

# CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, [macskass@usc.edu](mailto:macskass@usc.edu)

TAs: Nadeesha Ranashinghe ([nadeeshr@usc.edu](mailto:nadeeshr@usc.edu))

William Yeoh ([wyeoh@usc.edu](mailto:wyeoh@usc.edu))

Harris Chiu ([chiciu@usc.edu](mailto:chiciu@usc.edu))

Lectures: MW 5:00-6:20pm, OHE 122 / DEN

Office hours: By appointment

Class page: <http://www-rcf.usc.edu/~macskass/CS561-Spring2010/>

This class will use <http://www.uscden.net/> and class webpage

- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:

[AIMA] Artificial Intelligence: A Modern Approach,  
by Stuart Russell and Peter Norvig. (2nd ed)

# Inference in FOL [AIMA Ch. 9]

---

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution

# A brief history of reasoning

450b.c.	<b>Stoics</b>	propositional logic, inference (maybe)
322b.c.	<b>Aristotle</b>	“syllogisms” (inference rules), quantifiers
1565	<b>Cardano</b>	probability theory (propositional logic+ uncertainty)
1847	<b>Boole</b>	propositional logic (again)
1879	<b>Frege</b>	first-order logic
1922	<b>Wittgenstein</b>	proof by truth tables
1930	<b>Gödel</b>	$\exists$ complete algorithm for FOL
1930	<b>Herbrand</b>	complete algorithm for FOL (reduce to propositional)
1931	<b>Gödel</b>	$\neg\exists$ complete algorithm for arithmetic
1960	<b>Davis/Putnam</b>	“practical” algorithm for propositional logic
1965	<b>Robinson</b>	“practical” algorithm for FOL—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.,  $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$  yields  
 $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$   
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$   
 $\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$   
...

# Existential instantiation (EI)

- For any sentence  $\alpha$ , 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.,  $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$  yields  
 $\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$

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

- Another example: from  $\exists x d(x^y)/dy = x^y$  we obtain  
 $d(e^y)/dy = e^y$

provided  $e$  is a new constant symbol

# Existential instantiation contd.

---

- UI can be applied several times to **add** new sentences; the new KB is logically equivalent to the old
- EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but is satisfiable iff the old KB was satisfiable

# Reduction to propositional inference

Suppose the KB contains just the following:

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

Instantiating the universal sentence in **all possible ways**, we have

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

The new KB is **propositionalized**: proposition symbols are

$\text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard})$  etc.

# Reduction contd.

---

- Claim: a ground sentence is entailed by new KB iff entailed by original KB
- Claim: every FOL KB can be propositionalized so as to preserve entailment
- 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 contd.

- Theorem: Herbrand (1930). If a sentence  $\alpha$  is entailed by an FOL KB, it is entailed by a **finite** subset of the propositional KB
- Idea: For  $n = 0$  to  $\infty$  do
  - create a propositional KB by instantiating with depth- $n$  terms and see if  $\alpha$  is entailed by KB
- Problem: works if  $\alpha$  is entailed, loops if  $\alpha$  is not entailed
- Theorem: Turing (1936), Church (1936), entailment in FOL is **semidecidable**

# Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
- E.g., from
  - $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  - $\text{King}(\text{John})$
  - $\forall y \text{ Greedy}(y)$
  - $\text{Brother}(\text{Richard}, \text{John})$
- it seems obvious that  $\text{Evil}(\text{John})$ , but propositionalization produces lots of facts such as  $\text{Greedy}(\text{Richard})$  that are irrelevant
- With  $p$   $k$ -ary predicates and  $n$  constants, there are  $p \cdot n^k$  instantiations
- With function symbols, it gets much much worse!

# Unification

- We can get the inference immediately if we can find a substitution  $\theta$  such that  $King(x)$  and  $Greedy(x)$  match  $King(John)$  and  $Greedy(y)$
- $\theta = \{x/John, y/John\}$  works
- **UNIFY** $(\alpha, \beta) = \theta$  if  $\alpha\theta = \beta\theta$

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

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

# The unification algorithm

**function** 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

$\theta$ , the substitution built up so far

**if**  $\theta = \text{failure}$  **then return failure**

**else if**  $x = y$  **then return**  $\theta$

**else if** VARIABLE?( $x$ ) **then return** UNIFY-VAR( $x, y, \theta$ )

**else if** VARIABLE?( $y$ ) **then return** UNIFY-VAR( $y, x, \theta$ )

**else if** COMPOUND?( $x$ ) **and** COMPOUND?( $y$ ) **then**

**return** UNIFY(ARGs[ $x$ ], ARGs[ $y$ ], UNIFY(OP[ $x$ ], OP[ $y$ ],  $\theta$ ))

**else if** LIST?( $x$ ) **and** LIST?( $y$ ) **then**

**return** UNIFY(REST[ $x$ ], REST[ $y$ ], UNIFY(FIRST[ $x$ ], FIRST[ $y$ ],  $\theta$ ))

**else return failure**

# The unification algorithm

**function** UNIFY-VAR( $var, x, \theta$ ) returns a substitution

**inputs:**  $var$ , a variable

$x$ , any expression

$\theta$ , the substitution built up so far

**if**  $\{var/val\} \in \theta$  **then return** UNIFY( $val, x, \theta$ )

**else if**  $\{x/val\} \in \theta$  **then return** UNIFY( $var, val, \theta$ )

**else if** OCCUR-CHECK?( $var, x$ ) **then return** failure

**else return** add  $\{var/x\}$  to  $\theta$

# Generalized Modus Ponens (GMP)

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

- where  $p_i'\theta = p_i\theta$  for all  $i$
- $p_1'$  is *King(John)*                       $p_1$  is *King(x)*
- $p_2'$  is *Greedy(y)*                         $p_2$  is *Greedy(x)*
- $\theta$  is  $\{x/John, y/John\}$                    $q$  is *Evil(x)*
- $q\theta$  is *Evil(John)*

GMP used with KB of **definite clauses** (exactly one positive literal)

All variables assumed universally quantified

# Soundness of GMP

Need to show that

$$p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q) \vDash q\theta$$

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

Lemma: For any definite clause  $p$ , we have  $p \vDash p\theta$  by UI

1.  $(p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q) \vDash (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge p_2\theta \wedge \dots \wedge p_n\theta \Rightarrow q\theta)$
2.  $p_1', p_2', \dots, p_n' \vDash p_1' \wedge p_2' \wedge \dots \wedge p_n' \vDash p_1'\theta \wedge p_2'\theta \wedge \dots \wedge p_n'\theta$
3. From 1 and 2,  $q$  follows by ordinary Modus Ponens

# 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

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

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 Non, an enemy of America

$Enemy(Nono,America)$

# Forward chaining algorithm

```
function FOL-FC-ASK( $KB, \alpha$ ) returns a substitution or false
  repeat until new is empty
     $new \leftarrow \{ \}$ 
    for each sentence  $\tau$  in  $KB$  do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(\tau)$ 
      for each  $\theta$  such that  $(p_1 \wedge \dots \wedge p_n)\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$ 
        for some  $p'_1, \dots, p'_n$  in  $KB$ 
           $q' \leftarrow \text{SUBST}(\theta, q)$ 
          if  $q'$  is not a renaming of a sentence already in  $KB$  or new then do
            add  $q'$  to new
             $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
            if  $\phi$  is not fail then return  $\phi$ 
  add new to  $KB$ 
  return false
```

# Forward chaining proof

---

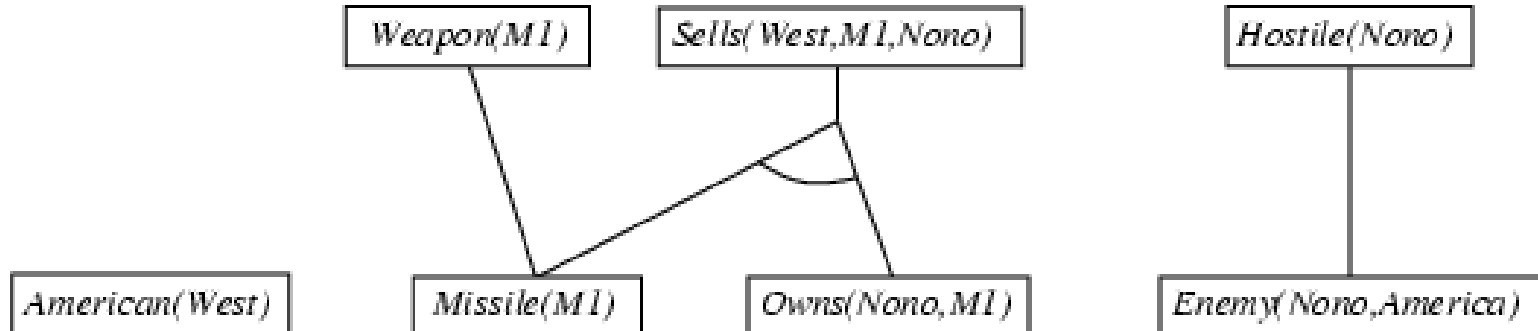
*American(West)*

*Missile(MI)*

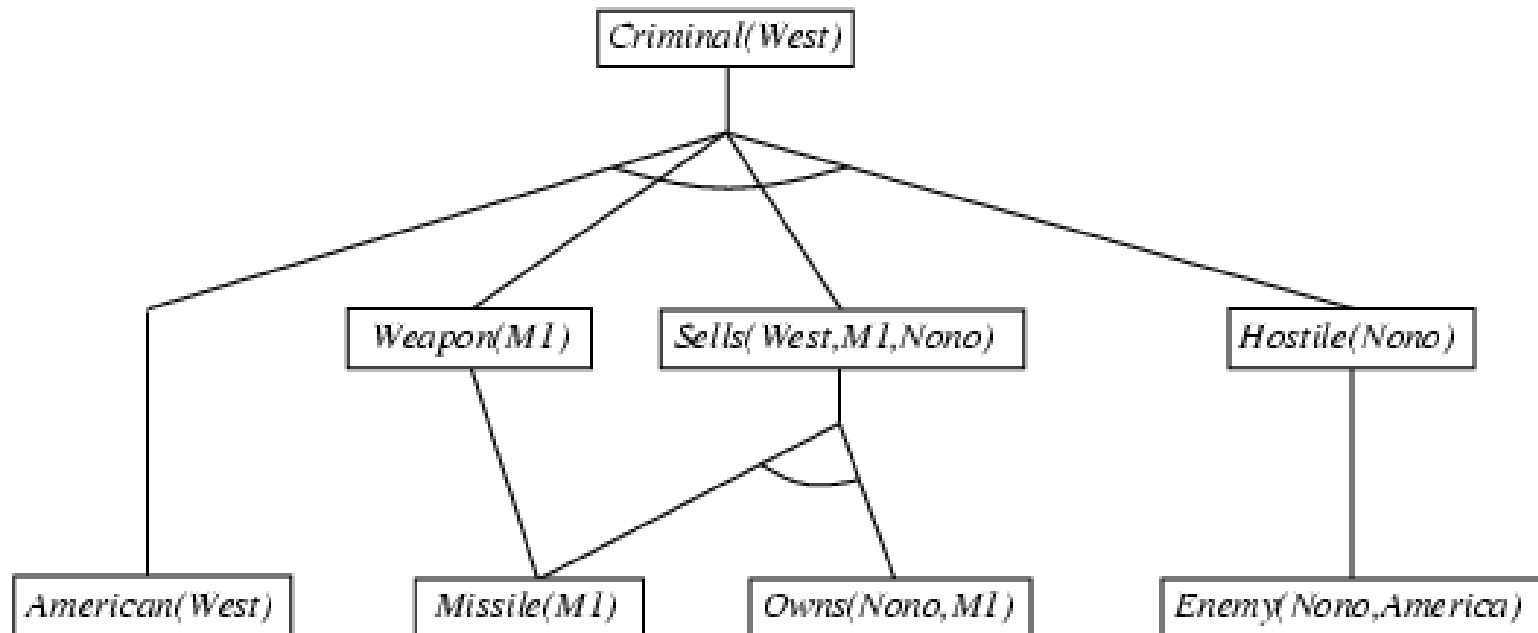
*Owns(Nono, MI)*

*Enemy(Nono, America)*

# Forward chaining proof



# Forward chaining proof



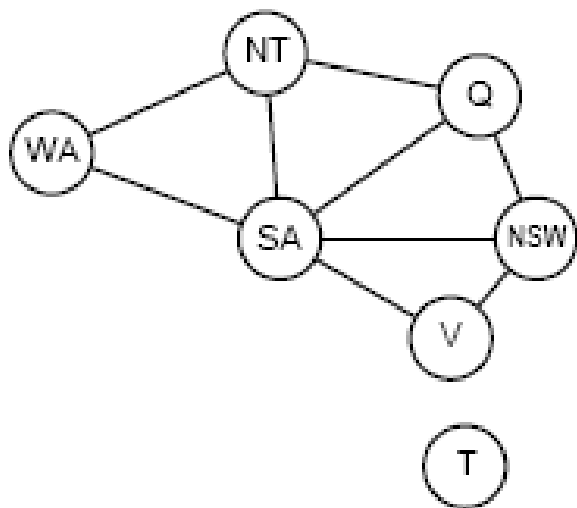
# Properties of forward chaining

- Sound and complete for first-order definite clauses (proof similar to propositional proof)
- **Datalog** = first-order definite clauses + **no functions** (e.g., crime KB)
- FC terminates for Datalog in poly iterations: at most  $p \cdot n^k$  literals
- May not terminate in general if  $\alpha$  is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable

# Efficiency of forward chaining

- Simple observation: no need to match a rule on iteration  $k$  if a premise wasn't added on iteration  $k-1$ 
  - ⇒ match each rule whose premise contains a newly added literal
- Matching itself can be expensive
- Database indexing allows  $O(1)$  retrieval of known facts  
e.g., query  $Missile(x)$  retrieves  $Missile(M_1)$
- Matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases

# Hard matching example



$$\begin{aligned} & Diff(wa, nt) \wedge Diff(wa, sa) \wedge \\ & Diff(nt, q) Diff(nt, sa) \wedge \\ & Diff(q, nsw) \wedge Diff(q, sa) \wedge \\ & Diff(nsw, v) \wedge Diff(nsw, sa) \wedge \\ & Diff(v, sa) \Rightarrow Colorable() \\ & Diff(Blue, Red) \quad Diff(Blue, Green) \\ & Diff(Green, Red) \quad Diff(Green, Blue) \\ & Diff(Red, Blue) \quad Diff(Red, Green) \end{aligned}$$

- **Colorable()** is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

# Backward chaining algorithm

```
function FOL-BC-ASK( $KB, goals, \theta$ ) returns a set of substitutions
  inputs:  $KB$ , a knowledge base
            $goals$ , a list of conjuncts forming a query ( $\theta$  already applied)
            $\theta$ , the current substitution, initially the empty substitution  $\{\}$ 
  local variables:  $answers$ , a set of substitutions, initially empty

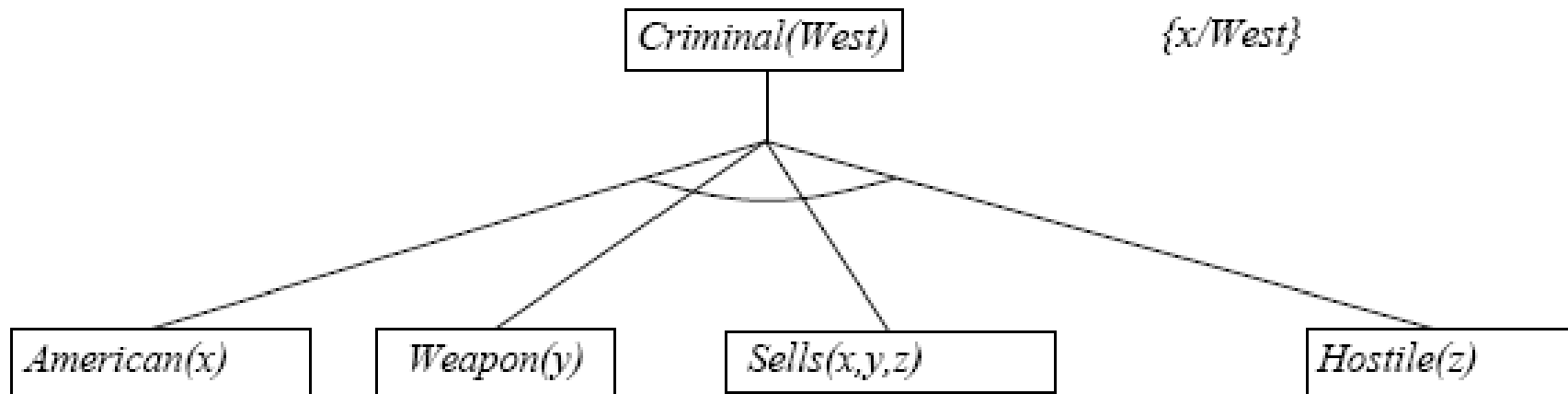
  if  $goals$  is empty then return  $\{\theta\}$ 
   $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$ 
  for each sentence  $r$  in  $KB$ 
    where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
     $new\_goals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

# Backward chaining example

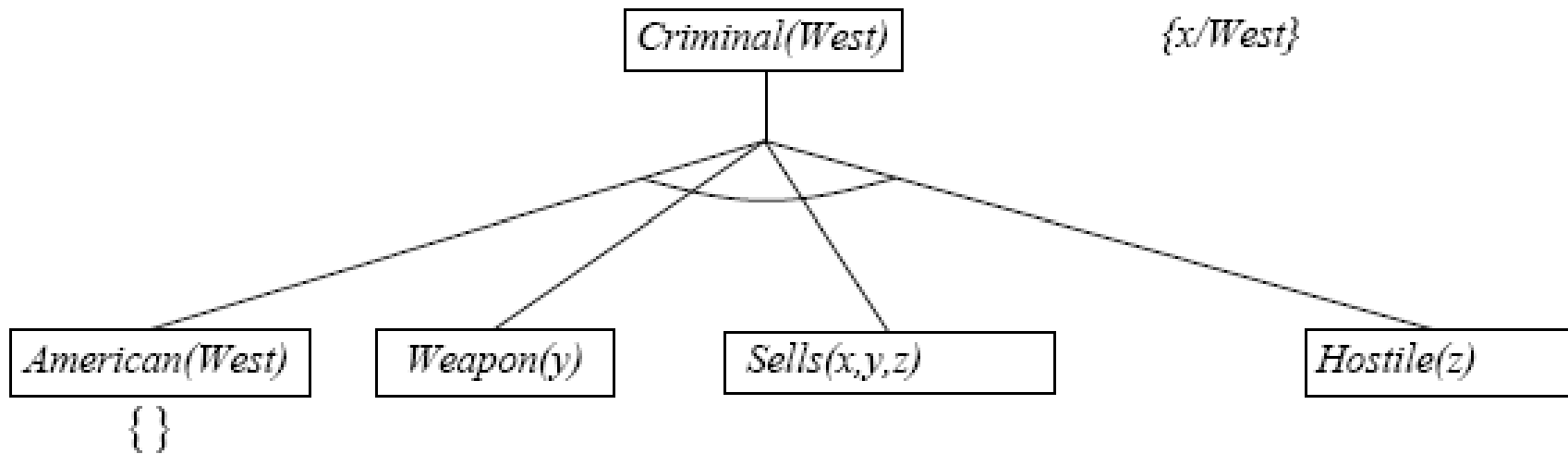
---

*Criminal(West)*

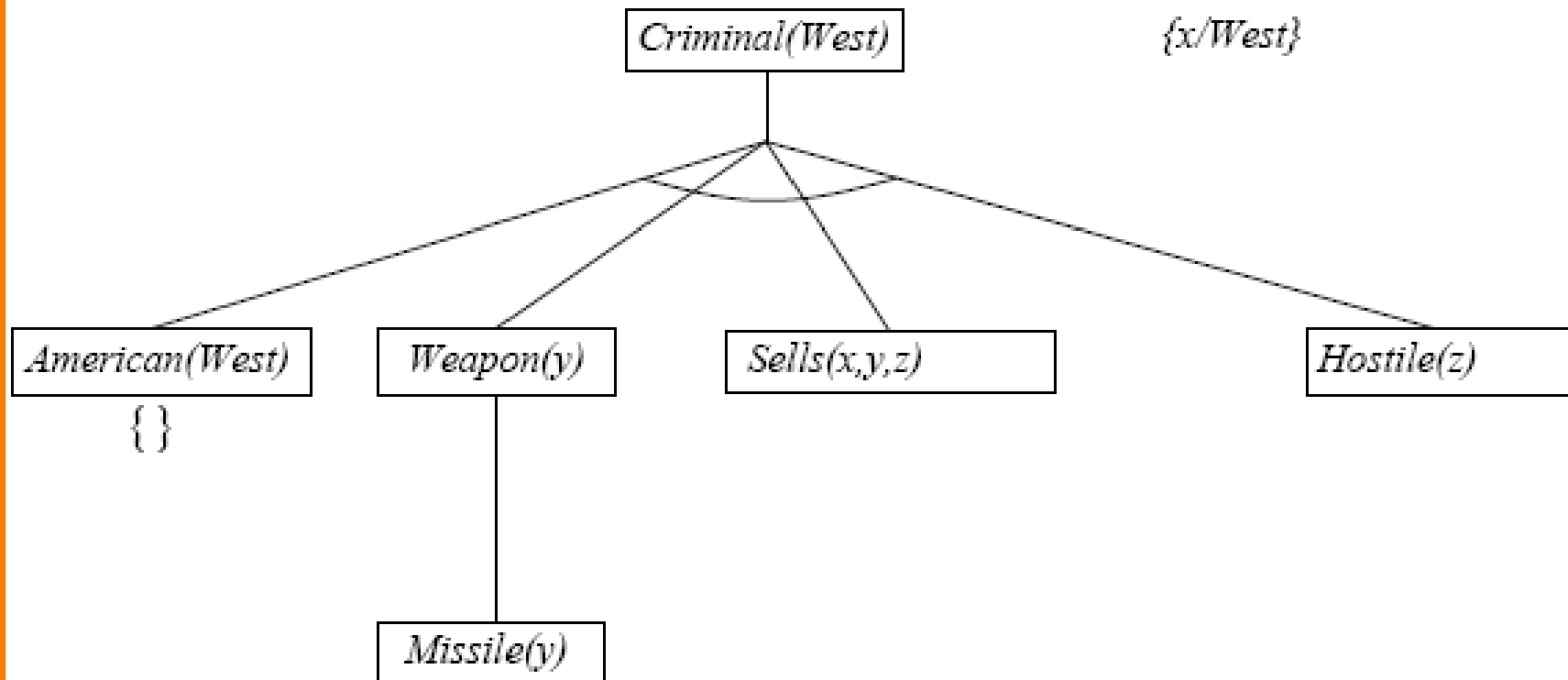
# Backward chaining example



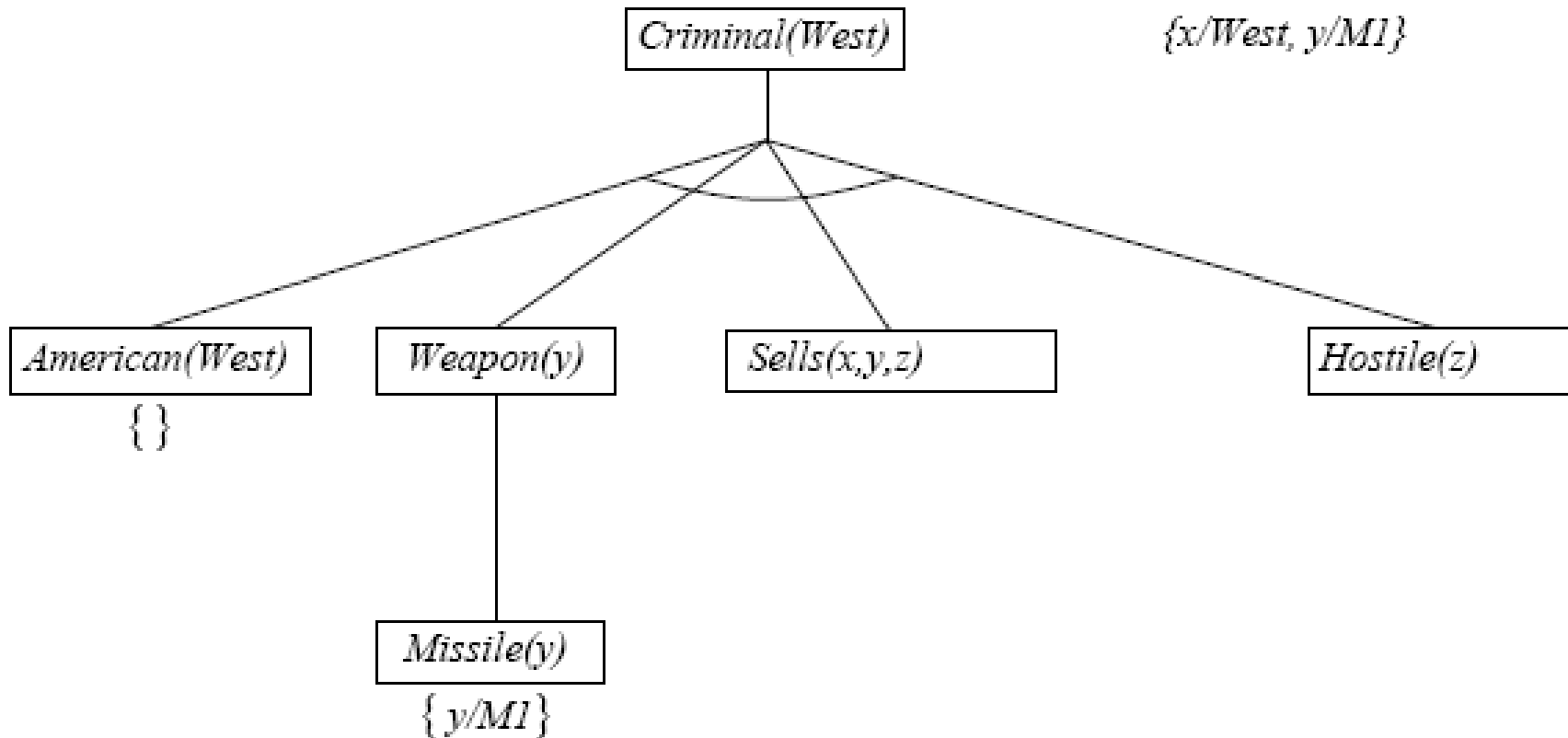
# Backward chaining example



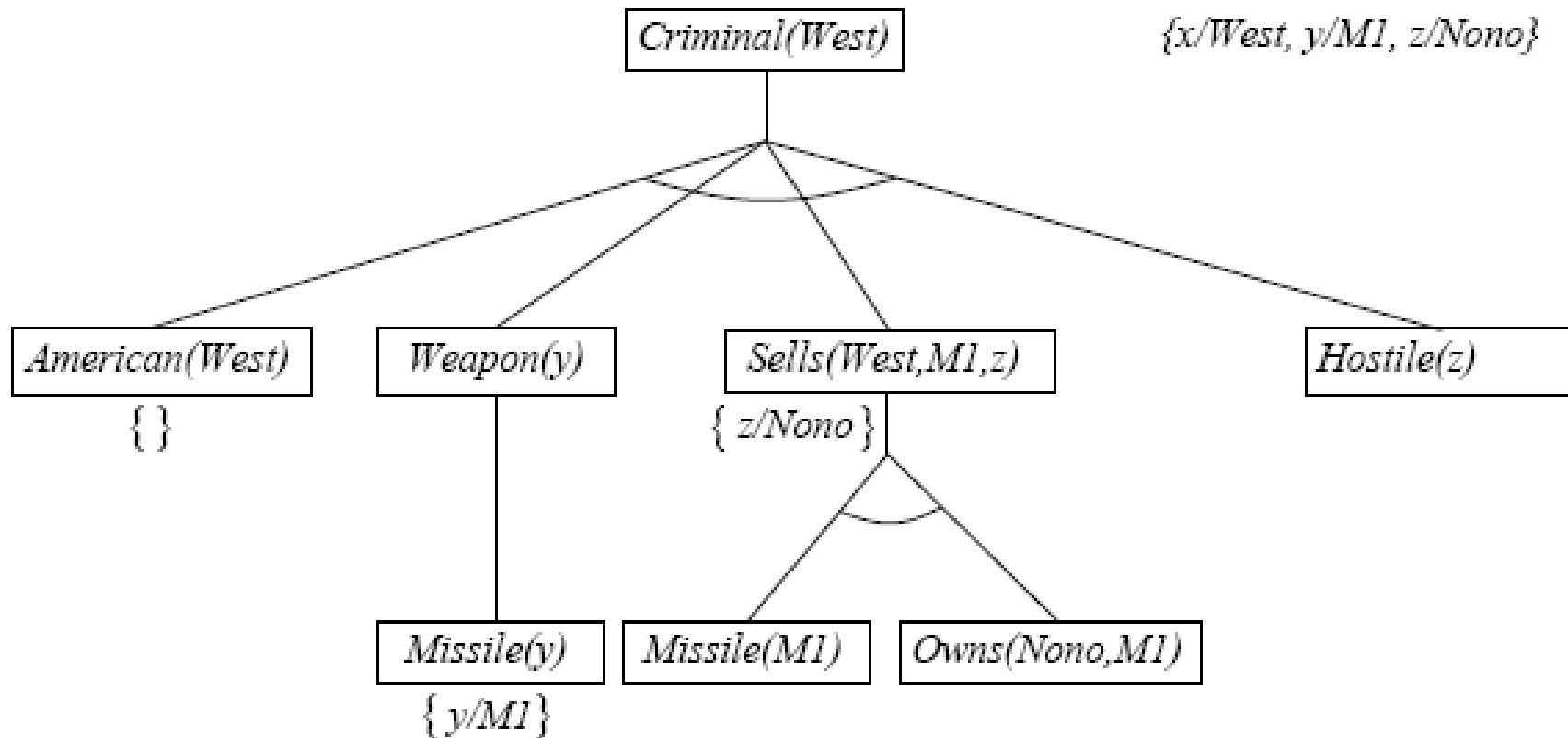
# Backward chaining example



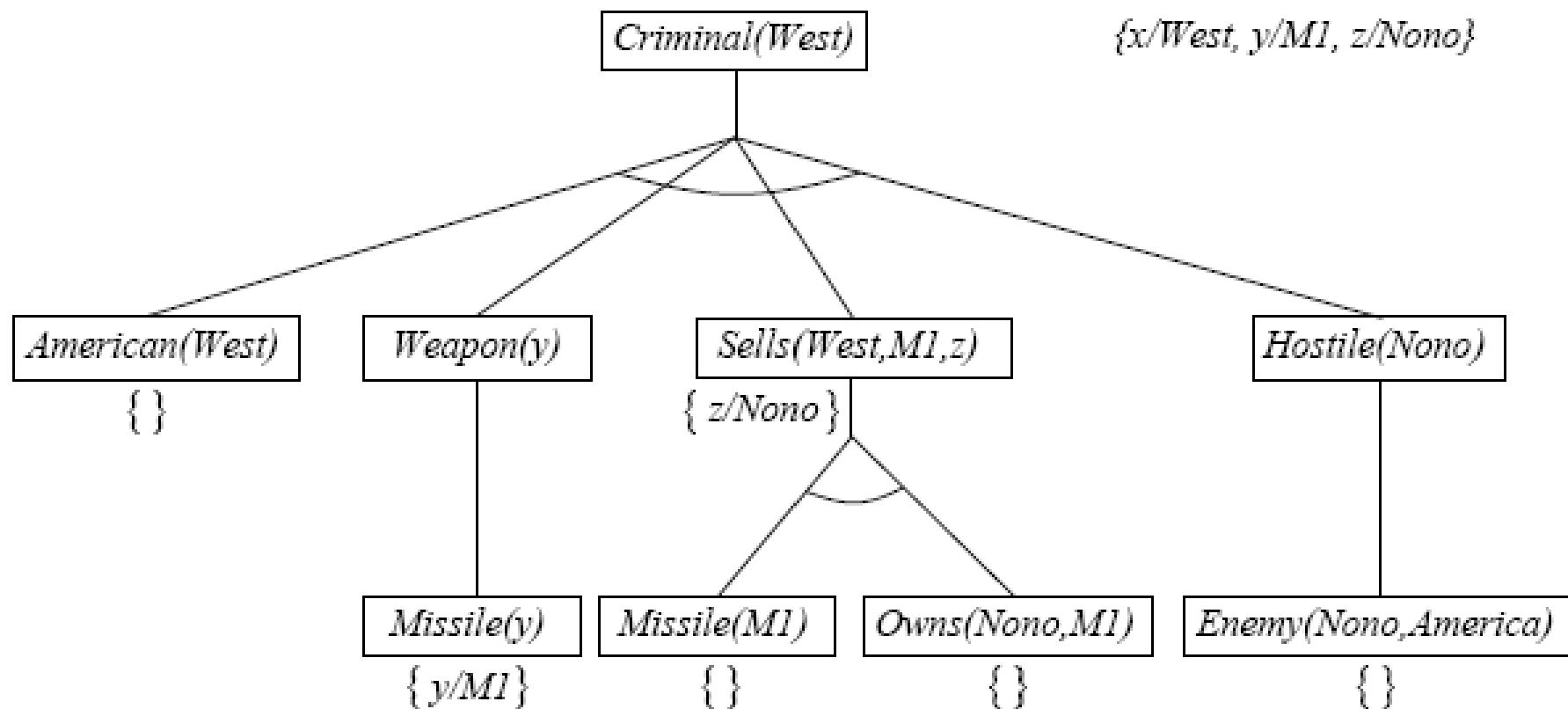
# Backward chaining example



# 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 (without improvements!) for **logic programming**

# Logic programming

- Sound bite: computation as inference on logical KBs

## Logic programming

1. Identify problem
2. Assemble information
3. Tea break
4. Encode information in KB
5. Encode problem as facts
6. Ask queries
7. Find false facts

## Ordinary programming

- Identify problem
- Assemble information
- Figure out solution
- Program solution
- Encode problem as data
- Apply program to data
- Debug procedural errors

Should be easier to debug *Capital(NewYork,US)* than  $x := x + 2!$

# Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles  
Widely used in Europe, Japan (basis of 5th Generation project)  
Compilation techniques  $\Rightarrow$  approaching a billion LIPS

Program = set of clauses = **head** :- **literal**<sub>1</sub>, ... **literal**<sub>n</sub>.  
`criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`

- Efficient unification by open coding
- Efficient retrieval of matching clauses by direct linking
- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., X is Y\*Z+3
- Closed-world assumption ("negation as failure")  
e.g., given `alive(X) :- not dead(X).`  
`alive(joe)` succeeds if `dead(joe)` fails

# Prolog examples

Depth-first search from a start state X:

```
dfs(X) :- goal(X) .
```

```
dfs(X) :- successor(X,S) , dfs(S) .
```

No need to loop over S: `successor` succeeds for each

Appending two lists to produce a third:

```
append([],Y,Y) .
```

```
append([X|L],Y,[X|Z]) :- append(L,Y,Z) .
```

```
query:      append(A,B,[1,2]) ?
```

```
answers:   A=[]          B=[1,2]
```

```
           A=[1]        B=[2]
```

```
           A=[1,2]      B=[]
```

# Resolution: brief summary

Full first-order version:

$$\frac{l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee m_n}{(l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta}$$

where  $\text{UNIFY}(l_i, \neg m_j) = \theta$ .

For example,

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

with  $\theta = \{x/\text{Ken}\}$

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

# Conversion to CNF

Everyone who loves all animals is loved by someone:

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

1. Eliminate biconditionals and implications

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

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

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

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

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

# Conversion to CNF contd.

- Standardize variables: each quantifier should use a different one

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

- Skolemize: a more general form of existential instantiation.  
Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

- Drop universal quantifiers:

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

- Distribute  $\wedge$  over  $\vee$ :

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

# Resolution proof: definite clauses

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

$\neg Criminal(West)$

$American(West)$

$\neg American(West) \vee \neg Weapon(y) \vee \neg Sells(West,y,z) \vee \neg Hostile(z)$

$\neg Missile(x) \vee Weapon(x)$

$\neg Weapon(y) \vee \neg Sells(West,y,z) \vee \neg Hostile(z)$

$Missile(M1)$

$\neg Missile(y) \vee \neg Sells(West,y,z) \vee \neg Hostile(z)$

$\neg Missile(x) \vee \neg Owns(Nono,x) \vee Sells(West,x,Nono)$

$\neg Sells(West,M1,z) \vee \neg Hostile(z)$

$Missile(M1)$

$\neg Missile(M1) \vee \neg Owns(Nono,M1) \vee \neg Hostile(Nono)$

$Owns(Nono,M1)$

$\neg Owns(Nono,M1) \vee \neg Hostile(Nono)$

$\neg Enemy(x,America) \vee Hostile(x)$

$\neg Hostile(Nono)$

$Enemy(Nono,America)$

$Enemy(Nono,America)$

□