Inference in first-order logic

Chapter 9

(Please turn your mobile devices to silent. Thanks!)

Last Time

- First Order Logic
  - Reasons about objects, predicates
  - Introduces equality and quantifiers
- Brief excursion into Prolog
  - To be finished and related to more in depth today

Outline

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

Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

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

  for any variable *v* and ground term *g*

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

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

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

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

  …

Existential instantiation (EI)

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

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

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

$$Crown(C_1) \land OnHead(C_1,John)$$

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

Reduction to propositional inference

Suppose the KB contains just the following:

$\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
King(John)
Greedy(John)
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)
  Greedy(John)
  Brother(Richard,John)
- The new KB is propositionalized: proposition symbols are

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

Blank spaces to fill in on this slide

Propositionalization

- Every FOL KB can be propositionalized so as to preserve entailment
  - Convert it to propositional logic
  - A ground sentence is entailed by new KB iff entailed by original KB
- Idea: propositionalize KB and query, apply resolution, return result
- But there's a problem: with , there are infinitely many ground terms:

Propositionalization, continued

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

Remind you of any other algorithm?

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 non-entailed 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
- With *p* *k*-ary predicates and *n* constants, there are $p \cdot n^k$ instantiations.
  - E.g., p=1 k=2 n=3, Rel(_,_). ➔ $3 \times 3 = 3^2 = 9$ possibilities

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)	

- 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 }

The unification algorithm

function UNIFY(x, y, θ) **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 VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  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

Blank spaces to fill in on this slide

Let's do one together

Knows(John,x)   Knows(y,Mother(y))

function UNIFY(x, y, θ) **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 VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure

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

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i\,\theta \text{ for all } i$$

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*)

- GMP used with KB of definite clauses (exactly one positive literal)
  - n.b. recall Horn form allows at most one positive literal (less restrictive)
- All variables assumed universally quantified

Let’s do an example with a KB

- 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 Colonel West is a criminal

The example KB in FOL

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

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

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile":

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

The country Nono, an enemy of America …

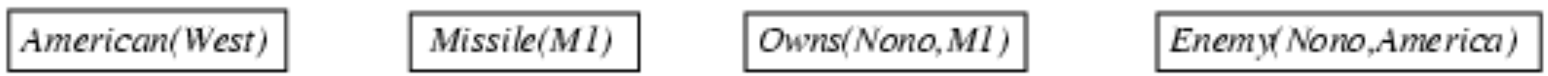
Enemy(Nono,America)

Forward chaining algorithm

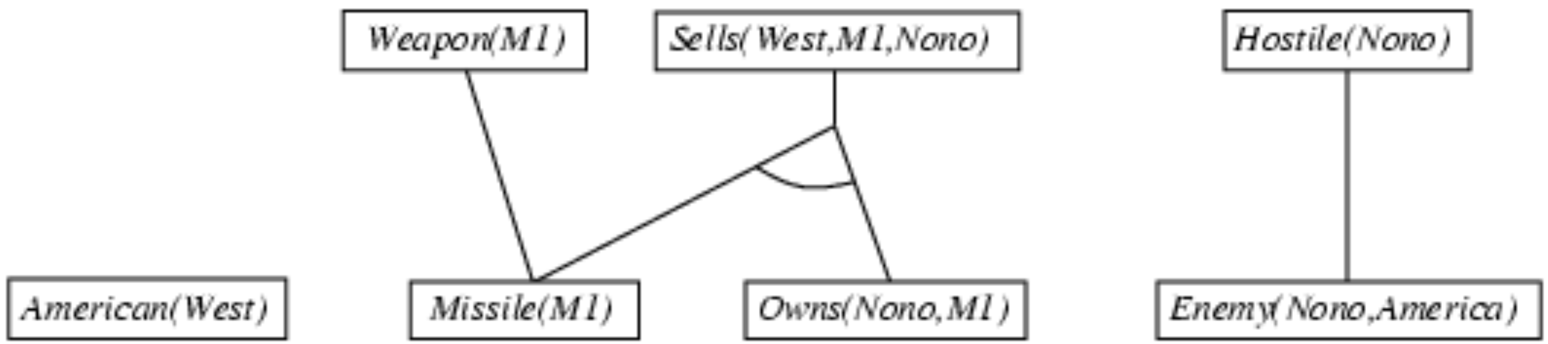
function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
  $new \leftarrow \{\}$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

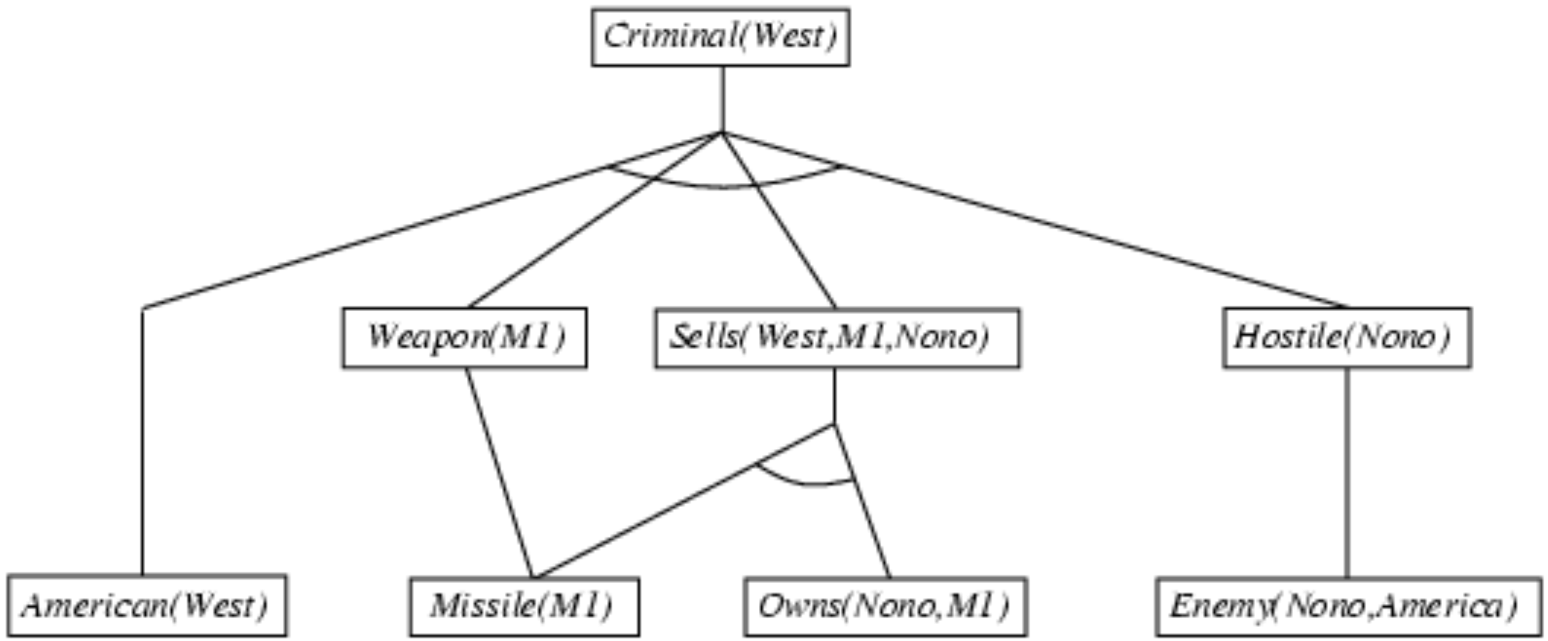
Forward chaining proof



Forward chaining proof



Forward chaining proof



Soundness of GMP

- Need to show that

$$p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \models q\theta$$

provided that $p_i'\theta = p_i\theta$ for all i

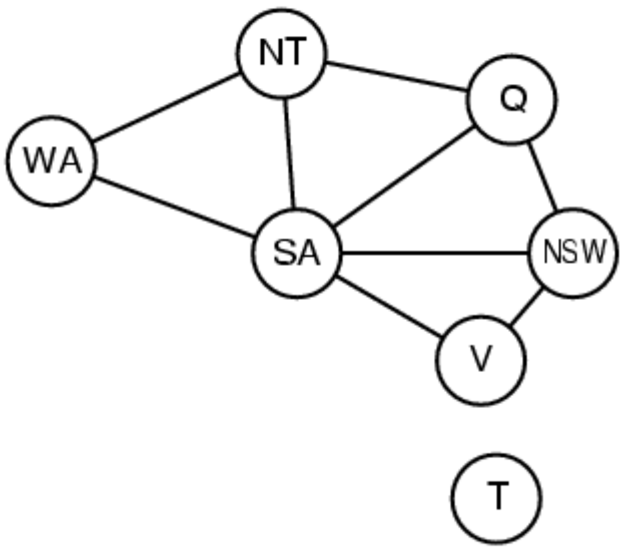
- Lemma: For any sentence *p*, we have *p* $\models p\theta$ by UI

1. $(p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta)$
2. $p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1'\theta \land \ldots \land p_n'\theta$
3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

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 again semidecidable

Equivalence to CSPs



Definite clause

Diff(wa,nt) ∧ *Diff(wa,sa)* ∧ *Diff(nt,q)* ∧ *Diff(nt,sa)* ∧ *Diff(q,nsw)* ∧ *Diff(q,sa)* ∧ *Diff(nsw,v)* ∧ *Diff(nsw,sa)* ∧ *Diff(v,sa)* ⇒ *Colorable()*

Diff(Red,Blue) *Diff (Red,Green)*
Diff(Green,Red) *Diff(Green,Blue)*
Diff(Blue,Red) *Diff(Blue,Green)*

Some facts

- Each conjunct can be viewed as a constraint on a variable.
- Every finite CSP can be expressed as a single definite clause together with some facts.

Improving efficiency

The algorithm presented earlier isn't efficient. Let's make it better.

1. Matching itself is expensive, Database indexing allows O(1) retrieval of known facts
   - e.g., query a table where all instantations of *p(x) are stored; Missile(x)* retrieves $Missile(M_1)$

   For predicates with many subgoals, the conjunct ordering problem applies
   - e.g., for $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West, x, Nono)$| if there are many things owned by Nono, perhaps better to start with *Missile(x)* conjunct

Improving efficiency, continued

2. Incremental forward chaining: Only match rules on iteration *k* if a premise was added on iteration *k-1*
   - ⇒ Original algorithm discards partially matched rules
   - ⇒ Instead, keep track of conjuncts matched to avoid duplicate work

   - ⇒ Match each rule whose premise contains a newly added positive literal

Leads to the development of Rete (*"Ree-Tee"*) networks in real world production systems

3. Irrelevant Facts: several ways to address … let's segue to Backward Chaining.

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
```

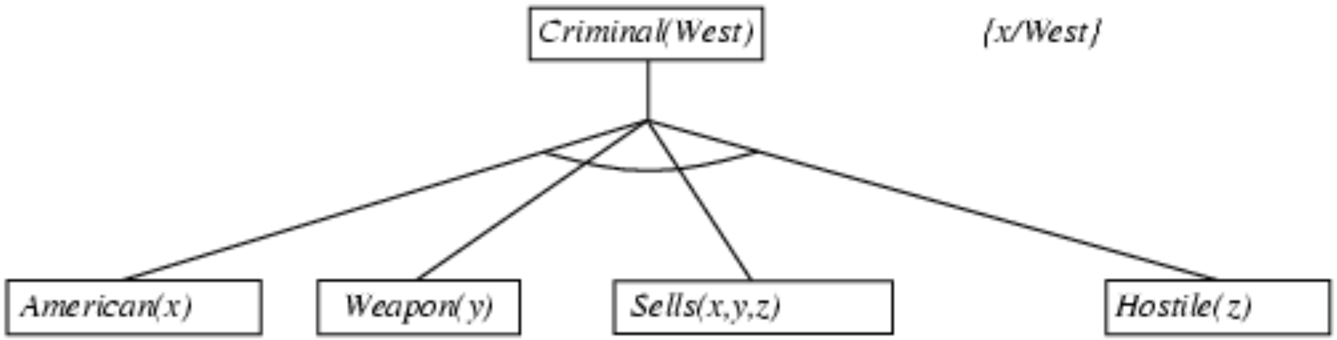
What type of search algorithm is this?

$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$

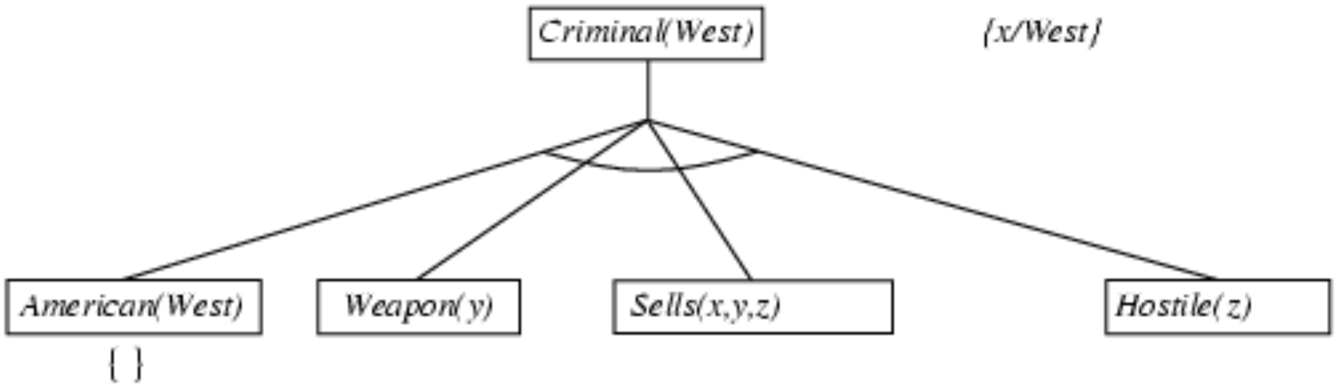
Backward chaining example

$Criminal(West)$

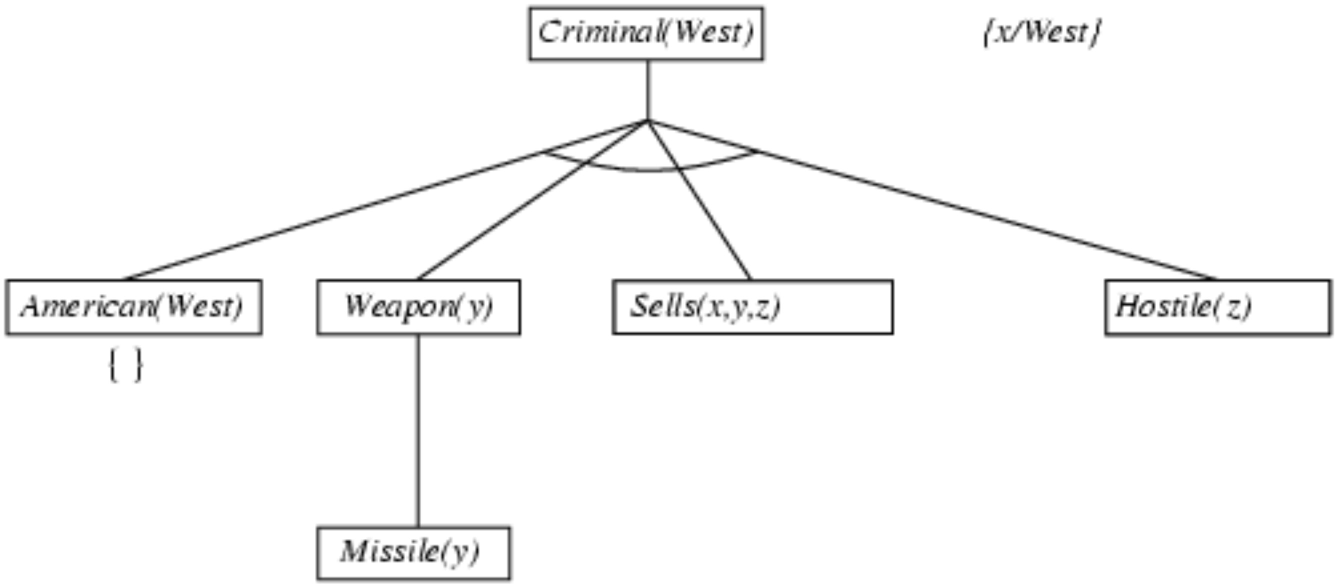
Backward chaining example



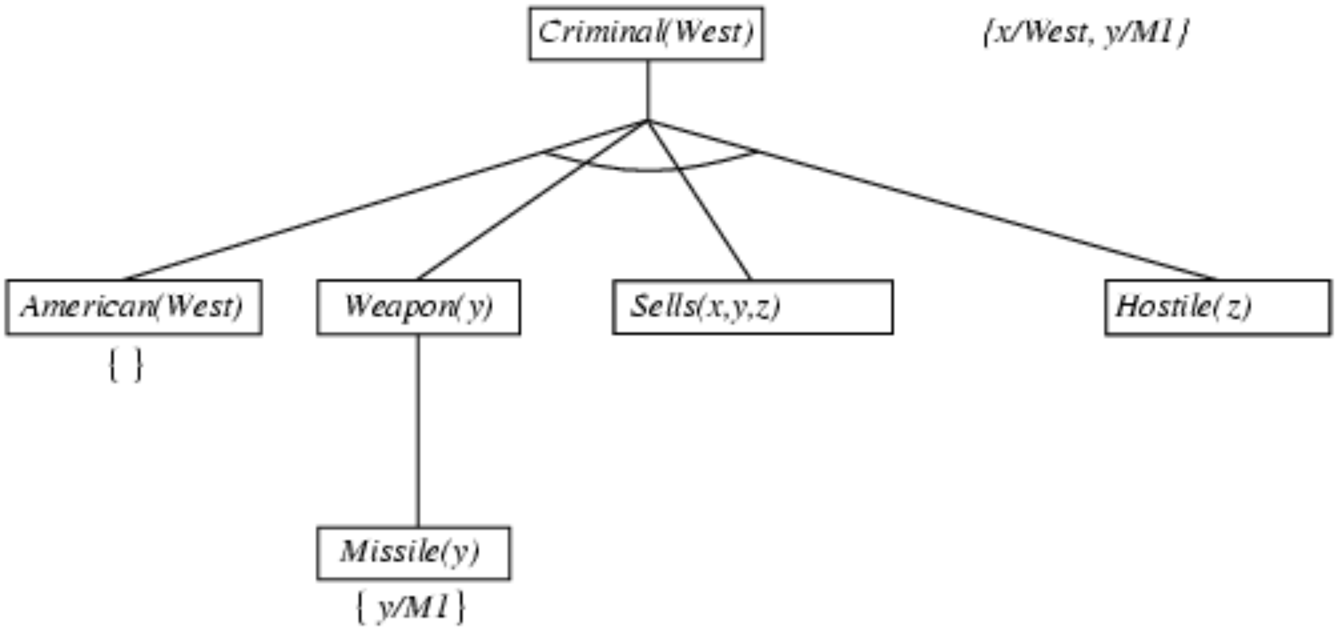
Backward chaining example



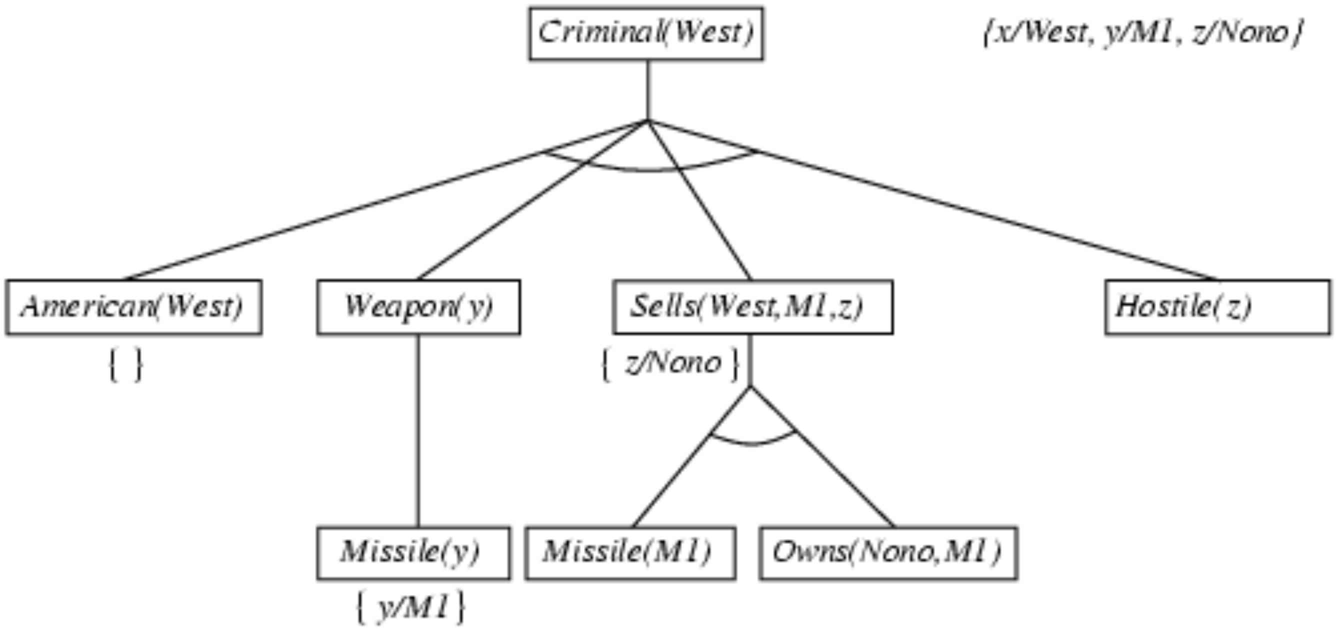
Backward chaining example



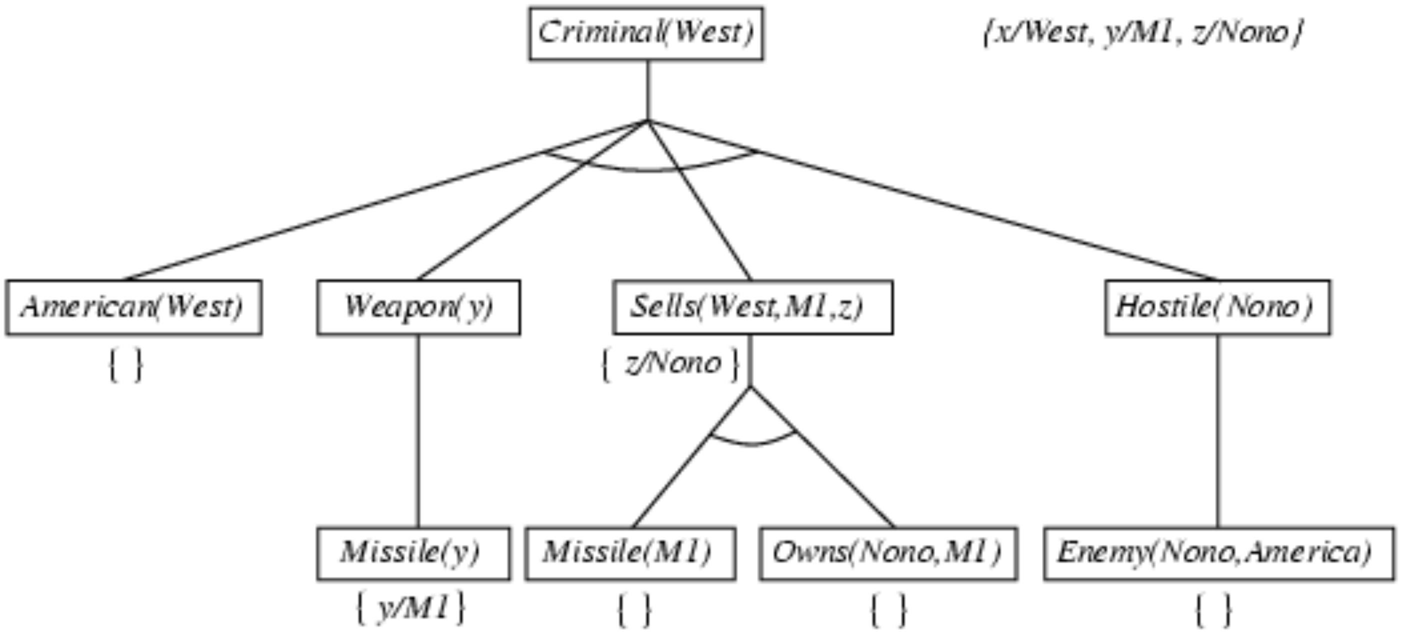
Backward chaining example



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
  - ⇒ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - ⇒ fix using caching of previous results (extra space)
- Widely used for logic programming

Logic programming: Prolog

- Backward chaining with Horn clauses + bells & whistles

- 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 can have side effects (e.g., input and output predicates, assert/retract predicates)
- Closed-world assumption / database semantics ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` succeeds if `dead(joe)` fails

- No checks for infinite recursion
- No occurs check for unification

Resolution: recap and look at FOL

- 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`$(\ell_i, \neg m_j) = \theta$.

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,

$$\frac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Ken)}{Unhappy(Ken)}$$

  with θ = {x/Ken}

- Apply resolution steps to CNF(KB ∧ ¬α); complete for FOL*

Conversion to CNF

- Everyone who loves all animals is loved by someone:

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

  Is this the same y?

- 1. Eliminate biconditionals and implications

  $\forall x\ [\neg \forall y\ \neg Animal(y) \lor Loves(x,y)] \lor [\exists y\ Loves(y,x)]$

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

  $\forall x\ [\exists y\ \neg(\neg Animal(y) \lor Loves(x,y))] \lor [\exists y\ Loves(y,x)]$

  $\forall x\ [\exists y\ \neg\neg Animal(y) \land \neg Loves(x,y)] \lor [\exists y\ Loves(y,x)]$

  $\forall x\ [\exists y\ Animal(y) \land \neg Loves(x,y)] \lor [\exists y\ Loves(y,x)]$

Conversion to CNF, continued

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:

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

   Why do we need a function and not a variable?

5. Drop universal quantifiers:

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

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

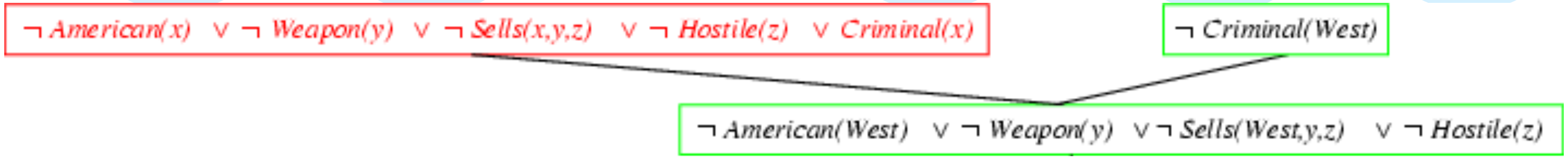
   $[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$

Resolution proof: definite clauses

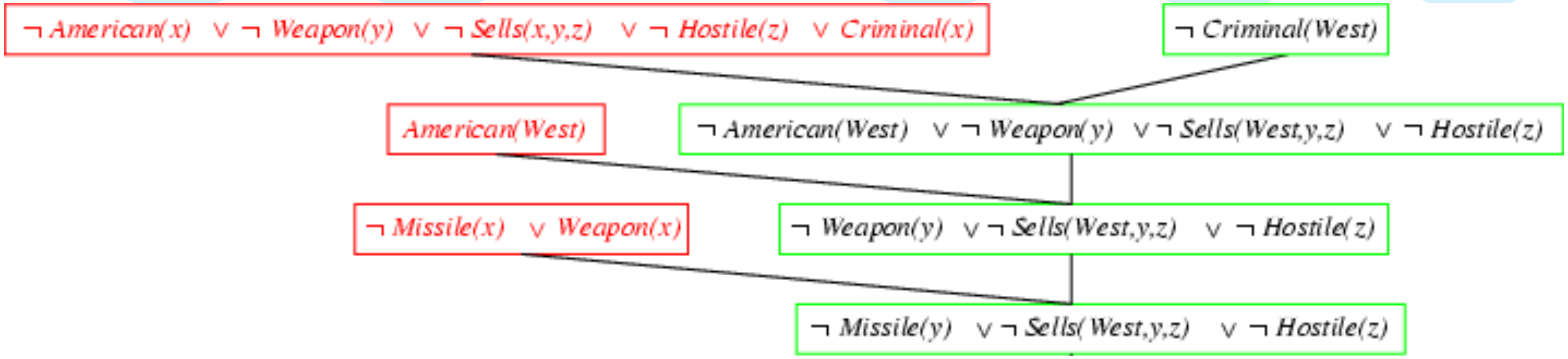
$\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor Criminal(x)$

$\neg Criminal(West)$

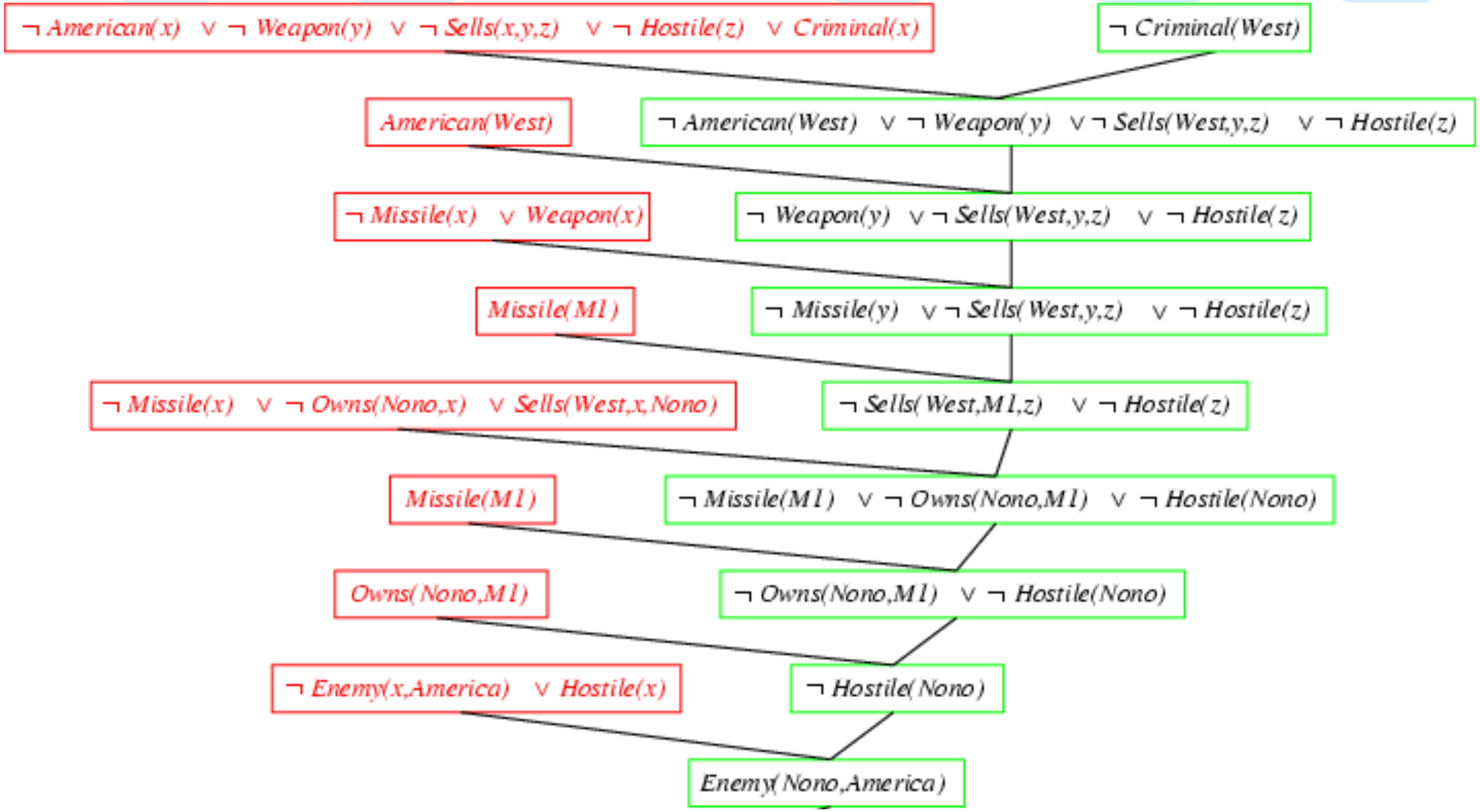
Resolution proof: definite clauses



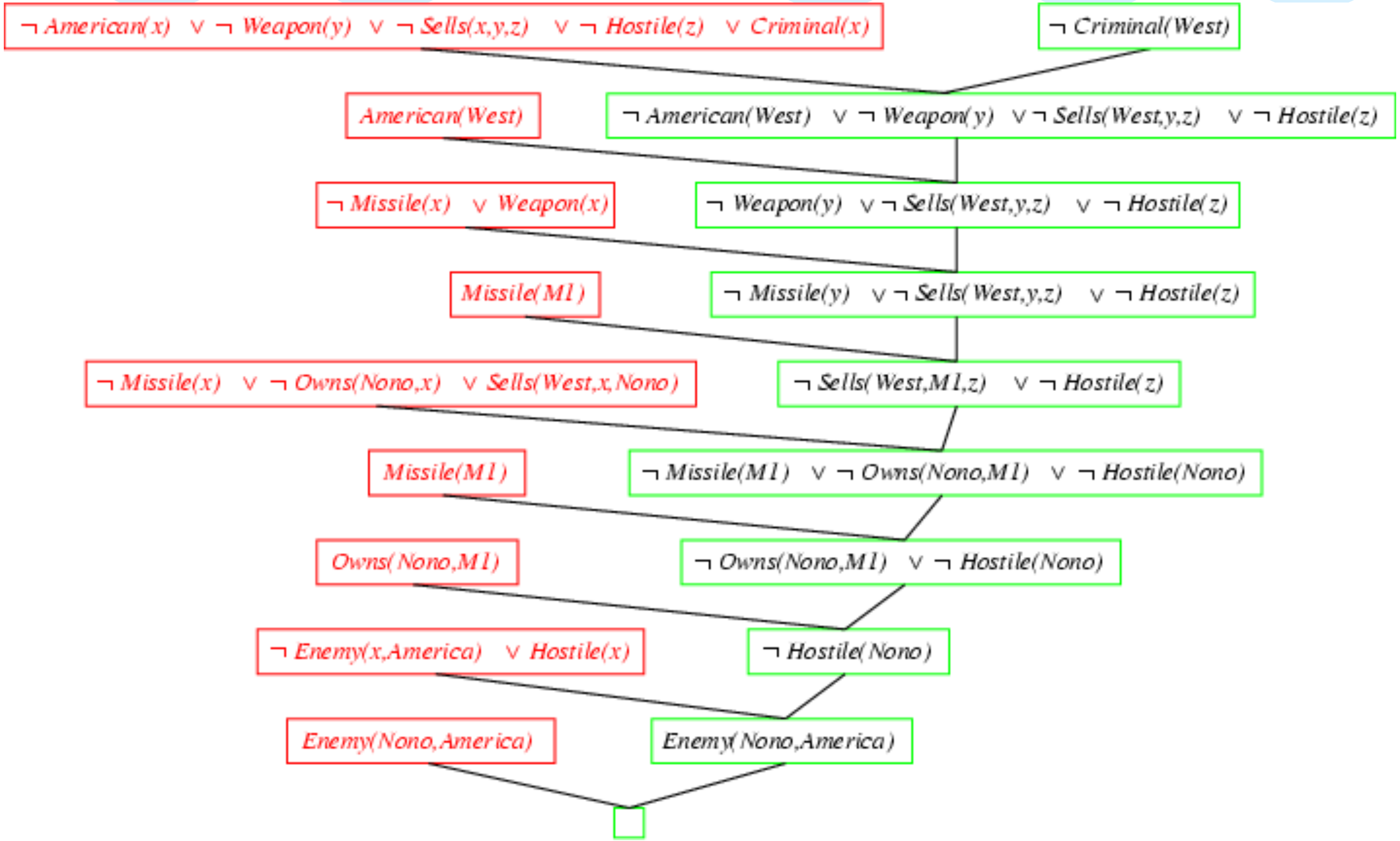
Resolution proof: definite clauses



Resolution proof: definite clauses



Resolution proof: definite clauses



Refutation completeness

- Resolution can say yes to any entailed sentence but cannot be used to *generate* all entailed sentences
  - E.g., won't generate Animal(x) $\vee$ $\neg$Animal(x)

Blank spaces to fill in on this slide

Resolution special cases

- Factoring: may need to remove redundant literals (literals that are unifiable)
- L(x) ∨ G(a,b)
- ¬L(x) ∨ G(K,L)
-

- To handle equality x=y, need to use demodulation (sub x for y in some clause that has x).
- B = Son(A)
- Property(B)
-

Summary

- Examined our three strategies for logic inference in FOL:
  - Forward Chaining
  - Backward Chaining (what Prolog uses)
  - Resolution

- To think about: when is each of the three systems the most appropriate?