Inference in first-order logic

CS4881 Aritificial Intelligence

Jay Urbain

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution

Difficulty of FOL Inference

- Inference in propositional logic (PL) is relatively easy
  – Enumerate all possibilities (truth tables)
  – Apply sound inference rules on facts

- But in FOL we have concepts and variables, *relations, and quantification*
  – This complicates things - *a lot*

- First, let's see how we can convert FOL into propositional logic, then use PL inference.
  – *And see why this is valid*

Convert (Reduce) FOL into PL

Get rid of quantifiers

- Universal instantiation (UI) (Universal Elimination)
- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\ \alpha}{\text{Subst}(\{v/g\},\ \alpha)}$$

Variable v substituted with ground term g from KB

- E.g., $\forall x$ *Eats(Bob, x)* ***infer*** *Eats(Bob, pizza), Eats(Bob, donughts), Eats(Bob, books), …*

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

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

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

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

Convert FOL into PL

- Existential instantiation (EI) (Existential Elimination)
  - Variable substituted for new term (Skolem constant).

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

- E.g., ∃ x *Eats(Bob, x)* **infer** Eats(Bob, NewFood)

- Why do we need a new term?
  - Bob eats something and Tyler eats something, but they don't eat the *same* thing, i.e.,
  - ∃ x *Eats(Bob, x)* ^ ∃ x *Eats(Tyler, x)*

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields: $Crown(C_1) \wedge OnHead(C_1,John)$

  provided C_1 is a new constant symbol, called a Skolem constant

Convert FOL into PL

- Basic idea: use substitution, treat ground terms like propositional symbols.

Translate English to FOL

- Tom is a turtle
  - turtle(Tom)
- Rob is a rabbit
  - rabbit(Rob)
- Turtles outlast rabbits
  - $\forall$x,y turtle(*x*) ^ rabbit(y) $\Rightarrow$ outlasts(x,y)

- *Now proove Tom outlasts Rob!*

Convert FOL into PL

Proof: Tom outlasts Rob

- Logical *AND* introduction:
  - turtle(x) ^ rabbit(y)

- Universal elimination {x/Tom, y/Rob}, *among millions of other things in KB*
  - turtle*(Tom*) ^ rabbit(Rob) ⇒ outlasts(Tom,Rob)

- Modus Ponens
  - outlasts(Tom,Rob)

- *Does not seem very efficient*
  - *Need more powerful inference rules for FOL*
  - *Why bother with the million things?*

Generalized Modus Ponens (GMP)

- Unify the rule premise with known facts and apply unifier to conclusion
  - *I.e., Find a substitution that makes the rule premise match known facts.*

- Rule: $\forall$x,y turtle(*x*) ^ rabbit(y) $\Rightarrow$ outlasts(x,y)
  - Known facts: turtle(Tom) ^ rabbit(Rob)
  - Unifier: {x/Tom, y/Rob}

- Apply unifier to conclusion:
  - outlasts(Tom, Rob)

Generalized Modus Ponens (GMP)

- Unify rule premise with known facts

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

Where $SUBST(\theta, p_i') = SUBST(\theta, p_i)$ for all i

- SUBST(θ,α) just means apply the substitutions contained witin θ to sentence α.

- Substitution list $\theta = \{v_1/t_1, v_2/t_2, \ldots, v_n/t_n\}$ means
  - Replace all occurrences of variable v_i with term t_i
  - Substitutions are made in left to right order

- All variables are assumed to be universally quantified
- Used with a KB in Horn normal form, i.e., KB of definite clauses (exactly one positive literal)

Generalized Modus Ponens (GMP)

- Unify rule premise with known facts
  - *I.e., Find a substitution that makes the rule premise match known facts.*

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

Where $SUBST(\theta, p_i') = SUBST(\theta, p_i)$ for all *I*

Example:

- *p1′* = taller(Larry,Curly)
- *p2′* = taller(Curly,Moe)
- *p1* ^ *p2* ⇒ *q* = taller(x,y) ^ taller(y,z) ⇒ taller(x,z)
- *θ* = {x/Larry, y/Curly, z/Moe}
- *SUBST(θ,q)* = taller(Larry,Moe)

Reduction to propositional inference

Suppose the KB contains just the following:

$\forall$x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
King(John), King(Richard)
Greedy(John), Greedy(Richard)
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), King(Richard)
  Greedy(John), Greedy(Richard)
  Brother(Richard,John)

- The new KB is propositionalized: proposition symbols are

  King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction contd.

- 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, return result.

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

Reduction cont'd.

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

Idea: For *n* = 0 to ∞ do

create a propositional KB by instantiating with depth-n terms

see if α is entailed by this 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.)

Unification

Substitution *θ* is said to unify *p* and *q* if *SUBST(θ, p)* = *SUBST(θ, q)*

e.g., 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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

Substitution *θ* is said to unify *p* and *q* if *SUBST(θ, p) = SUBST(θ, q)*

e.g., 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)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

Substitution θ is said to unify *p* and *q* if *SUBST(θ, p) = SUBST(θ, q)*

e.g., 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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

Substitution θ is said to unify p and q if $SUBST(\theta, p) = SUBST(\theta, q)$

e.g., 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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

Substitution *θ* is said to unify *p* and *q* if *SUBST(θ, p) = SUBST(θ, q)*

e.g., 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}

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

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 }

- MGU is unique up to renaming of variables

- Cannot unify if a variable itself occurs in the other term: UNIFY(x, f(x)) = FAIL

- Cannot unify different ground terms: UNIFY(Nik, Ryan)

Completeness of FOL inference

- Truth table enumeration is incomplete for FOL
  - Table may be of infinite size
- Natural deduction complete for FOL
  - Impractical: branching factor too large
- GMP is sound
- GMP is incomplete for FOL
- Not all sentences can be converted to Horn clauses
- GMP is complete for FOL kB in HNF
  - Forward chaining
  - Bakcward chaining

The unification algorithm

function $\textsc{Unify}(x, y, \theta)$ **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
    y, a variable, constant, list, or compound
    θ, the substitution built up so far

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if $\textsc{Variable?}(x)$ **then return** $\textsc{Unify-Var}(x, y, \theta)$
  else if $\textsc{Variable?}(y)$ **then return** $\textsc{Unify-Var}(y, x, \theta)$
  else if $\textsc{Compound?}(x)$ **and** $\textsc{Compound?}(y)$ **then**
    return $\textsc{Unify}(\textsc{Args}[x], \textsc{Args}[y], \textsc{Unify}(\textsc{Op}[x], \textsc{Op}[y], \theta))$
  else if $\textsc{List?}(x)$ **and** $\textsc{List?}(y)$ **then**
    return $\textsc{Unify}(\textsc{Rest}[x], \textsc{Rest}[y], \textsc{Unify}(\textsc{First}[x], \textsc{First}[y], \theta))$
  else return failure

The unification algorithm

function UNIFY-VAR(var, x, θ) **returns** a substitution
  inputs: var, a variable
    x, any expression
    θ, the substitution built up so far

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

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 contd.

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

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono … has some missiles, i.e., $\exists x\ Owns(Nono,x) \wedge Missile(x)$:

$Owns(Nono,M_1)$ and $Missile(M_1)$

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

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile“:

$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American …

$American(West)$

The country Nono, an enemy of America …

$Enemy(Nono,America)$

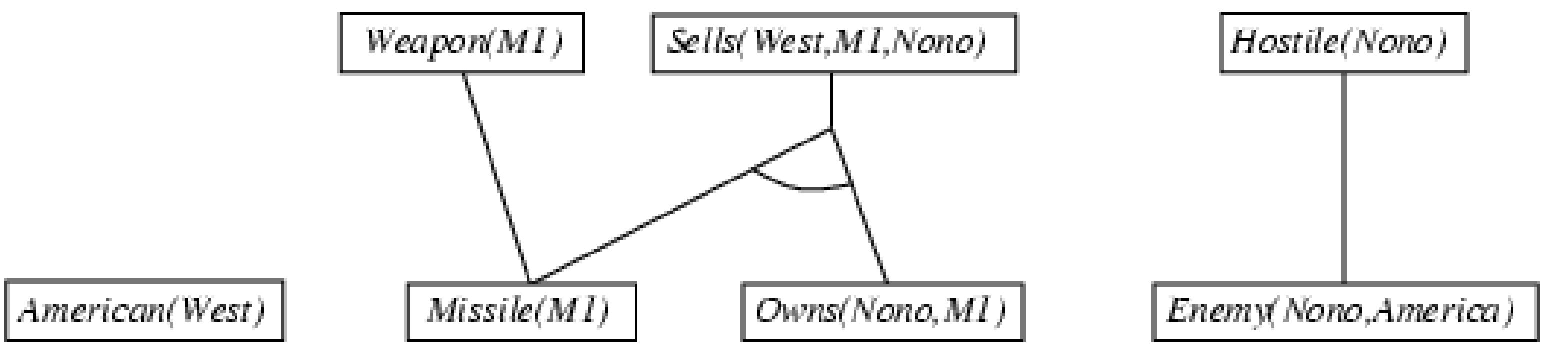
Forward chaining algorithm

```
function FOL-FC-ASK(KB, α) returns a substitution or false
   repeat until new is empty
      new ← { }
      for each sentence r in KB do
         ( p_1 ∧ ... ∧ p_n ⇒ q) ← STANDARDIZE-APART(r)
         for each θ such that (p_1 ∧ ... ∧ p_n)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
```

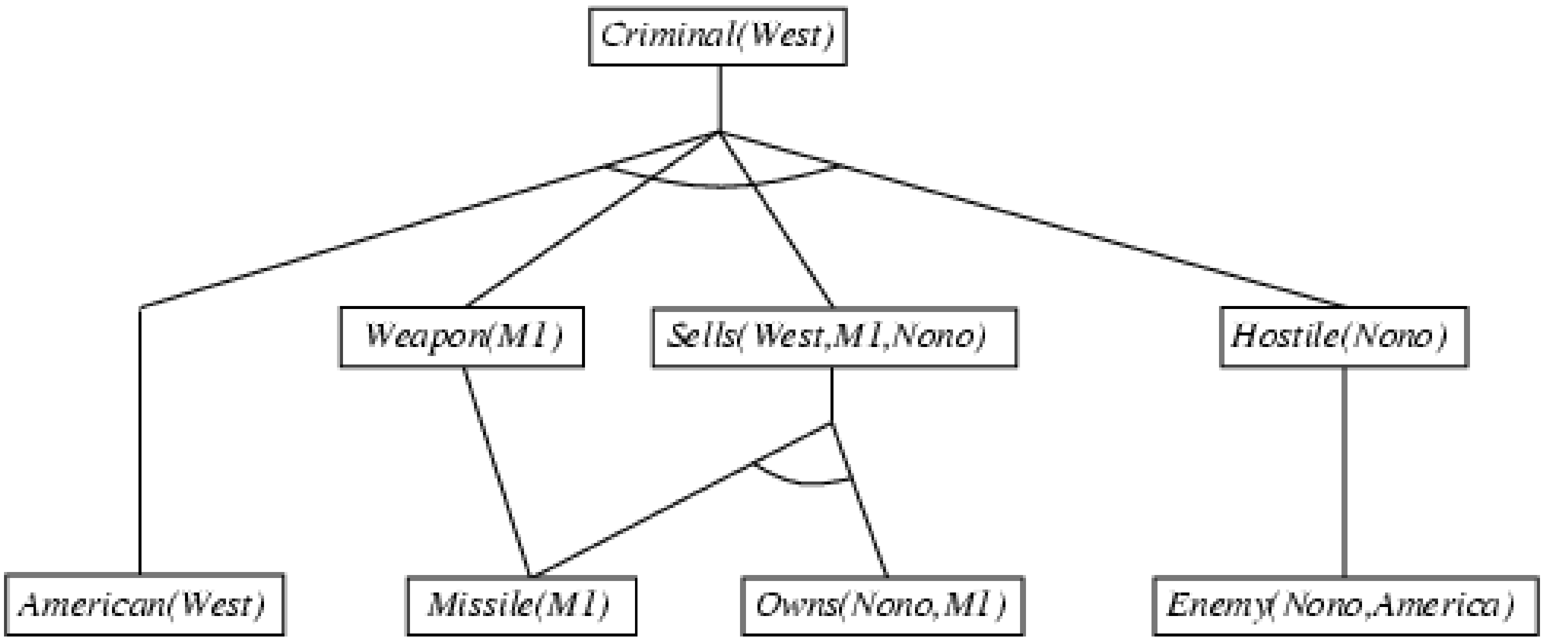
Forward chaining proof

American(West) *Missile(M1)* *Owns(Nono,M1)* *Enemy(Nono,America)*

Forward chaining proof



Forward chaining proof



Properties of forward chaining

- 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*

Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

- $\Rightarrow$ match each rule whose premise contains a newly added positive literal

Matching itself can be expensive:

Database indexing allows O(1) retrieval of known facts

- – e.g., query *Missile(x)* retrieves *Missile*(M_1)

Forward chaining is widely used in deductive databases

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

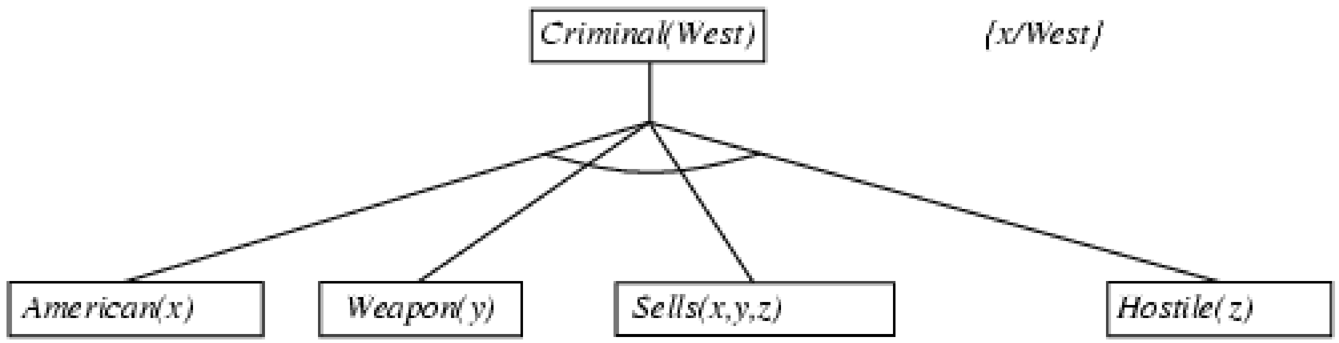
  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$
  for each r **in** KB where $\text{STANDARDIZE-APART}(r) = (\, p_1 \;\wedge\; \ldots \;\wedge\; p_n \;\Rightarrow\; q)$
    and $\theta' \leftarrow \text{UNIFY}(q, q')$ succeeds
   $ans \leftarrow \text{FOL-BC-ASK}(KB, [p_1, \ldots, p_n | \text{REST}(goals)], \text{COMPOSE}(\theta, \theta')) \cup ans$
  return ans

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

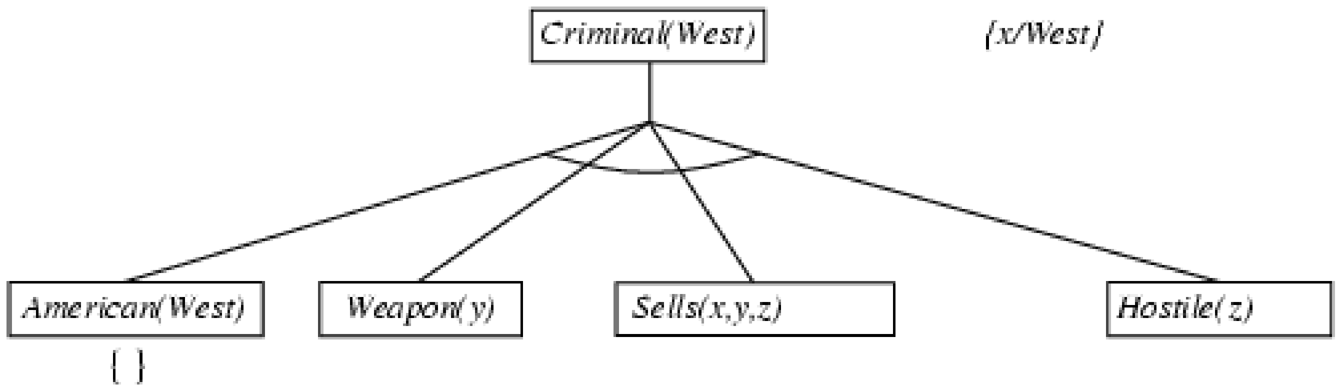
Backward chaining example

$\boxed{Criminal(West)}$

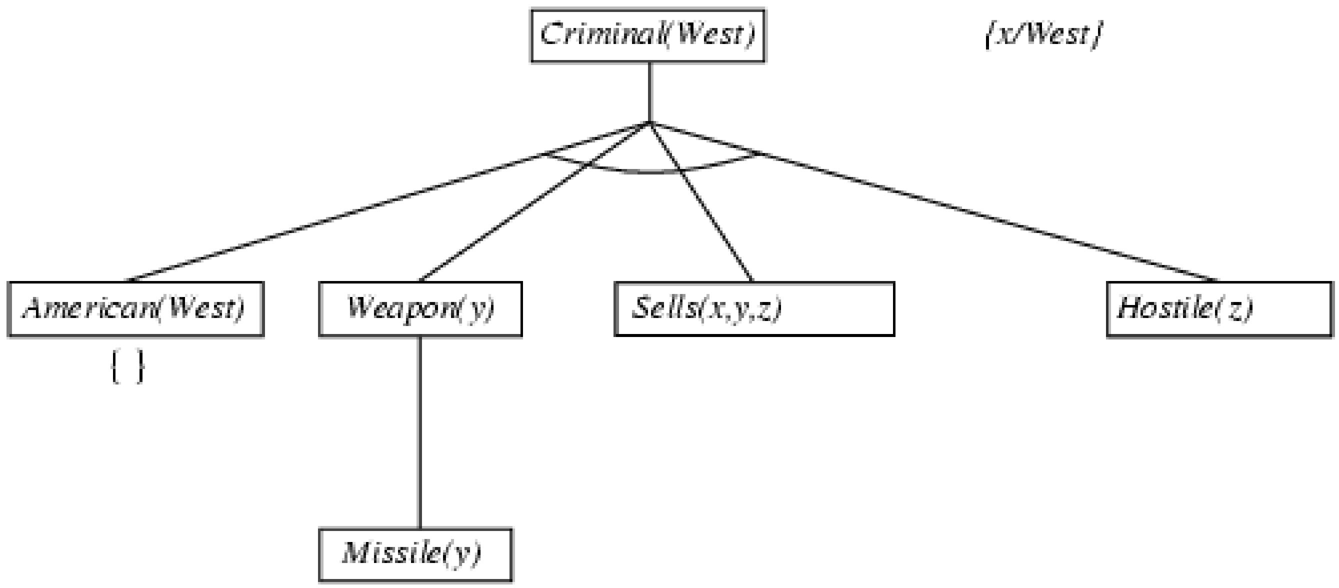
Backward chaining example



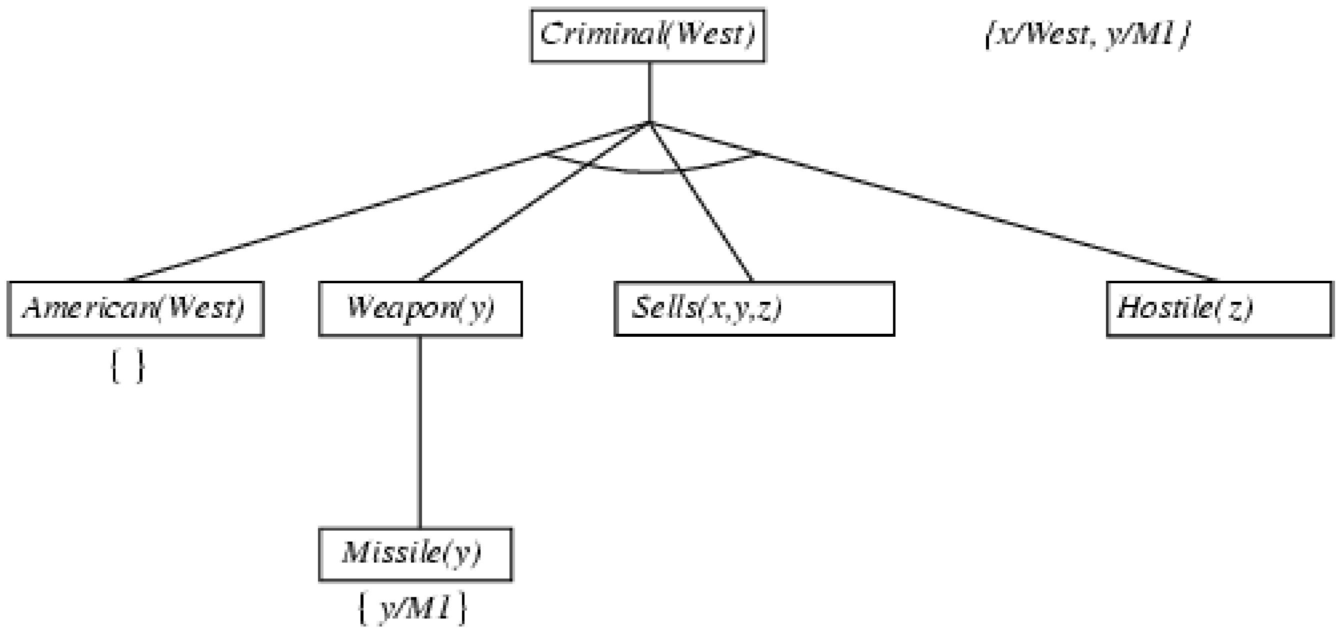
Backward chaining example



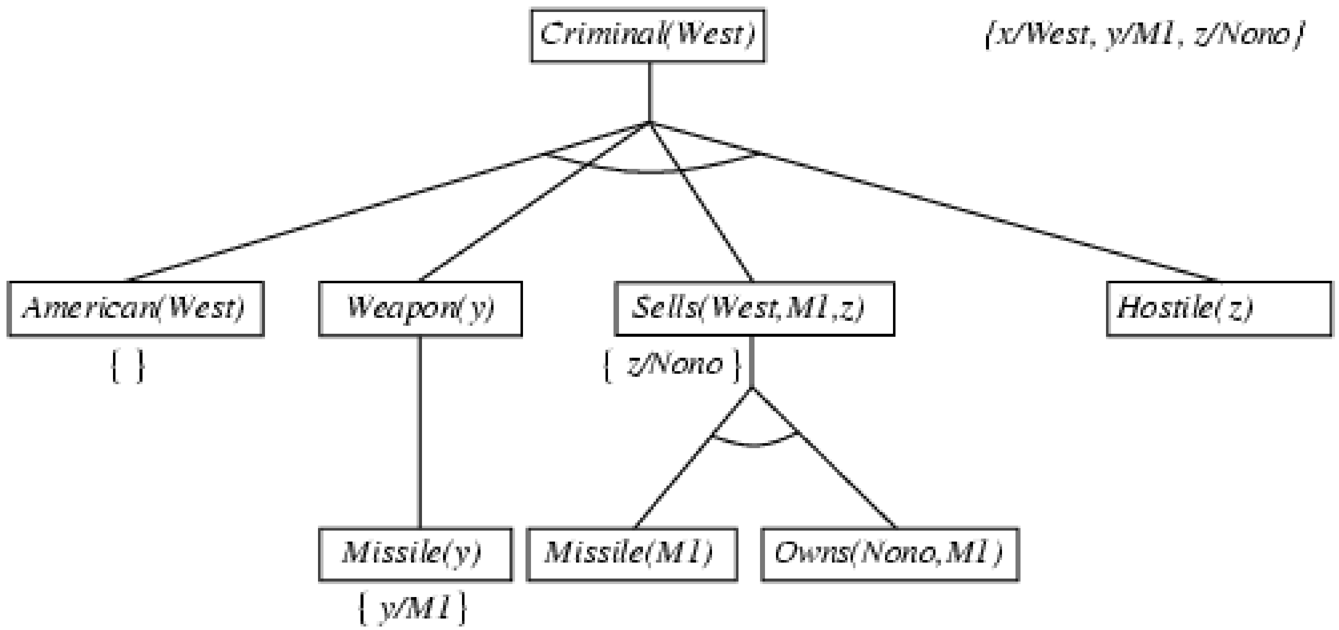
Backward chaining example



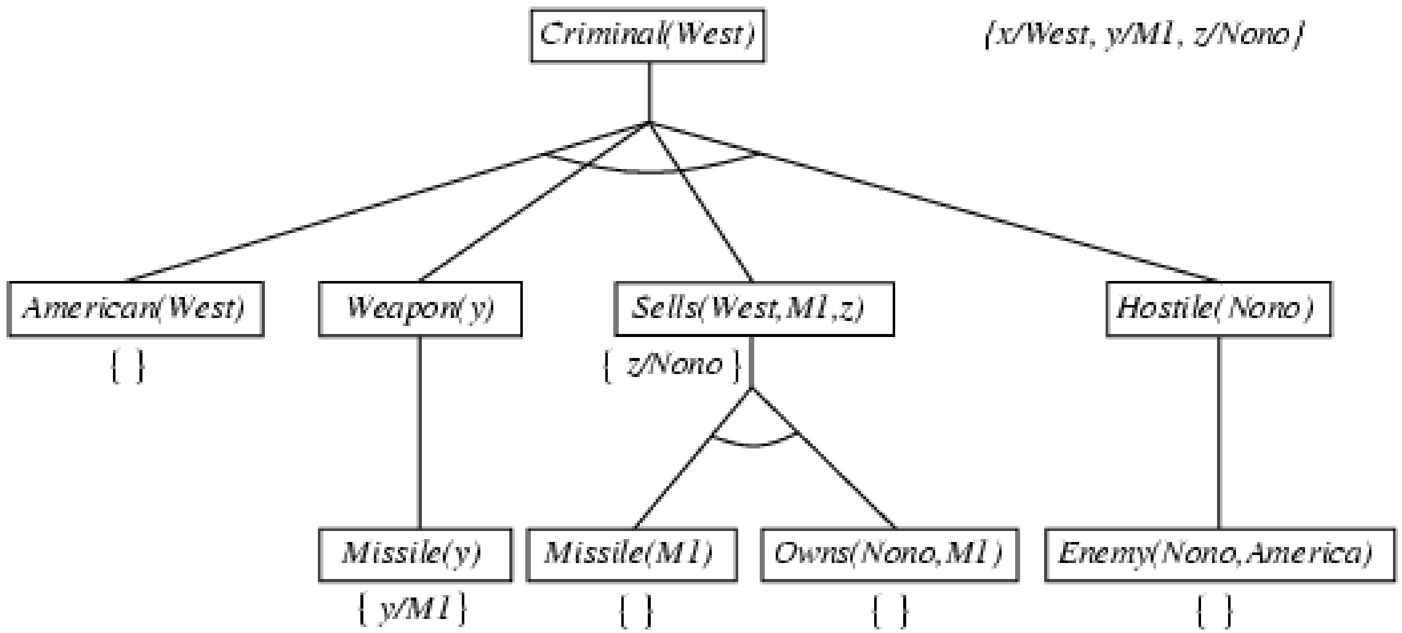
Backward chaining example



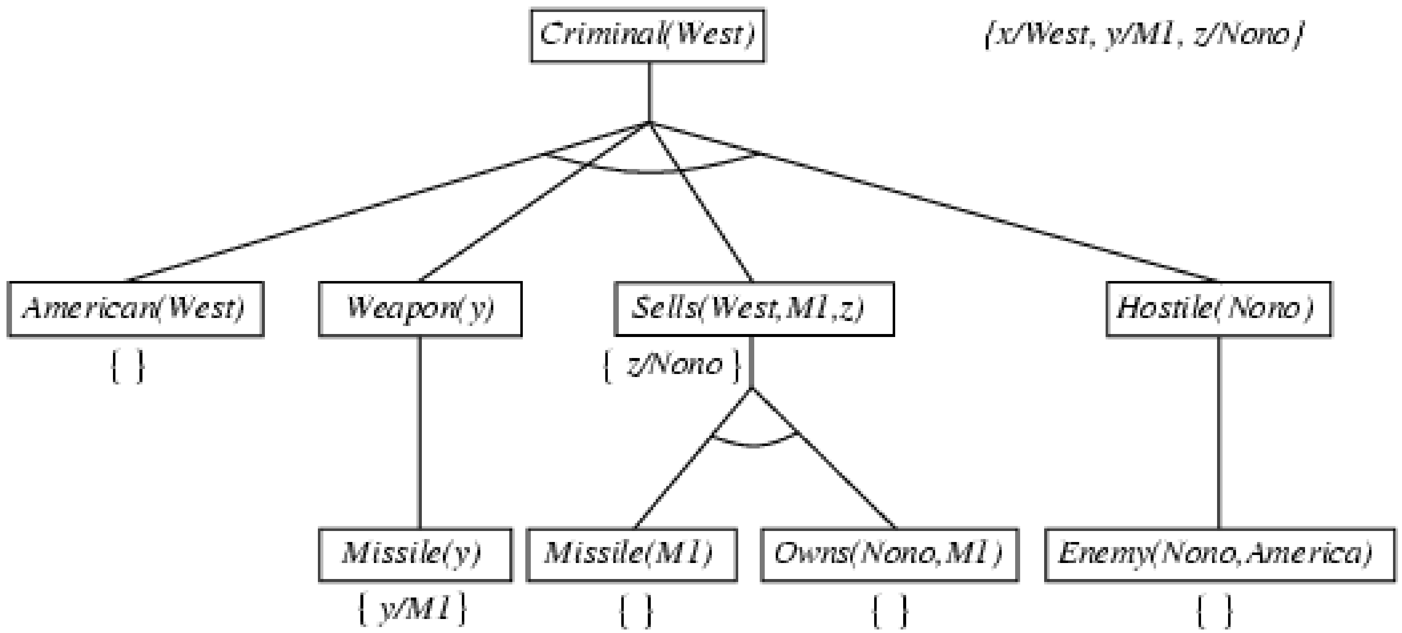
Backward chaining example



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

Resolution in FOL

- FC and BC are not complete for FOL
- Resolution is. (refutation-complete) But slower.
- To prove **KB** ╞ ***α***, show that **KB ^ ¬*α*** is unsatisfiable
  - **KB** and **¬ *α*** need to be in CNF: conjunction of clauses that are disjunction of literals. Any FOL KB can be converted into CNF
  - Repeatedly combines two clauses to make a new one until an empty clause is derived: a contradiction

Resolution: brief summary

- Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

  where `Unify`(ℓ_i, $\neg m_j$) = θ.

- 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 θ = {x/Ken}

- Apply resolution steps to CNF(KB $\wedge \neg\alpha$); complete for FOL

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

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

4. Skolemize: a more general form of existential instantiation.

   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

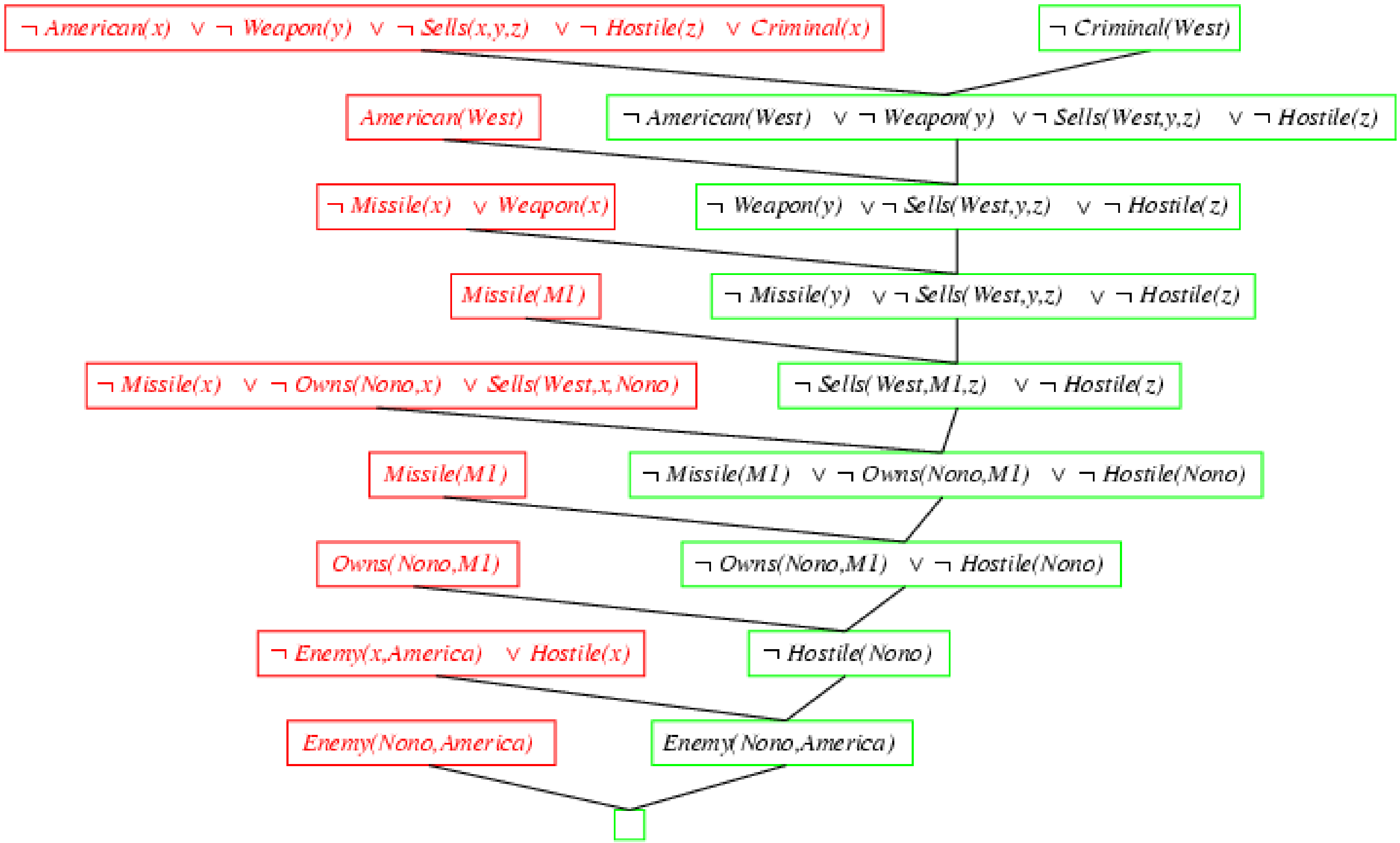
5. Drop universal quantifiers:

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

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

   $[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$□

Resolution proof: definite clauses



Logic programming: Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques ⇒ 60 million LIPS
- 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
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` succeeds if `dead(joe)` fails

Summary

- FOL is a very expressive language, but difficult to perform inference with.
  - One inference method is removing all variables / quantifiers (**propositionalizing**), which is slow.
  - We can also use **unification** to identify appropriate substitutions with **generalized Modus Ponens**, which is complete for HNF but not general FOL.
  - The **forward chaining** and **backward chaining** algorithms use GMP to KBs in HNF.
  - Generalized **resolution** inference is complete for all of FOL. KB must be in CNF. Use refutation.