

INFERENCE IN FIRST ORDER LOGIC

Outline

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

$\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ yields

$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$

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

Existential Instantiation (continued)

- 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 (continued)

- 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,

Father(Father(Father(John)))

- Problem: works if α is entailed, loops if not entailed

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 $p * n^k$ instantiations
- With function symbols, it gets much much worse!

Unification

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

Unification

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

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

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

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

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., $Knows(z_{17}, OJ)$

Generalized Modus Ponens (GMP)

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

p_1' is *King(John)* p_1 is *King(x)*
 p_2' is *Greedy(y)* p_2 is *Greedy(x)*
 θ is $\{x/\text{John}, y/\text{John}\}$ 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
- GMP is sound

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:

Example Knowledge Base

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

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

Example Knowledge Base

- ...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

Owns(Nono, M₁) and *Missile(M₁)*

- ... all of its missiles were sold to it by Colonel West

Example Knowledge Base

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

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

- Nono ...has some missiles

$$\textit{Owns}(\textit{Nono}, M_1) \text{ and } \textit{Missile}(M_1)$$

- ... all of its missiles were sold to it by Colonel West

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

- Missiles are weapons

Example Knowledge Base

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

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

- Nono ...has some missiles

$$\textit{Owns}(\textit{Nono}, M_1) \text{ and } \textit{Missile}(M_1)$$

- ... all of its missiles were sold to it by Colonel West

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

- Missiles are weapons

$$\textit{Missile}(x) \Rightarrow \textit{Weapon}(x)$$

- An enemy of America counts as “hostile”

Example Knowledge Base

- ...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

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

- ... all of its missiles were sold to it by Colonel West

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

- 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( $KB, \alpha$ ) returns a substitution or false
  repeat until new is empty
     $new \leftarrow \{ \}$ 
    for each sentence  $r$  in  $KB$  do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)$ 
      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

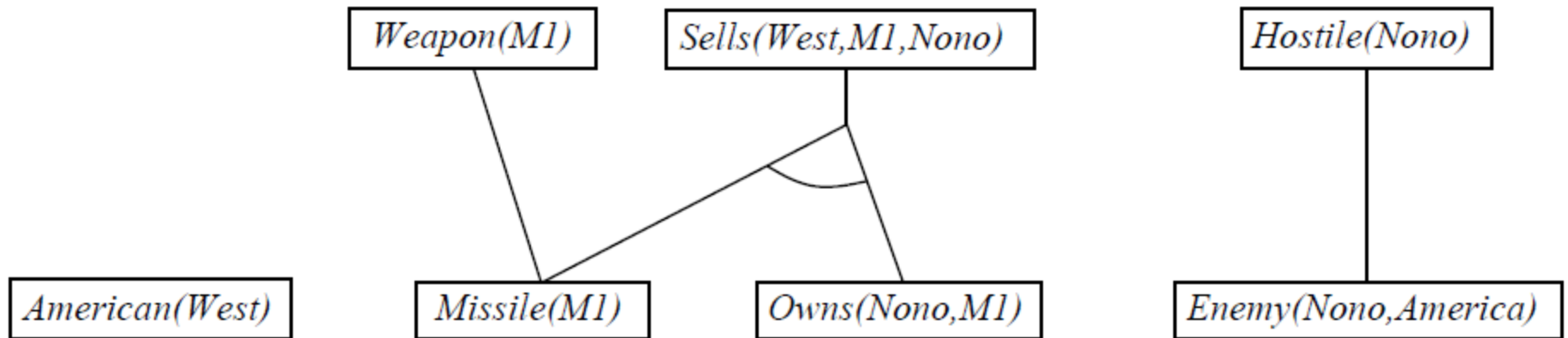
American(West)

Missile(M1)

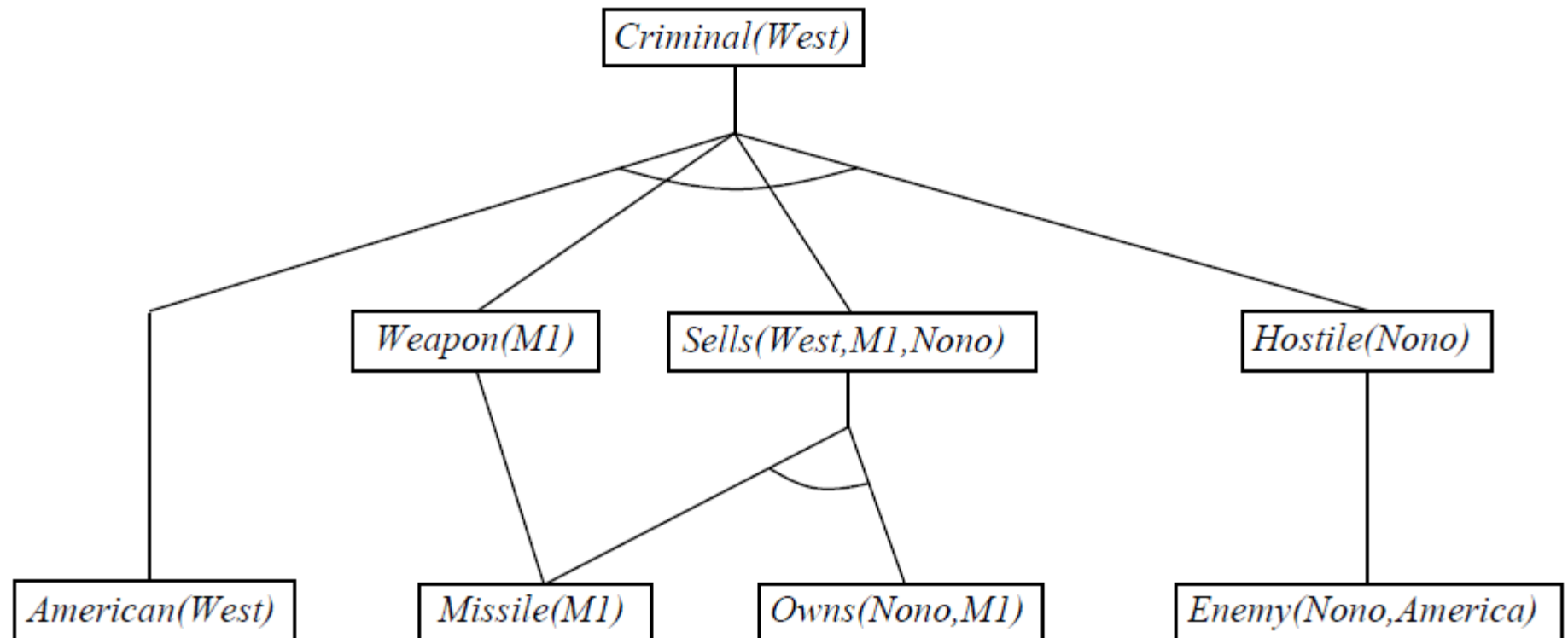
Owns(Nono,M1)

Enemy(Nono,America)

Forward Chaining Proof



Forward Chaining Proof



Properties of Forward Chaining

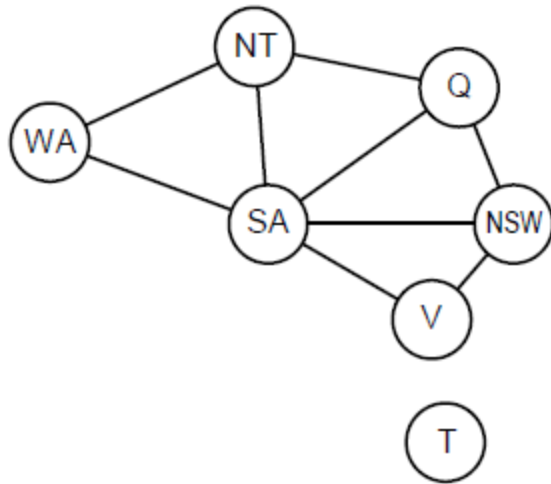
- Sound and complete for first-order definite clauses (proof similar to propositional proof)
- Datalog is a declarative logic programming language
 - Subset of Prolog
 - Datalog = first-order definite clauses + no functions (e.g., crime KB)
 - FC terminates for Datalog in poly iterations: at most $p * n^k$ literals
 - May not terminate in general if α 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(M1)`
- Matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases

- Colorable() is inferred iff the CSP has a solution

Hard Matching Example



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

$$\begin{aligned}
 & Diff(Red, Blue) \quad Diff(Red, Green) \\
 & Diff(Green, Red) \quad Diff(Green, Blue) \\
 & Diff(Blue, Red) \quad Diff(Blue, Green)
 \end{aligned}$$

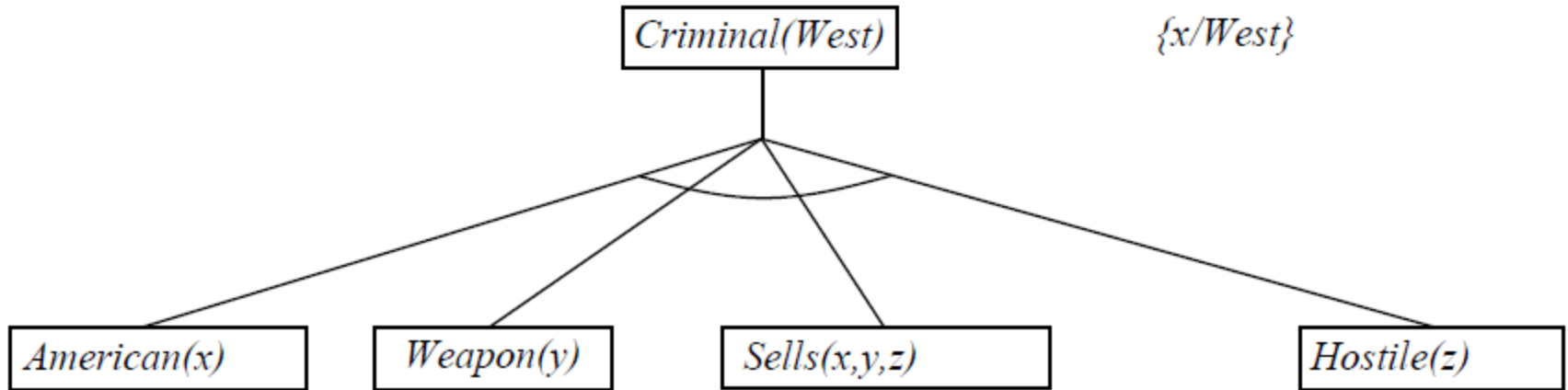
Backward Chaining Algorithm

```
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  $\{ \theta \}$ 
 $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\textit{goals}))$ 
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
     $\textit{new\_goals} \leftarrow [p_1, \dots, p_n | \text{REST}(\textit{goals})]$ 
     $\textit{answers} \leftarrow \text{FOL-BC-ASK}(\textit{KB}, \textit{new\_goals}, \text{COMPOSE}(\theta', \theta)) \cup \textit{answers}$ 
return answers
```

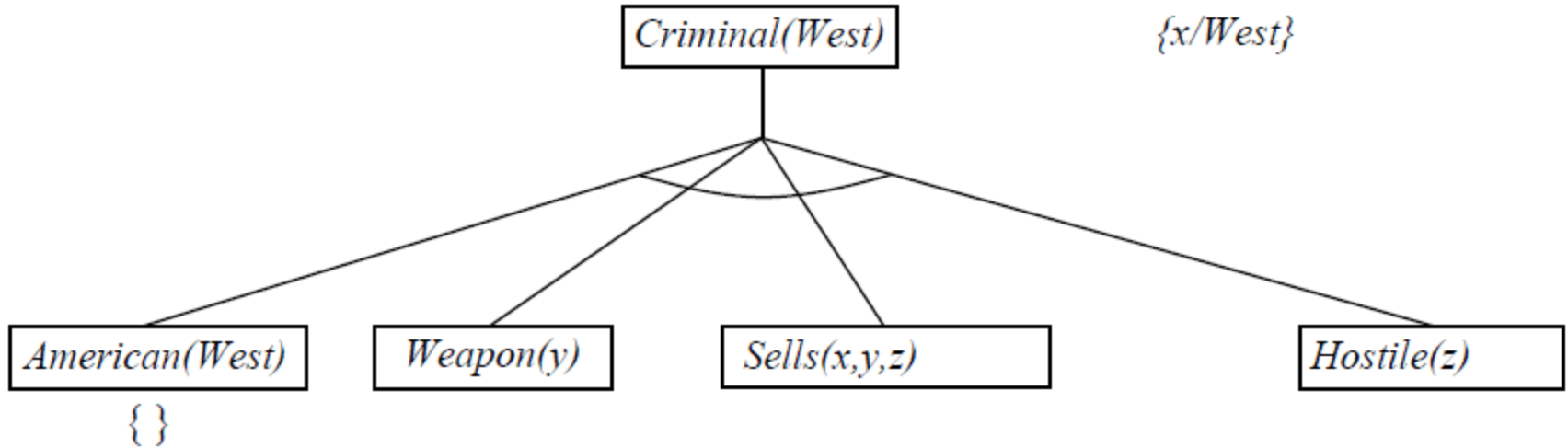
Backward Chaining Example

Criminal(West)

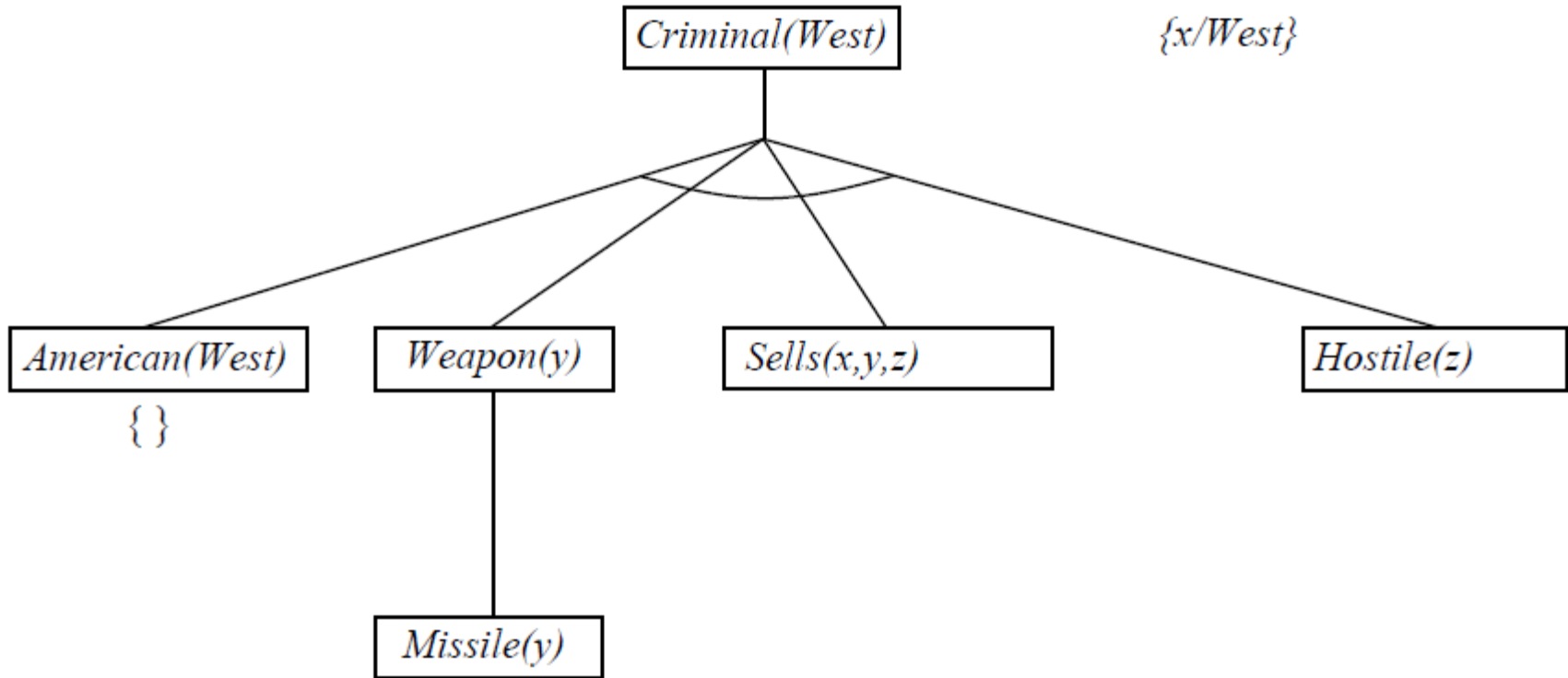
Backward Chaining Example



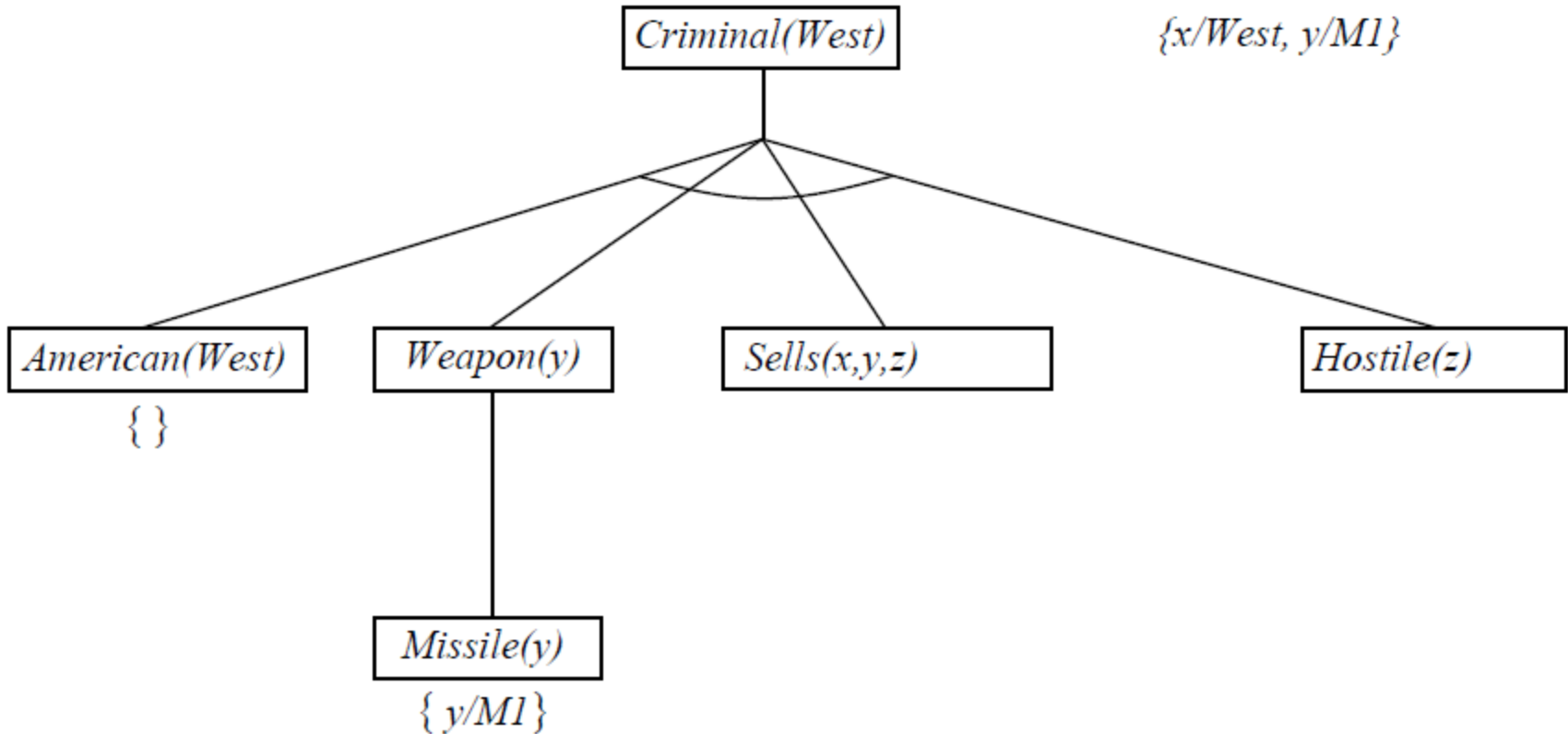
Backward Chaining Example



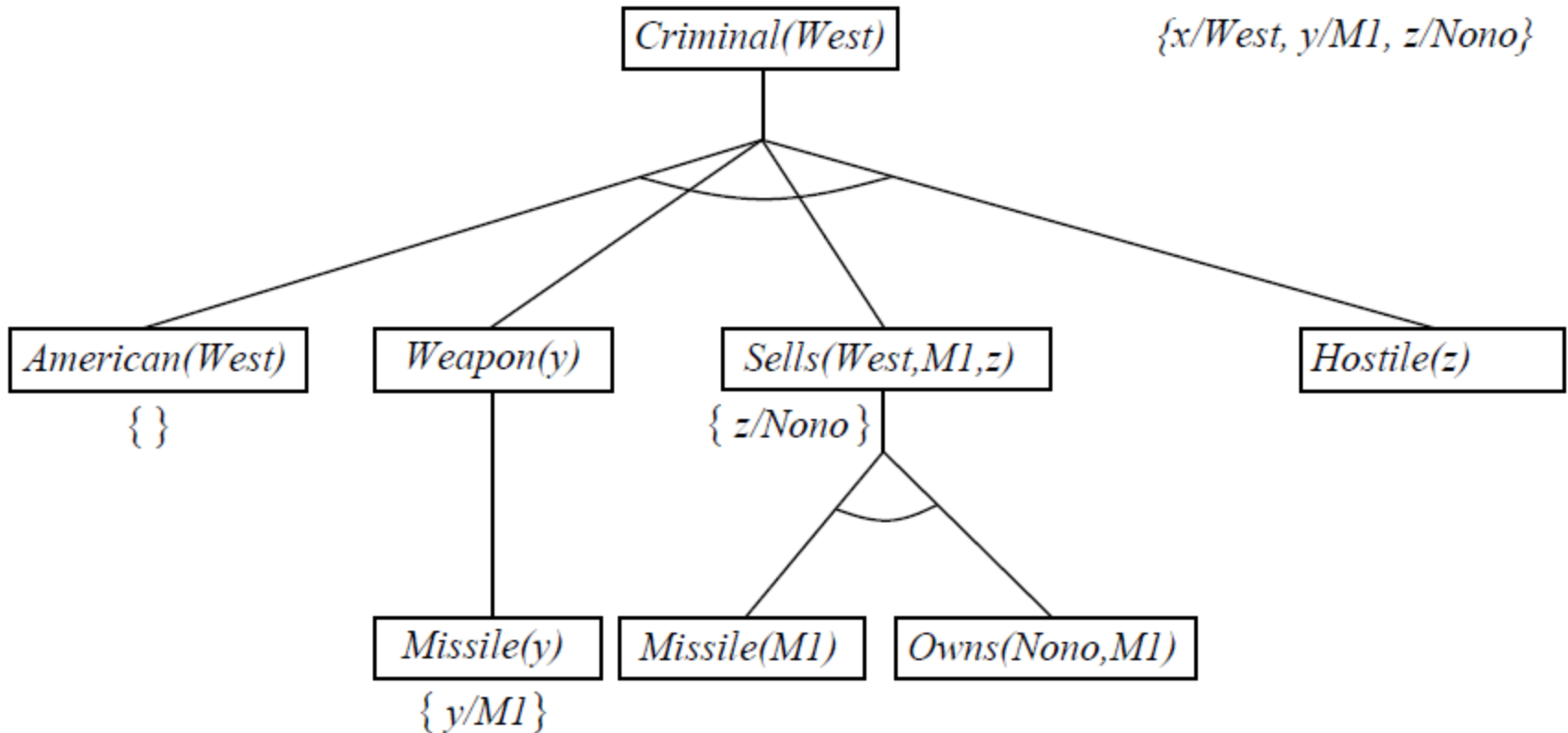
Backward Chaining Example



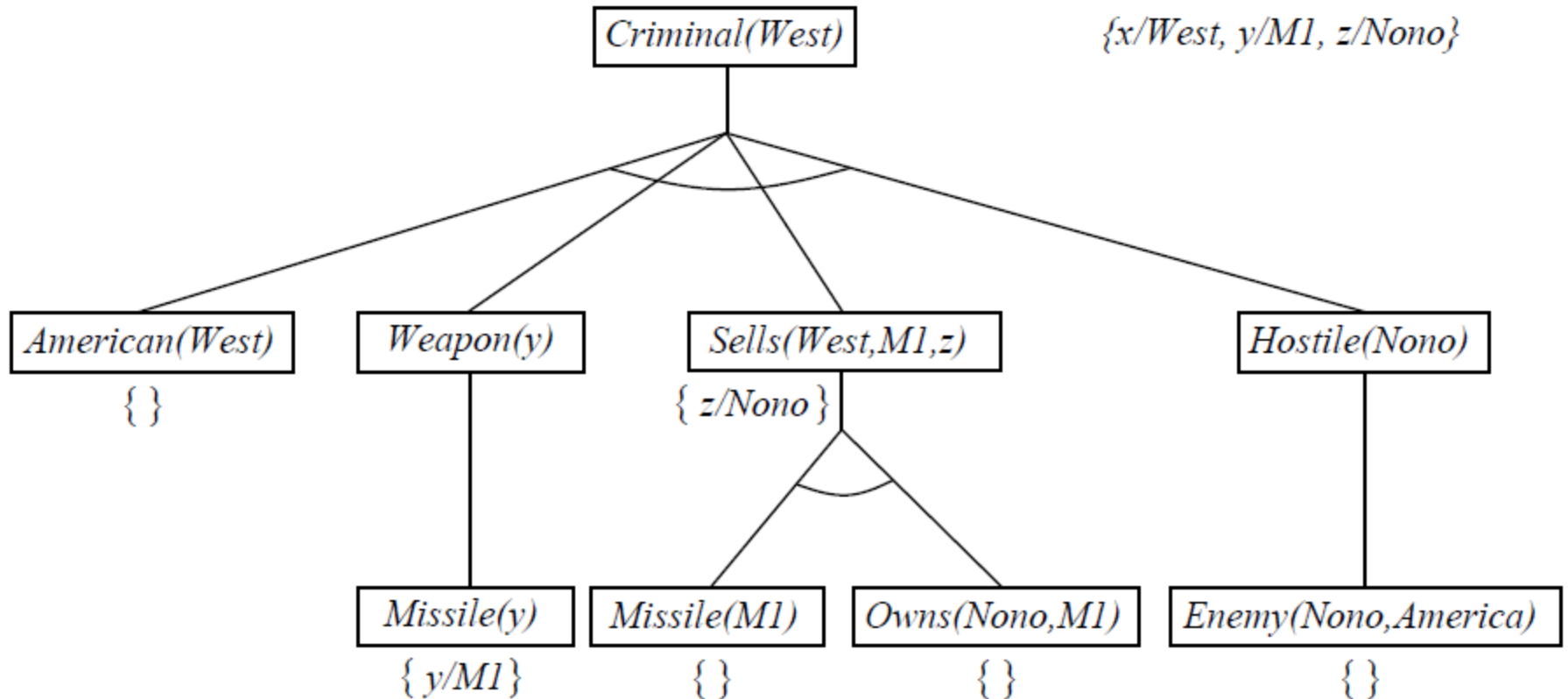
Backward Chaining Example



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

- Sound bite: computation as inference on logical KBs

Logic Programming

Logic programming

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

Ordinary programming

- Identify problem
- Assemble information
- Figure out solution
- Program solution
- Encode problem instance 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)
- Program = set of clauses = head :- literal₁, ... literal_n.
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
- Efficient unification
- Efficient retrieval of matching clauses
- 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:

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

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

- For example,

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

- with $\theta = \{x/Ken\}$
- Apply resolution steps to $CNF(KB \wedge \neg\alpha)$
 - Complete for FOL

Conversion to CNF

- Everyone who loves all animals is loved by someone:

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

- 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

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

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

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

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

- 5. Drop universal quantifiers:

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

- 6. 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

