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:

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

- for any variable v and ground term g

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

$$King(John) \wedge Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \;\Rightarrow\; Evil(Father(John))$$
$$\vdots$$

Existential Instantiation (EI)

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

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

- 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

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:

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

- 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.

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 is not entailed

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 $p * n^k$ 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 King(x) and Greedy(x) match King(John) and Greedy(y)
- $\Theta = \{x/John, y/John\}$ works
- Unify(α, β) = Θ 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)

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

p_1' is $King(John)$ $\quad$ p_1 is $King(x)$

p_2' is $Greedy(y)$ $\quad$ p_2 is $Greedy(x)$

θ is $\{x/John, y/John\}$ $\quad$ 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:

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

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

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

- … 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:

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

- Nono …has some missiles

$Owns(Nono, M_1)$ 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

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

  $Owns(Nono, M_1)$ 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”

Example Knowledge Base

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

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

- Nono …has some missiles

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

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

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

- 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

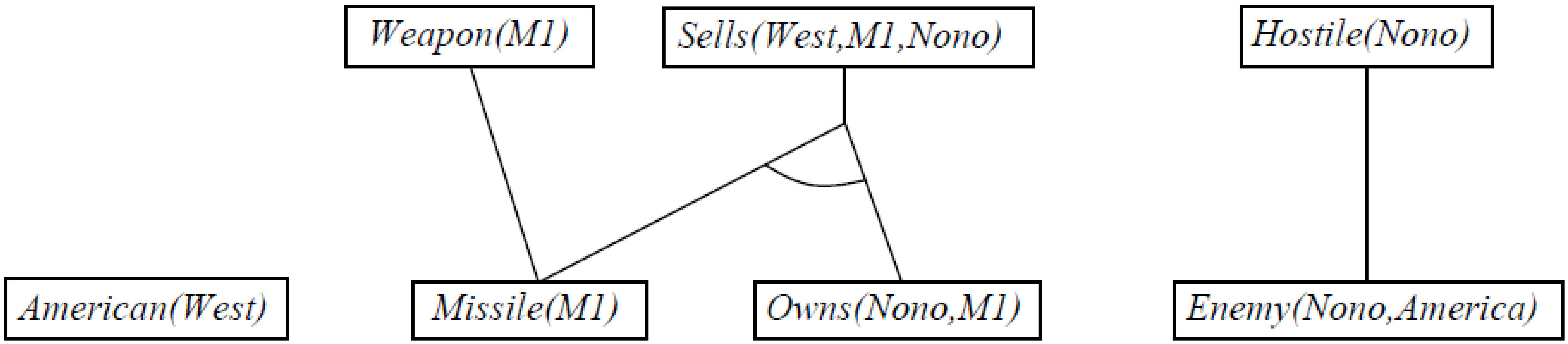
```
function FOL-FC-ASK(KB, α) returns a substitution or false

  repeat until new is empty
     new ← { }
     for each sentence r in KB do
        ( p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
        for each θ such that (p1 ∧ ... ∧ pn)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
```

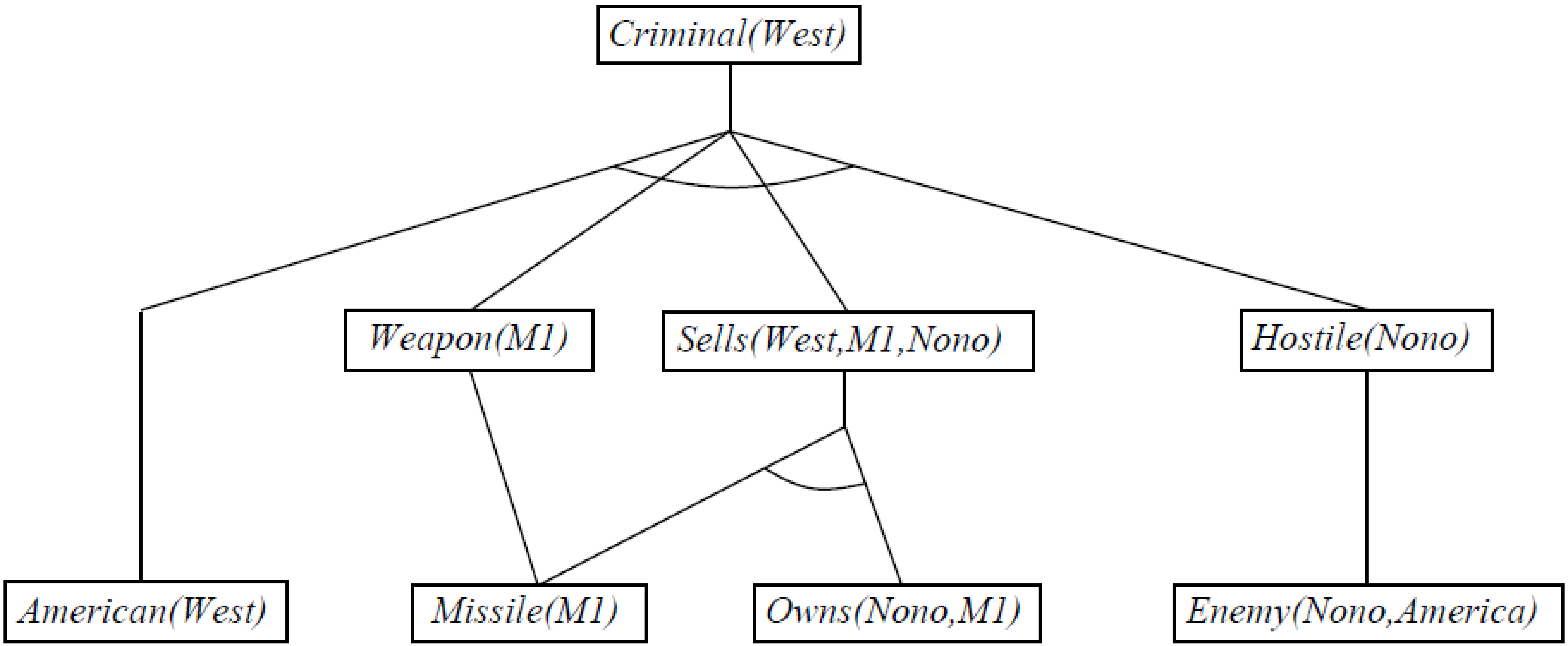
Forward Chaining Proof

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

Forward Chaining Proof



Forward Chaining Proof



Properties of Forward Chaining

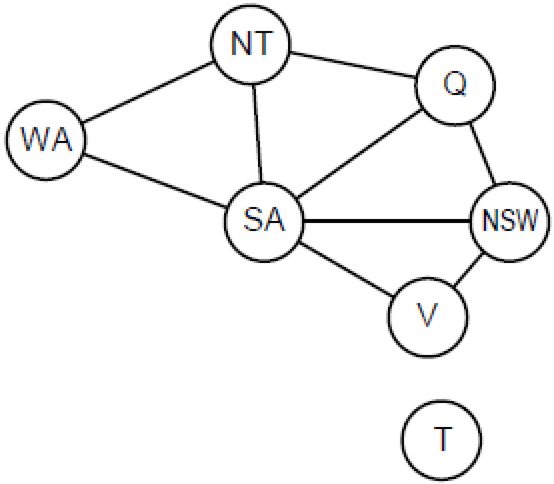
- Sound and complete for first-order definite clauses (proof similar to propositional proof)
- Datalog is a declarative logic programming language
  - Subset of Prolog
  - Datalog = first-order definite clauses + no functions (e.g., crime KB)
  - FC terminates for Datalog in poly iterations: at most $p * n^k$ 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

Hard Matching Example

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



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

$$Diff(Red, Blue) \quad Diff(Red, Green)$$

$$Diff(Green, Red) \quad Diff(Green, Blue)$$

$$Diff(Blue, Red) \quad Diff(Blue, Green)$$

Backward Chaining Algorithm

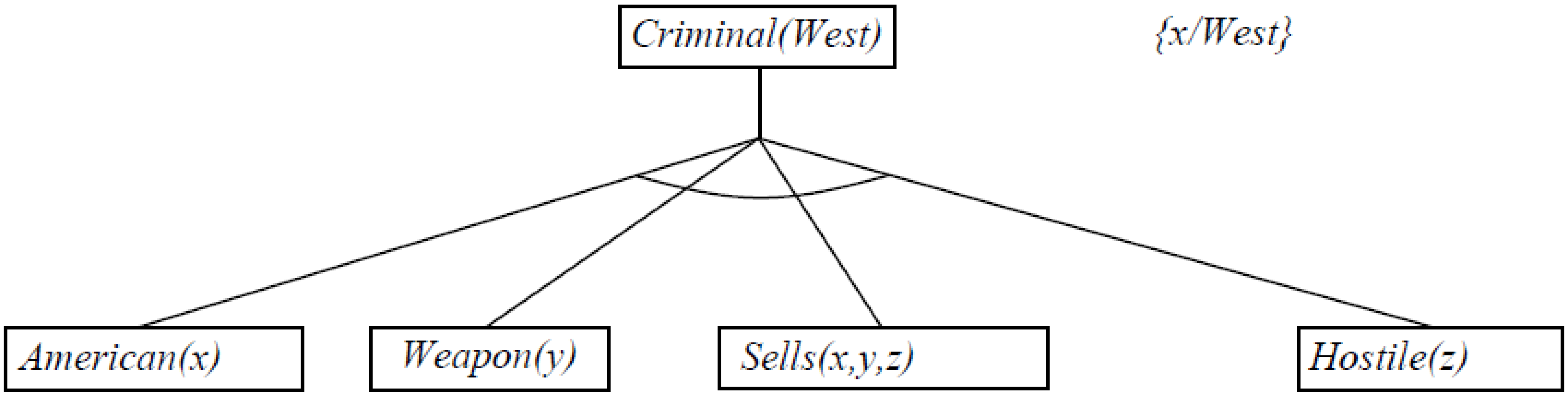
function FOL-BC-ASK(KB, $goals$, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution $\{\,\}$
  local variables: $answers$, a set of substitutions, initially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

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

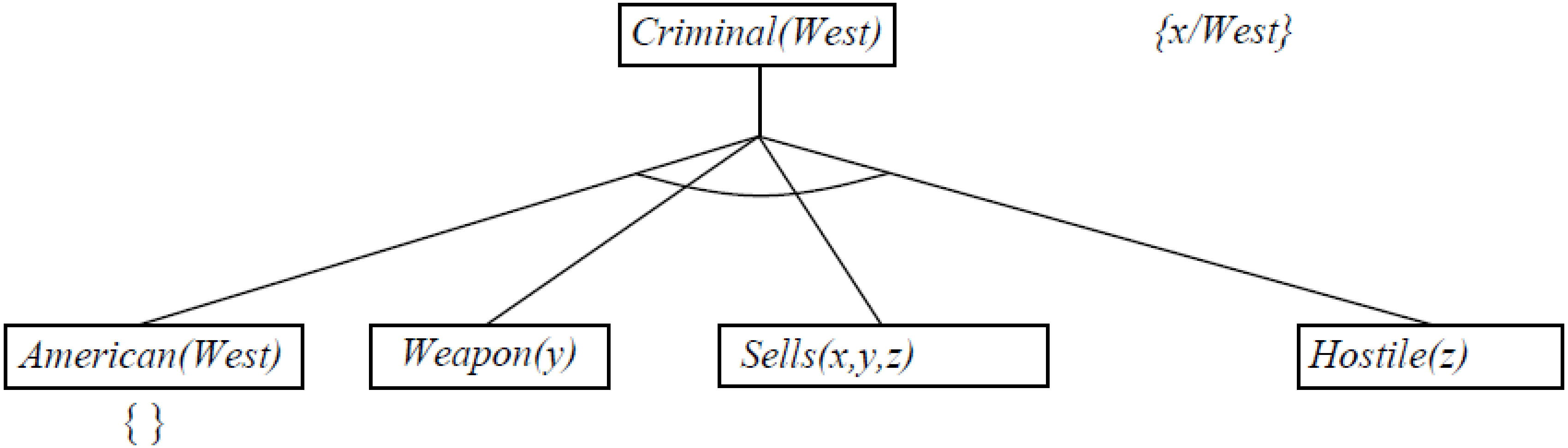
Backward Chaining Example

$Criminal(West)$

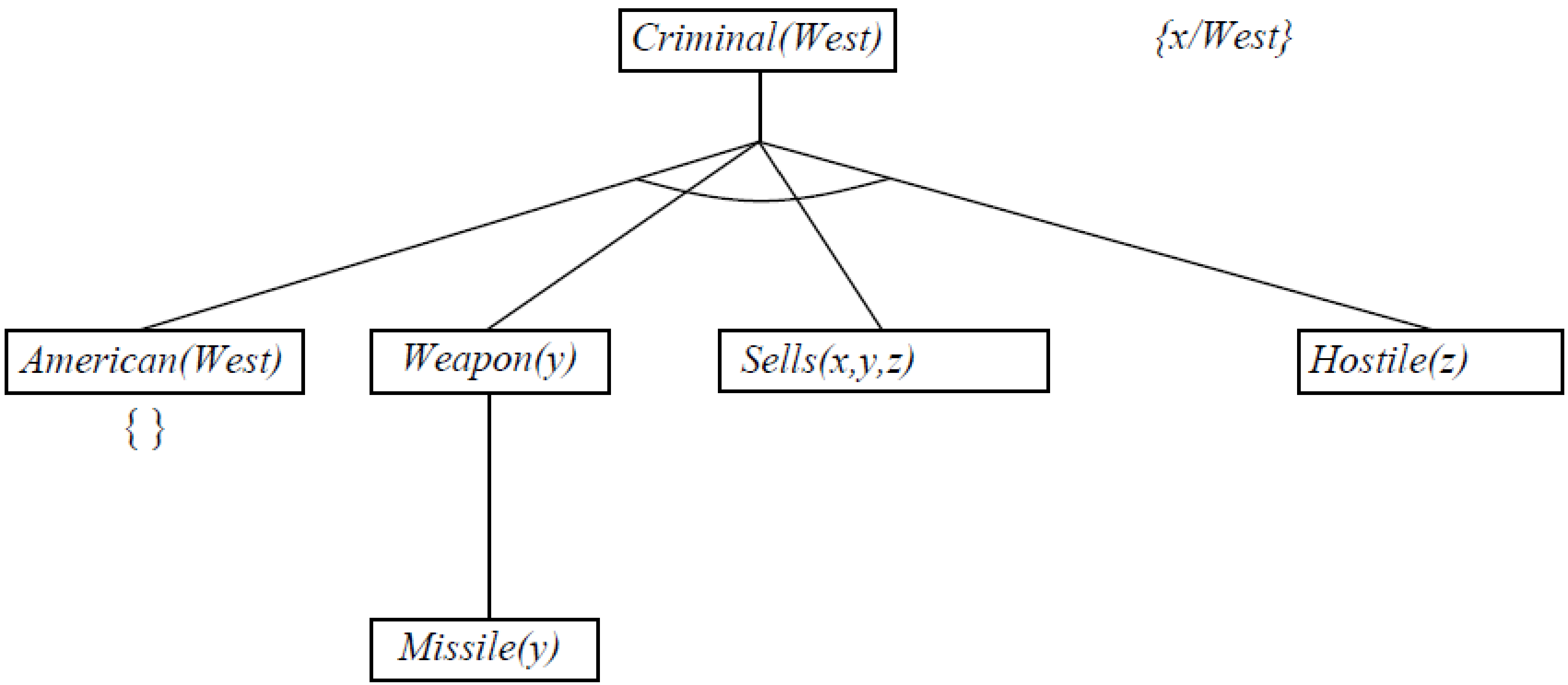
Backward Chaining Example



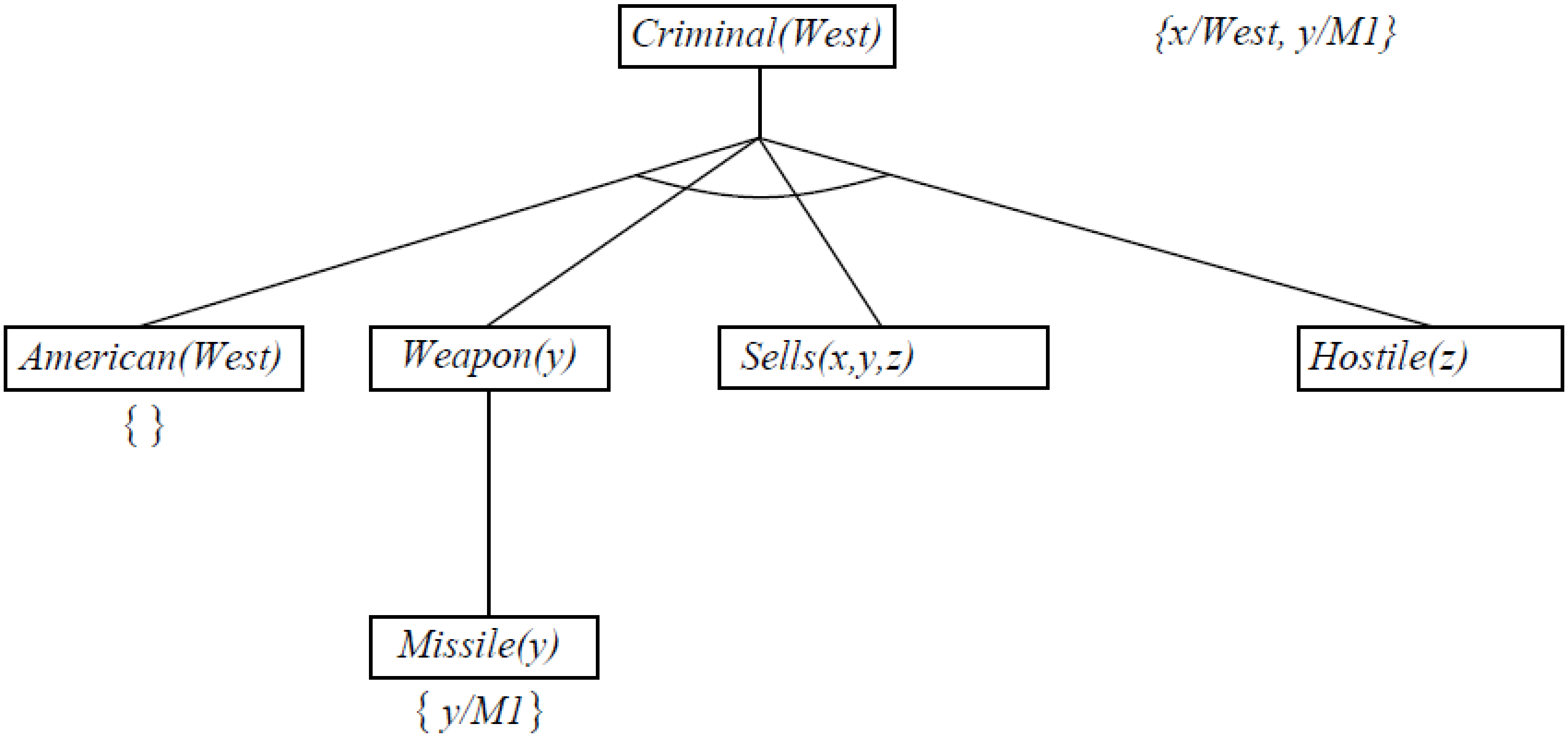
Backward Chaining Example



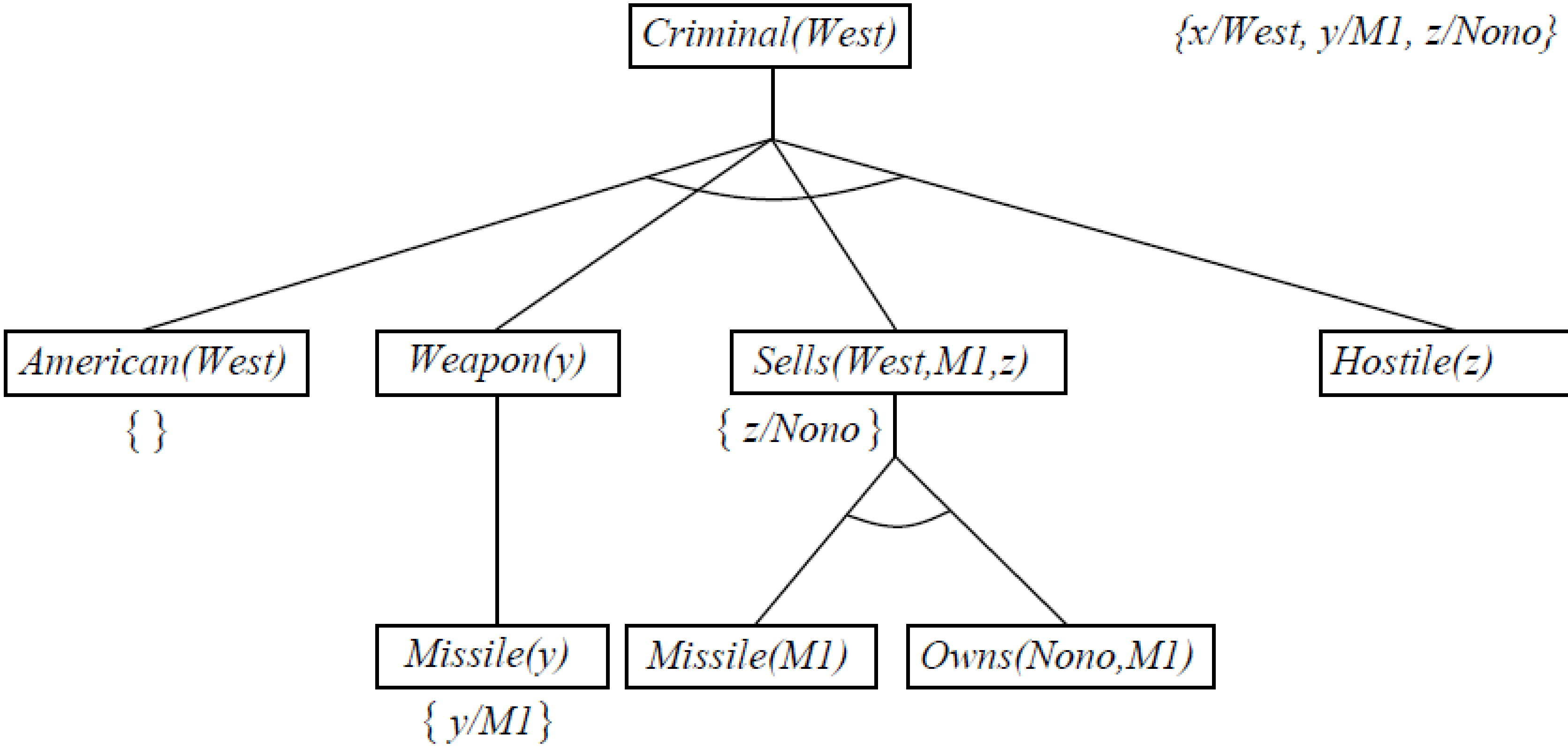
Backward Chaining Example



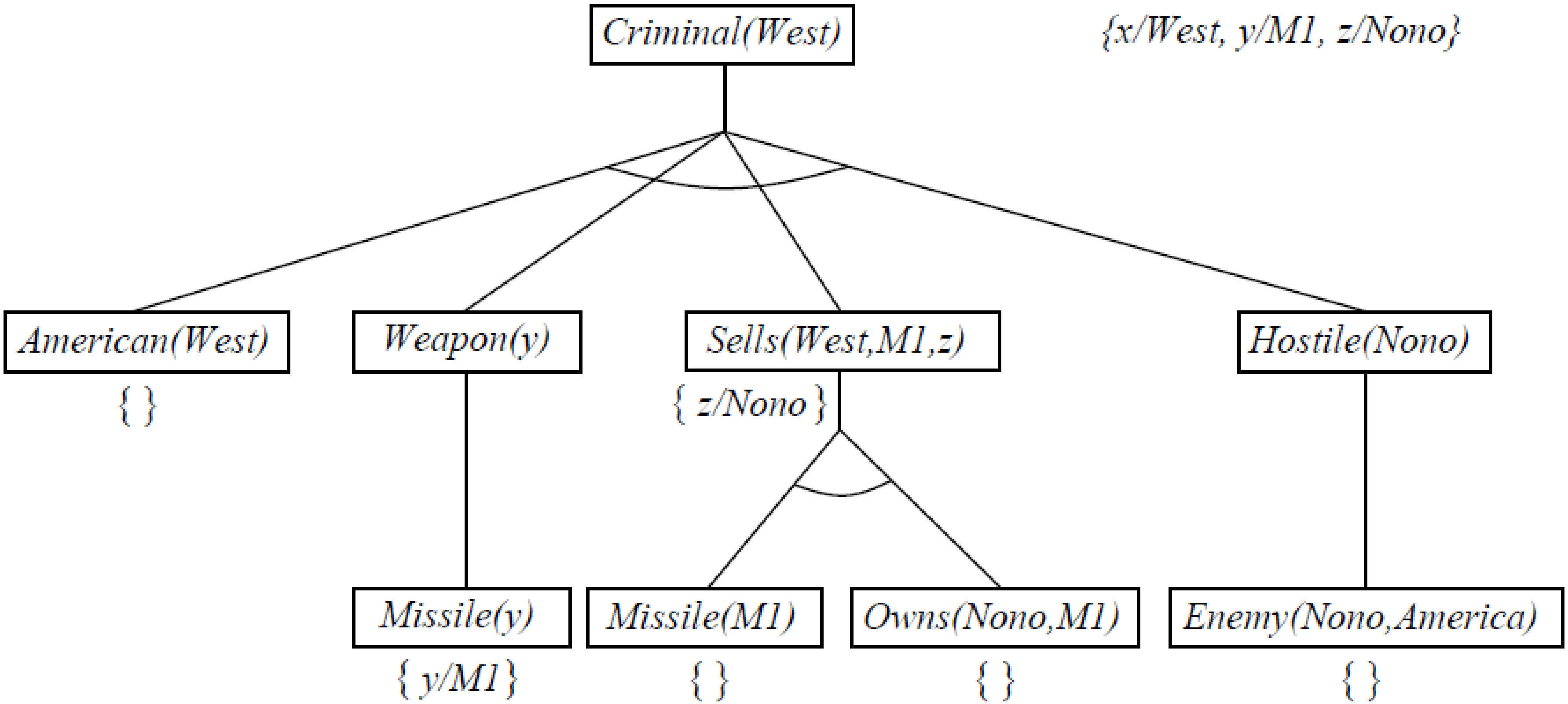
Backward Chaining Example



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	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	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 :- literal_1, … 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:

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

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

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x) \\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

- 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:

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

- 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 ¬ inwards: $\neg\forall x, p \ \equiv \exists x \ \ \neg p, \quad \neg\exists x, p \ \equiv \forall x \ \ \neg p$:

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

Conversion to CNF

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

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

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

$$\forall x \ \ [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$$

- 5. Drop universal quantifiers:

$$[Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$$

- 6. Distribute ∧ over ∨:

$$[Animal(F(x)) \lor Loves(G(x),x)] \land [\neg Loves(x,F(x)) \lor Loves(G(x),x)]$$

Resolution Proof: Definite Clauses