a **forward-chaining** inference procedure because it moves “forward” from the KB to the goal [eventually]
- Inference using GMP is **complete** for KBs containing **only Horn clauses**

Forward chaining proof

Forward chaining proof



Forward chaining proof

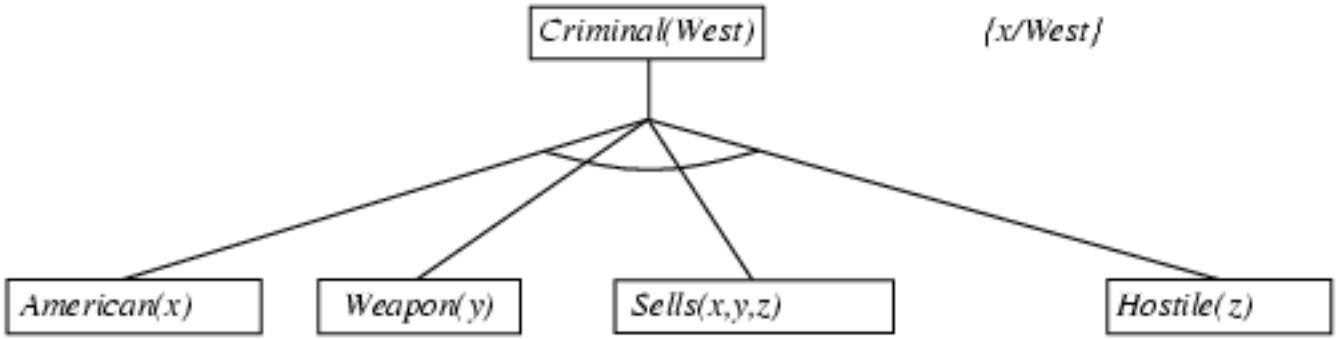


Properties of forward chaining

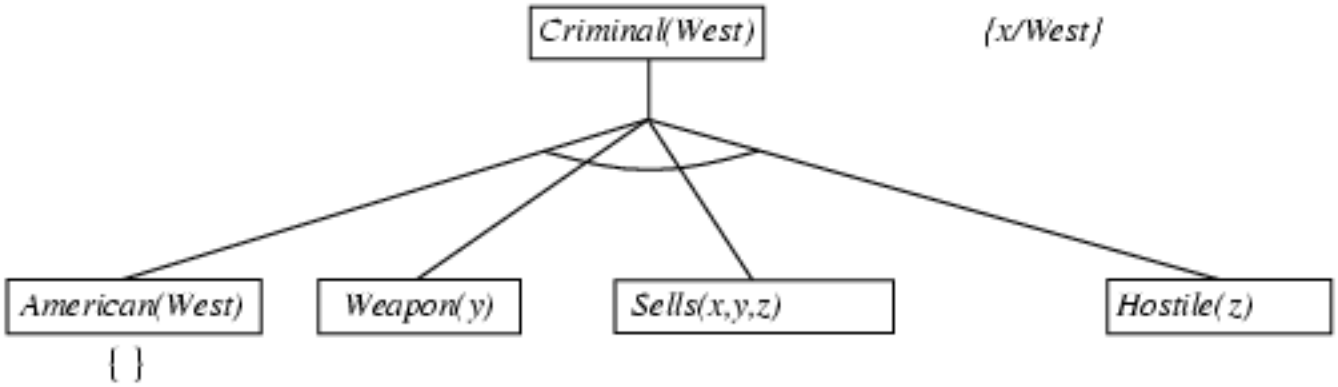
- Sound and complete for first-order definite clauses

Backward chaining example

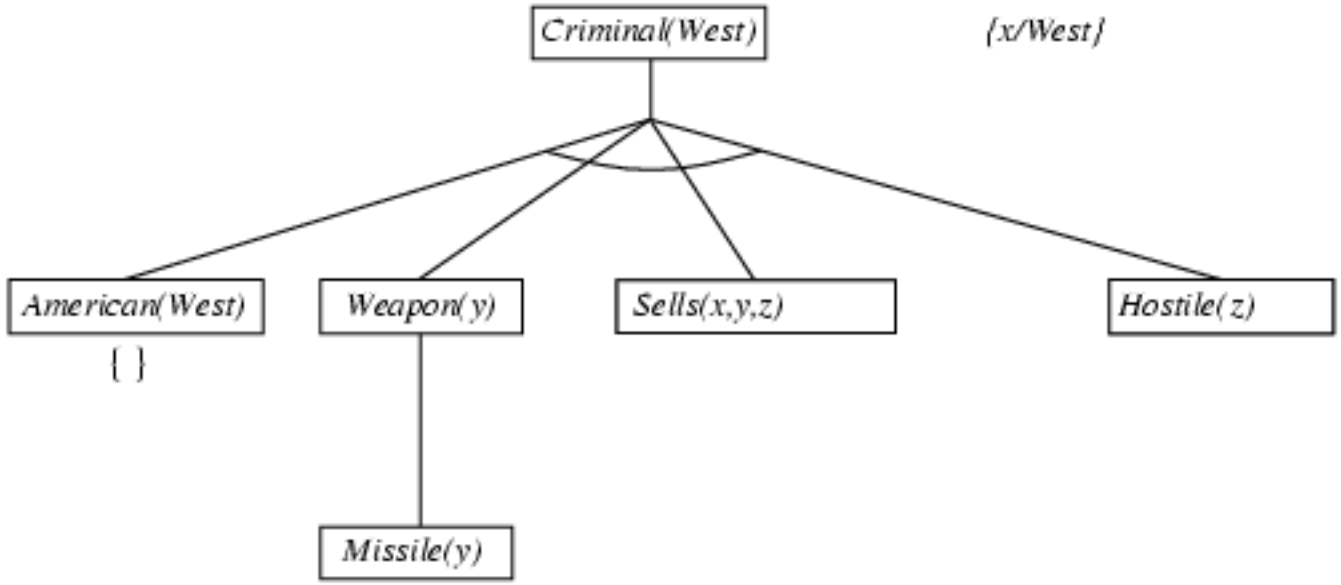
Backward chaining example



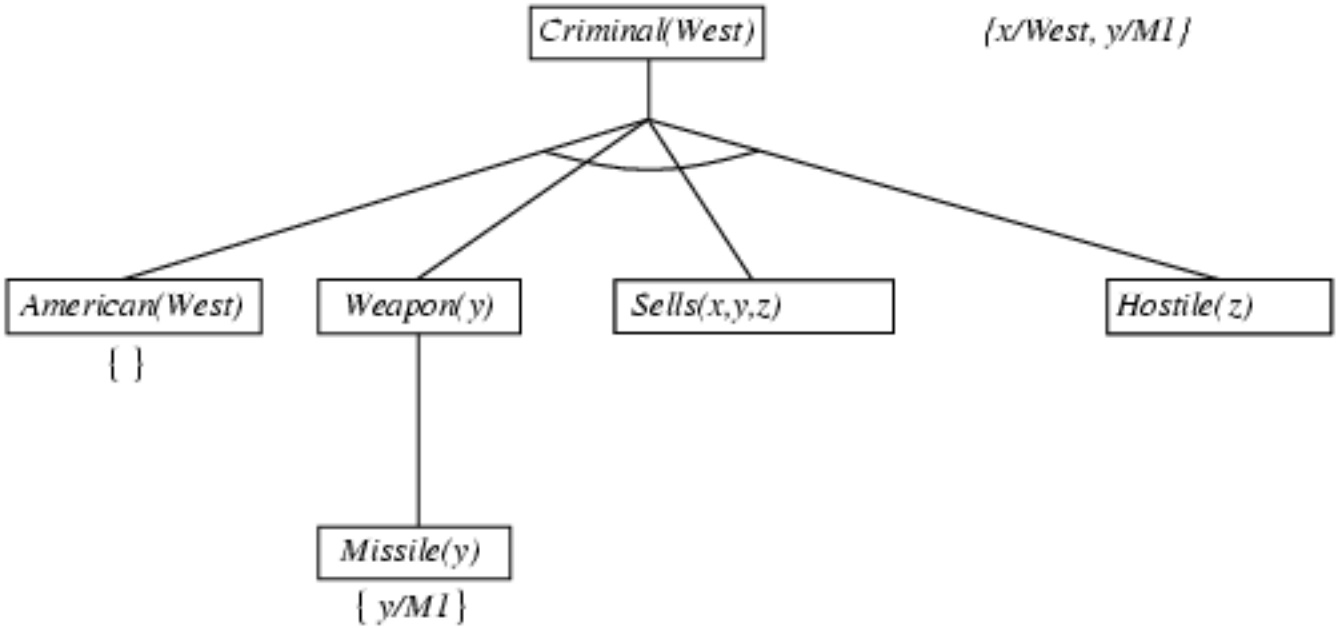
Backward chaining example



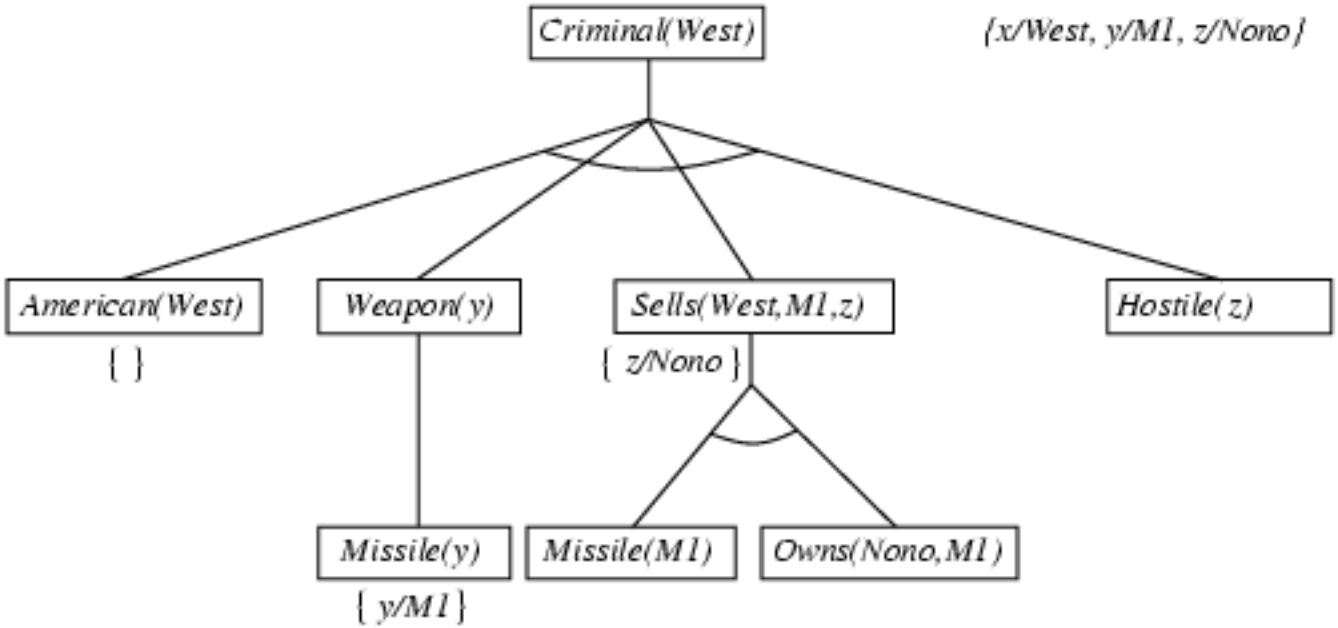
Backward chaining example



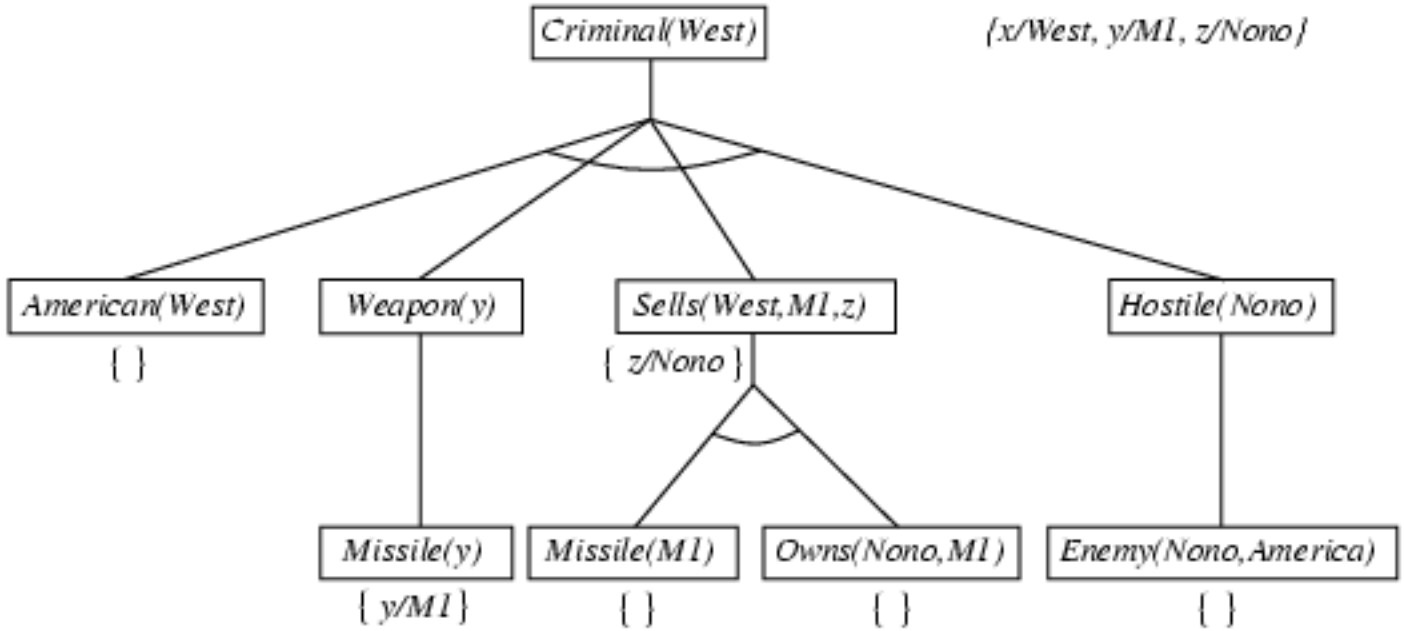
Backward chaining example



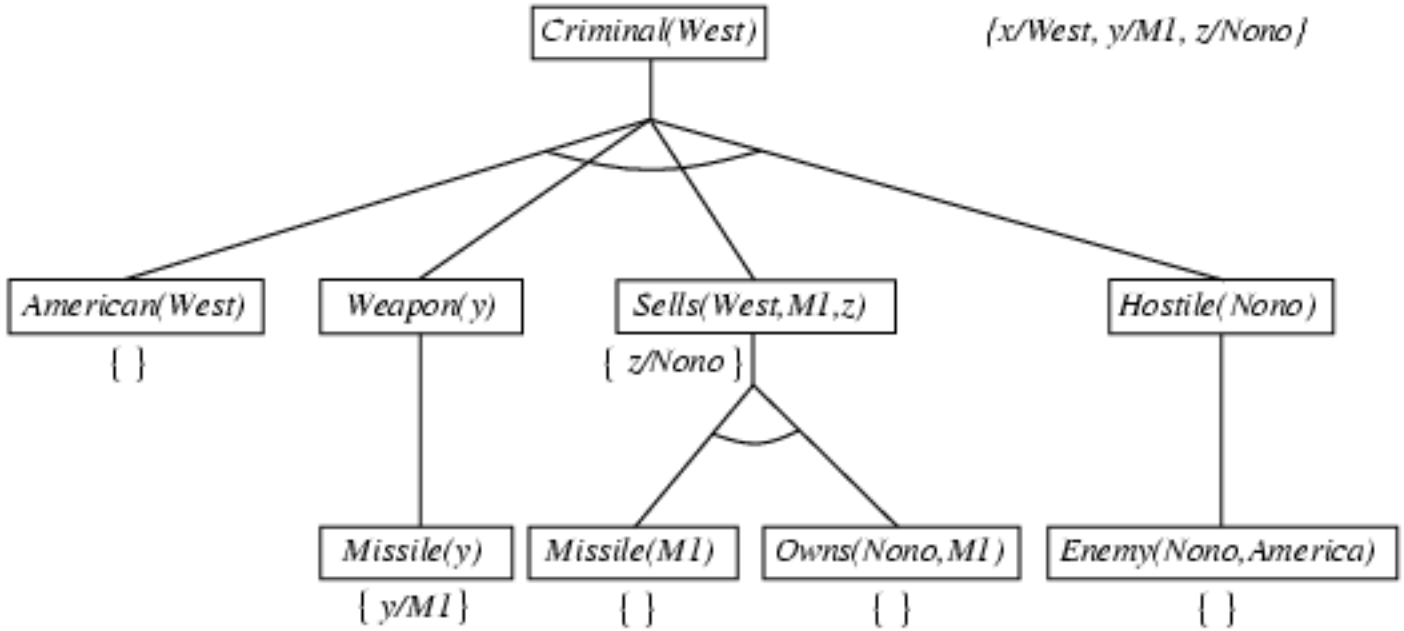
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming

Forward vs. backward chaining

- FC is data-driven
  - Automatic, unconscious processing
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving
  - Where are my keys? How do I get to my next class?
  - Complexity of BC can be much less than linear in the size of the KB

Horn clauses

- A Horn clause is a sentence of the form:

  $(\forall x)\ P_1(x) \wedge P_2(x) \wedge \ldots \wedge P_n(x) \rightarrow Q(x)$

where

  - there are 0 or more P_is and 0 or 1 Q
  - the P_is and Q are positive (i.e., non-negated) literals

- Equivalently: $P_1(x) \acute{U} P_2(x) \ldots \acute{U} P_n(x)$ where the P_i are all atomic and *at most one* of them is positive
- Prolog is based on Horn clauses
- Horn clauses represent a *subset* of the set of sentences representable in FOL

Prolog

- A logic programming language based on Horn clauses
  - Resolution refutation
  - Control strategy: goal-directed and depth-first
    - always start from the goal clause
    - always use the new resolvent as one of the parent clauses for resolution
    - backtracking when the current thread fails
    - complete for Horn clause KB
  - Support answer extraction (can request single or all answers)
  - Orders the clauses and literals within a clause to resolve non-determinism
    - Q(a) may match both Q(x) <= P(x) and Q(y) <= R(y)
    - A (sub)goal clause may contain more than one literals, i.e., <= P1(a), P2(a)
  - Use “closed world” assumption (negation as failure)
    - If it fails to derive P(a), then assume ~P(a)

Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses
  Widely used in Europe, Japan (basis of 5th Generation project)
- Program = set of clauses = `head :- literal`$_1$`, … literal`$_n$.

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- Depth-first, left-to-right backward chaining

Examples & Exercises

Example

- Example: KB = All cats like fish, cats eat everything they like, and Ziggy is a cat. In FOL, KB =

1. $(\forall x)\ cat(x) \Rightarrow likes(x, Fish)$
2. $(\forall x)(\forall y)\ (cat(x) \wedge likes(x,y)) \Rightarrow eats(x,y)$
3. $cat(Ziggy)$

- Goal query: Does Ziggy eat fish?

Proof:

1. Use GMP with (1) and (3) to derive: 4. $likes(Ziggy, Fish)$
2. Use GMP with (3), (4) and (2) to derive $eats(Ziggy, Fish)$
3. So, Yes, Ziggy eats fish.

Unification…

- Examples

Literal 1	Literal 2	Literal 3
`parents(x, father(x), mother(Bill))`	`parents(Bill, father(Bill), y)`	`{x/Bill, y/mother(Bill)}`
`parents(x, father(x), mother(Bill))`	`parents(Bill, father(y), z)`	`{x/Bill, y/Bill, z/mother(Bill)}`
`parents(x, father(x), mother(Jane))`	`parents(Bill, father(y), mother(y))`	`Failure`

Resolution example (using PL sentences)

- From “Heads I win, tails you lose” prove that “I win”
- First, define the axioms in KB:
  1. "Heads I win, tails you lose."
     `(Heads => IWin)` or, equivalently, **`(~Heads v IWin)`**
     `(Tails => YouLose)` or, equivalently, **`(~Tails v YouLose)`**
  2. Add some general knowledge axioms about coins, winning, and losing:
     `(Heads v Tails)`
     `(YouLose => IWin)` or, equivalently, **`(~YouLose v IWin)`**
     `(IWin => YouLose)` or, equivalently, **`(~IWin v YouLose)`**

- Goal: **`IWin`**

Resolution example (using PL sentences)…

Sentence 1	Sentence 2	Resolvent
~IWin	~Heads v IWin	~Heads
~Heads	Heads v Tails	Tails
Tails	~Tails v YouLose	YouLose
YouLose	~YouLose v Iwin	IWin
IWin	~IWin	False

Converting FOL sentences to clause form…

Examples:

- Convert the sentence

(∀ x)(P(x) => ((∀ y)(P(y) ⇒ P(f(x,y))) ^ ¬(∀ y)(Q(x,y) => P(y))))

1. Eliminate **<=>**
   Nothing to do here.

2. Eliminate ⇒
   (∀ x)(¬ P(x) v ((∀ y)(¬ P(y) v P(f(x,y))) ^ ¬(∀ y)(¬ Q(x,y) v P(y))))

3. Reduce scope of negation
   (∀ x)(¬ P(x) v ((∀ y)(¬ P(y) v P(f(x,y))) ^ (Ey)(Q(x,y) ^ ¬ P(y))))

Converting FOL sentences to clause form…

4. Standardize variables
   (∀ x)(¬ P(x) v ((∀ y)(¬ P(y) v P(f(x,y))) ^ (Ez)(Q(x,z) ^ ¬ P(z))))

5. Eliminate existential quantification
   (∀ x)(¬ P(x) v ((∀ y)(¬ P(y) v P(f(x,y))) ^ (Q(x,g(x)) ^ ¬ P(g(x)))))

6. Drop universal quantification symbols
   (¬ P(x) v ((¬ P(y) v P(f(x,y))) ^ (Q(x,g(x)) ^ ¬ P(g(x)))))

7. Convert to conjunction of disjunctions
   (¬ P(x) v ¬ P(y) v P(f(x,y))) ^ (¬ P(x) v Q(x,g(x))) ^ (¬ P(x) v ¬ P(g(x)))

Converting FOL sentences to clause form…

8. Create separate clauses
   - ¬ **P(x) v** ¬ **P(y) v P(f(x,y))**
   - ¬ **P(x) v Q(x,g(x))**
   - ¬ **P(x) v** ¬ **P(g(x))**

9. Standardize variables
   - ¬ **P(x) v** ¬ **P(y) v P(f(x,y))**
   - ¬ **P(z) v Q(z,g(z))**
   - ¬ **P(w) v** ¬ **P(g(w))**

Example: Hoofers Club

- **Problem Statement:** Tony, Shi-Kuo and Ellen belong to the Hoofers Club. Every member of the Hoofers Club is either a skier or a mountain climber or both. No mountain climber likes rain, and all skiers like snow. Ellen dislikes whatever Tony likes and likes whatever Tony dislikes. Tony likes rain and snow.

- **Query:** Is there a member of the Hoofers Club who is a mountain climber but not a skier?

Example: Hoofers Club…

- **Translation into FOL Sentences**
- Let **S(x)** mean **x** is a skier, **M(x)** mean x is a mountain climber, and **L(x,y)** mean **x** likes **y**, where the domain of the first variable is Hoofers Club members, and the domain of the second variable is snow and rain. We can now translate the above English sentences into the following FOL wffs:

1. **(Ax) S(x) v M(x)**
2. **¬(Ex) M(x) ^ L(x, Rain)**
3. **(Ax) S(x) => L(x, Snow)**
4. **(Ay) L(Ellen, y) <=> ¬ L(Tony, y)**
5. **L(Tony, Rain)**
6. **L(Tony, Snow)**

7. **Query: (Ex) M(x) ^ ¬ S(x)**
8. **Negation of the Query: ¬(Ex) M(x) ^ ¬ S(x)**

Example: Hoofers Club…

- **Conversion to Clause Form**

1. **S(x1) v M(x1)**
2. **¬ M(x2) v ¬ L(x2, Rain)**
3. **¬ S(x3) v L(x3, Snow)**
4. **¬ L(Tony, x4) v ¬ L(Ellen, x4)**
5. **L(Tony, x5) v L(Ellen, x5)**
6. **L(Tony, Rain)**
7. **L(Tony, Snow)**
8. **Negation of the Query: ¬ M(x7) v S(x7)**

Example: Hoofers Club…

- **Resolution Refutation Proof**

Clause 1	Clause 2	Resolvent	MGU (i.e., Theta)
8	1	9. S(x1)	{x7/x1}
9	3	10. L(x1, Snow)	{x3/x1}
10	4	11. ~L(Tony, Snow)	{x4/Snow, x1/Ellen}
11	7	12. False	{}

Example: Hoofers Club…

- **Answer Extraction**

Clause 1	Clause 2	Resolvent	MGU (i.e., Theta)
~M(x7) v S(x7) v (M(x7) ^ ~S(x7))	1	9. S(x1) v (M(x1) ^ ~S(x1))	{x7/x1}
9	3	10. L(x1, Snow) v (M(x1) ^ ~S(x1))	{x3/x1}
10	4	11. ~L(Tony, Snow) v (M(Ellen) ^ ~S(Ellen))	{x4/Snow, x1/Ellen}
11	7	12. M(Ellen) ^ ~S(Ellen)	{}

- **Answer to the query:** Ellen!

Example

We know the following facts about some elephants:

- Sam is pink.
- Clyde is gray and likes Oscar.
- Oscar is either pink or gray and likes Sam.

who is the gray elephant who likes a pink elephant ?

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - Syntax: formal structure of sentences
  - Semantics: truth of sentences wrt models
  - Entailment: necessary truth of one sentence given another
  - Inference: deriving sentences from other sentences
  - Soundness: derivations produce only entailed sentences
  - Completeness: derivations can produce all entailed sentences
- FC and BC are linear time, complete for Horn clauses
- Resolution is a sound and complete inference method for propositional and first-order logic