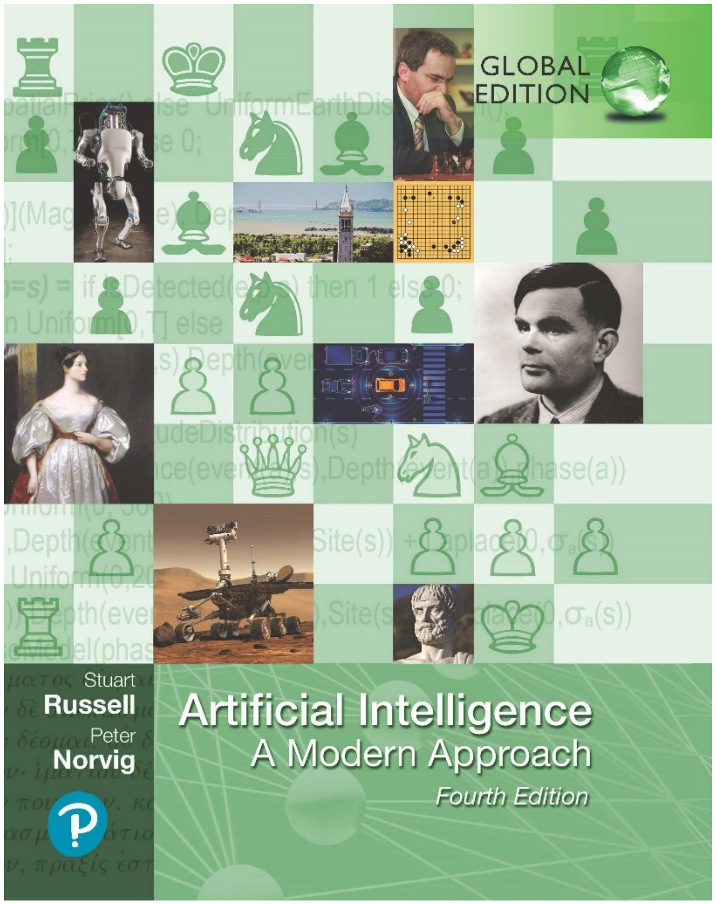
Artificial Intelligence: A Modern Approach

Fourth Edition, Global Edition



Chapter 9

Inference in first-order logic

Lecture Presentations: Artificial Intelligence

Adapted from:
"Artificial Intelligence: A Modern Approach, Global Edition", 4th Edition by Stuart Russell and Peter Norvig © 2021 Pearson Education.

Adapted for educational use at ACE Engineering College. Some slides customized by Mr. Shafakhatullah Khan Mohammed, Assistant Professor @ ACE Engineering College. For instructional use only. Not for commercial distribution.

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:

$$\frac{\forall v \;\; a}{\mathrm{Subst}(\{v/g\}, a)}$$

for any variable v and ground term g

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

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

© 2022 Pearson Education Ltd.

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

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:

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

Instantiating the universal sentence in all possible ways, we have

$$King(John) \land Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \land Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

The new KB is propositionalized: proposition symbols are

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

Reduction 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., $Father(Father(Father(John)))$

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

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

With function symbols, it gets nuch 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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\mathrm{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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 $King(John)$ p_1 is $King(x)$
p_2' is $Greedy(y)$ p_2 is $Greedy(x)$
θ 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

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

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

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

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
2. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. From 1 and 2, $q\theta$ 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:

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

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

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:

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":

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 Nono, an enemy of America . . .

$Enemy(Nono, America)$

Forward chaining algorithm

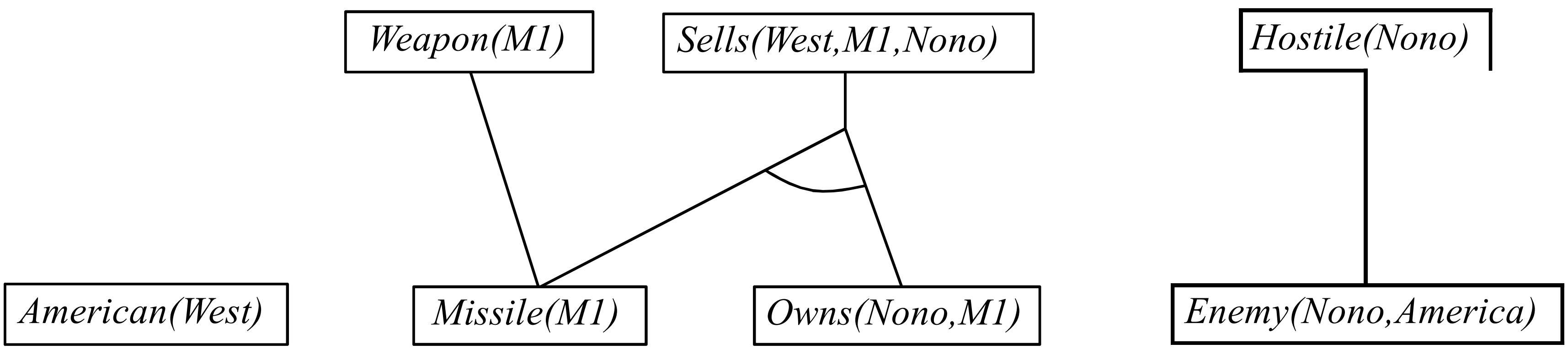
function FOL-FC-Ask(KB, α) **returns** a substitution or *false*

    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*
                    $\varphi \leftarrow$ Unify(q', α)
                    if φ is not *fail* **then return** φ
        add *new* to KB
    return *false*

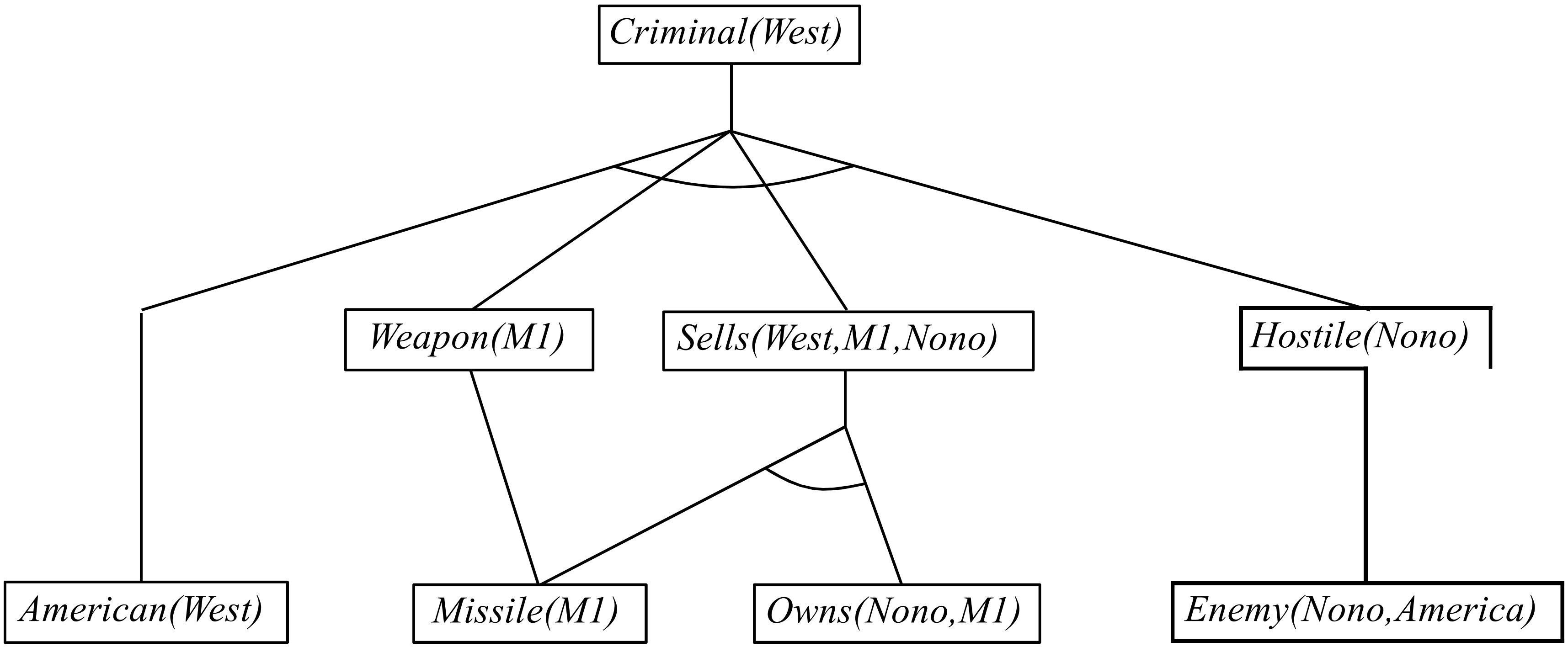
Forward chaining proof

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 = 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 a 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 $k - 1$
$\Rightarrow$ match each rule whose premise contains a newly added literal

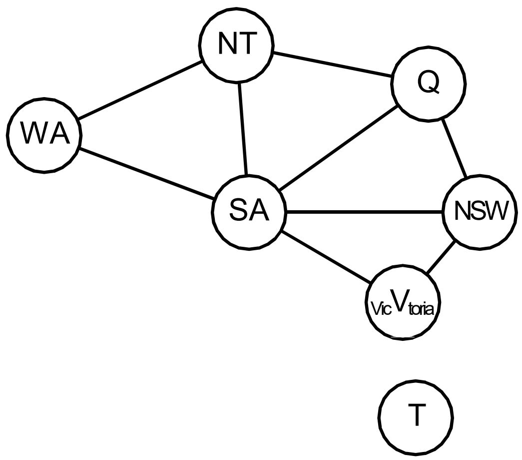
Matching itself can be expensive

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



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

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

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

CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining algorithm

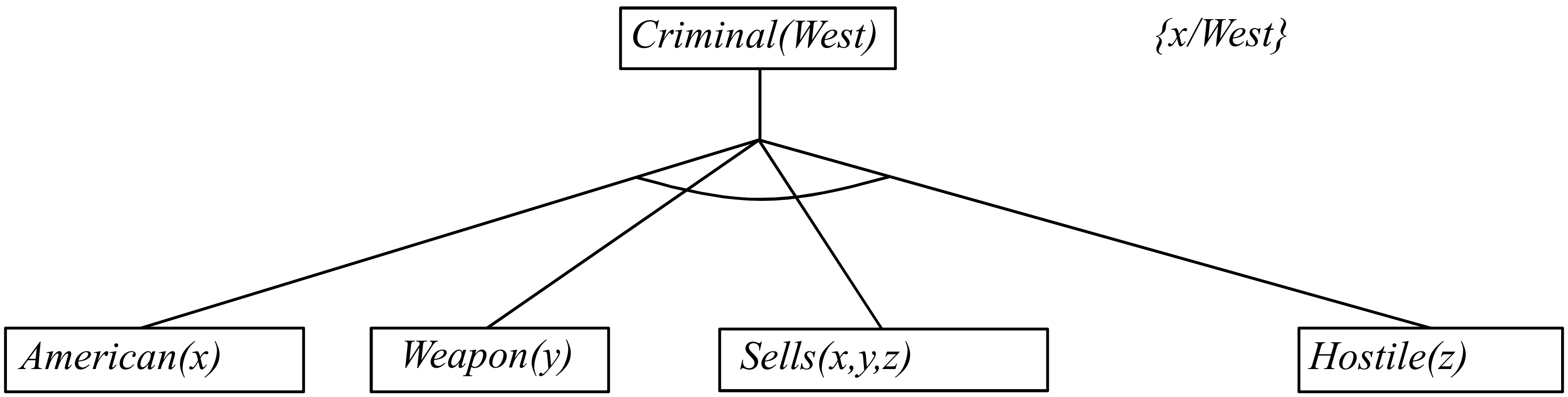
function FOL-BC-Ask(KB, $goals$, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution { }
  local variables: $answers$, a set of substitutions, initially 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
   $new\_goals \leftarrow [p_1, \ldots, p_n | \text{Rest}(goals)]$
   $answers \leftarrow$ FOL-BC-Ask(KB, new_goals, Compose(θ', θ)) $\cup$ $answers$
  return $answers$

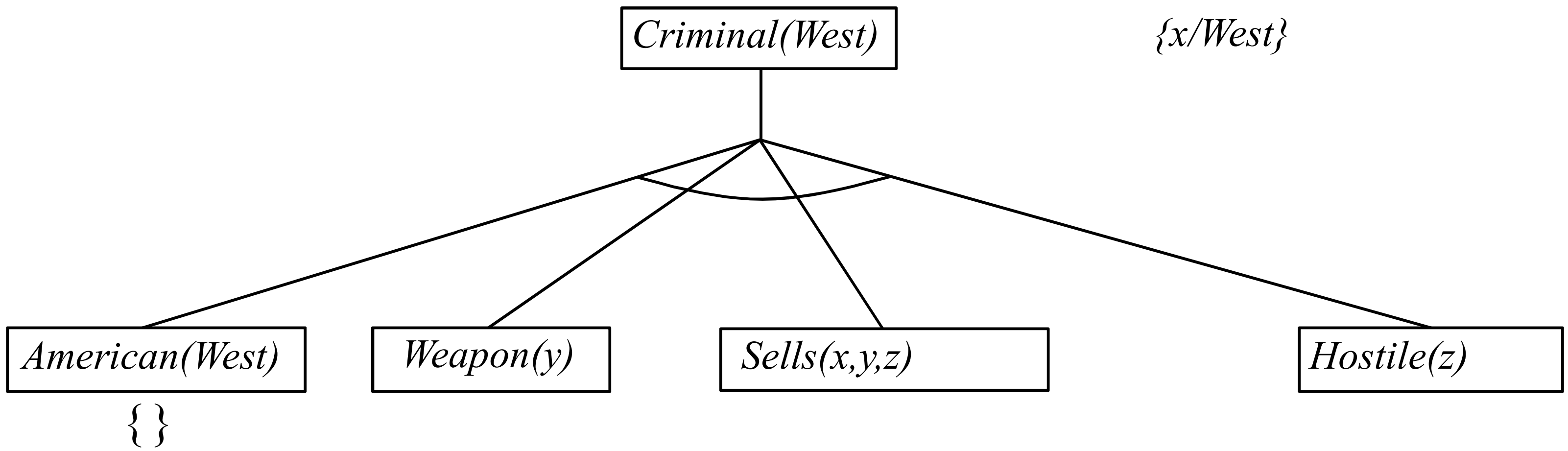
Backward chaining example

$\boxed{Criminal(West)}$

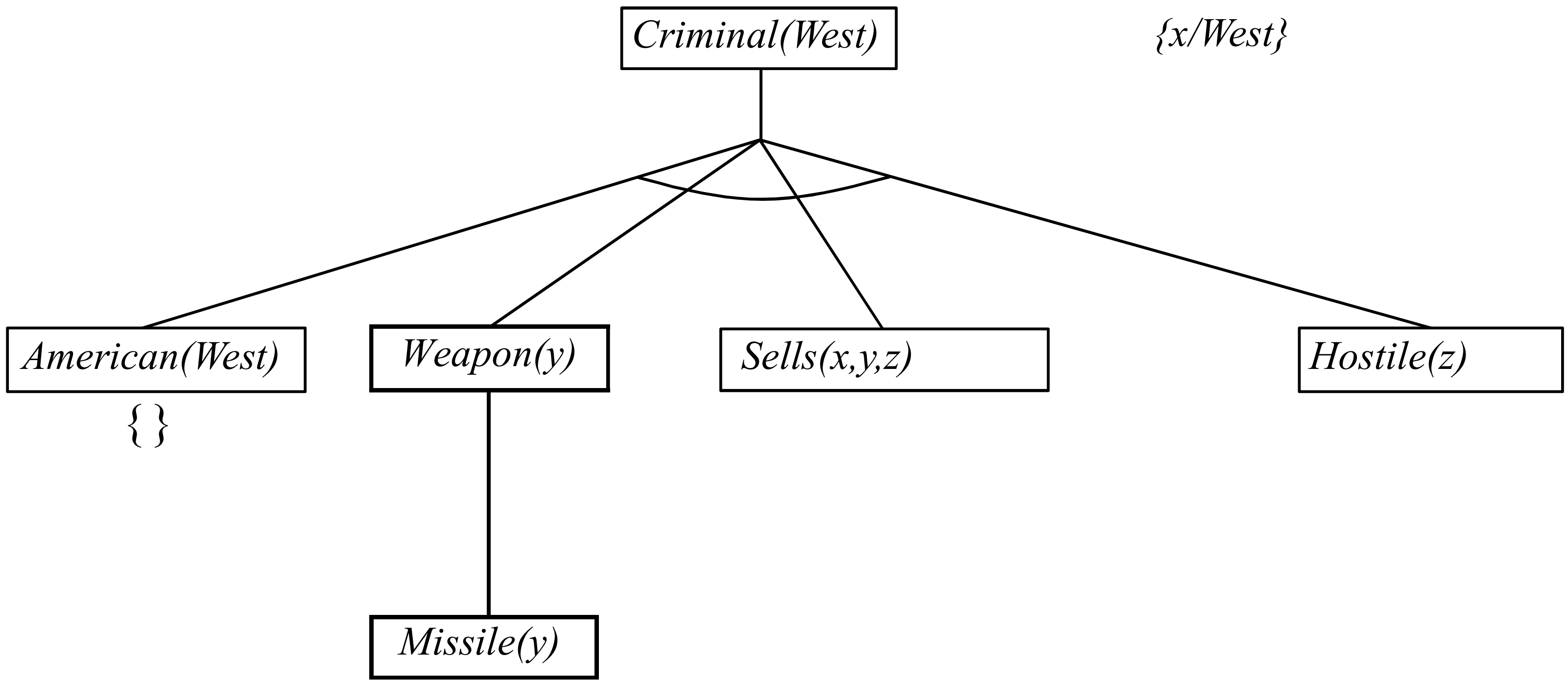
Backward chaining example



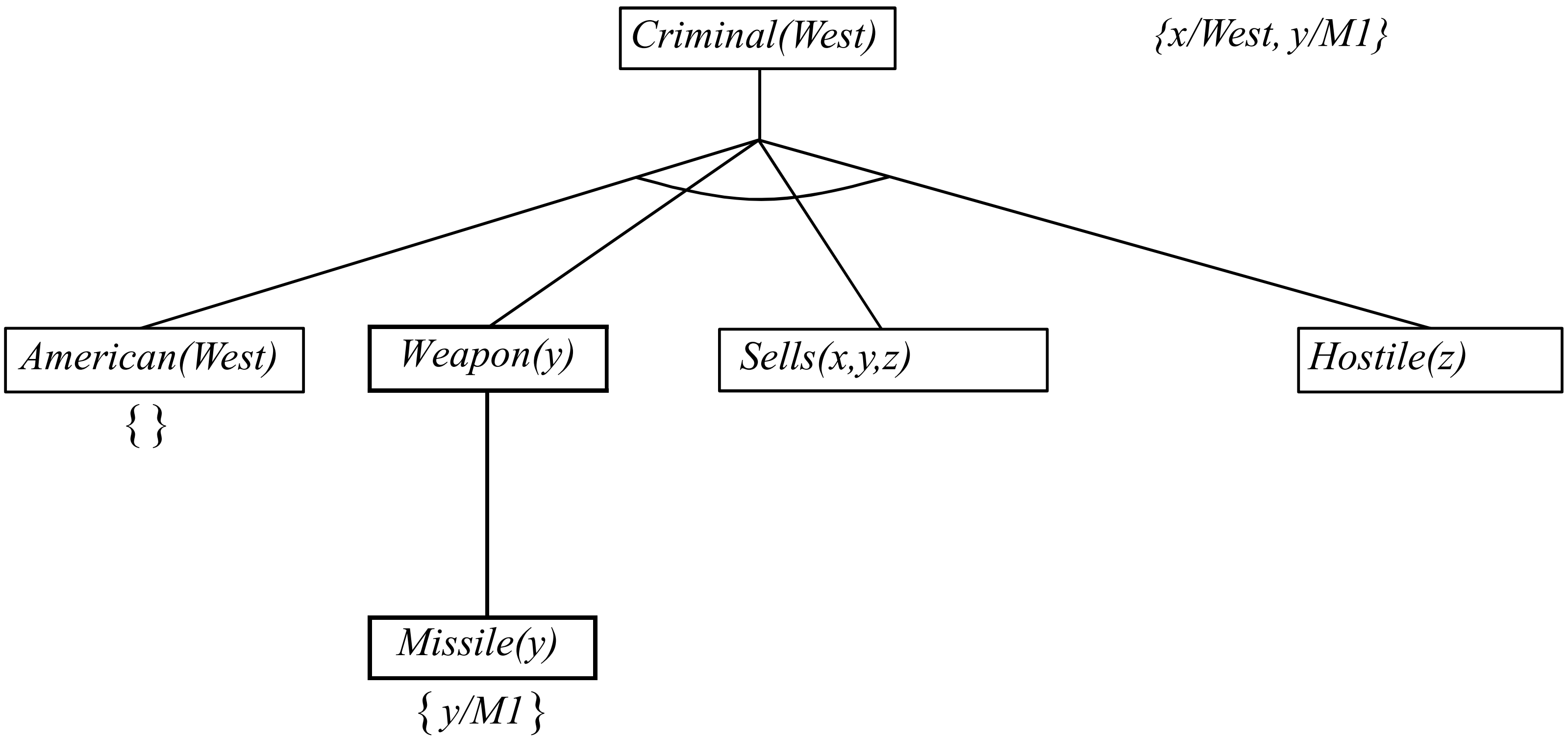
Backward chaining example



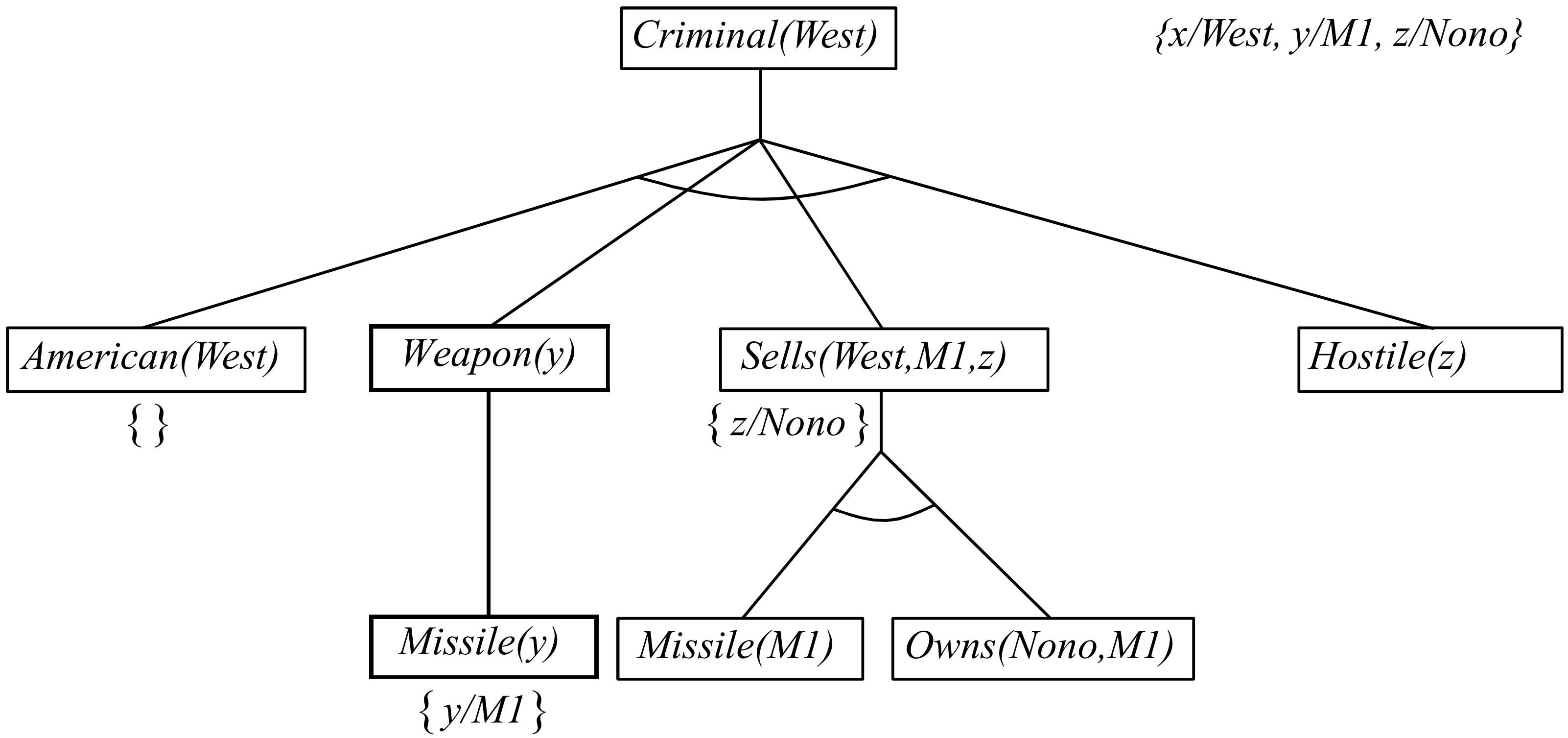
Backward chaining example



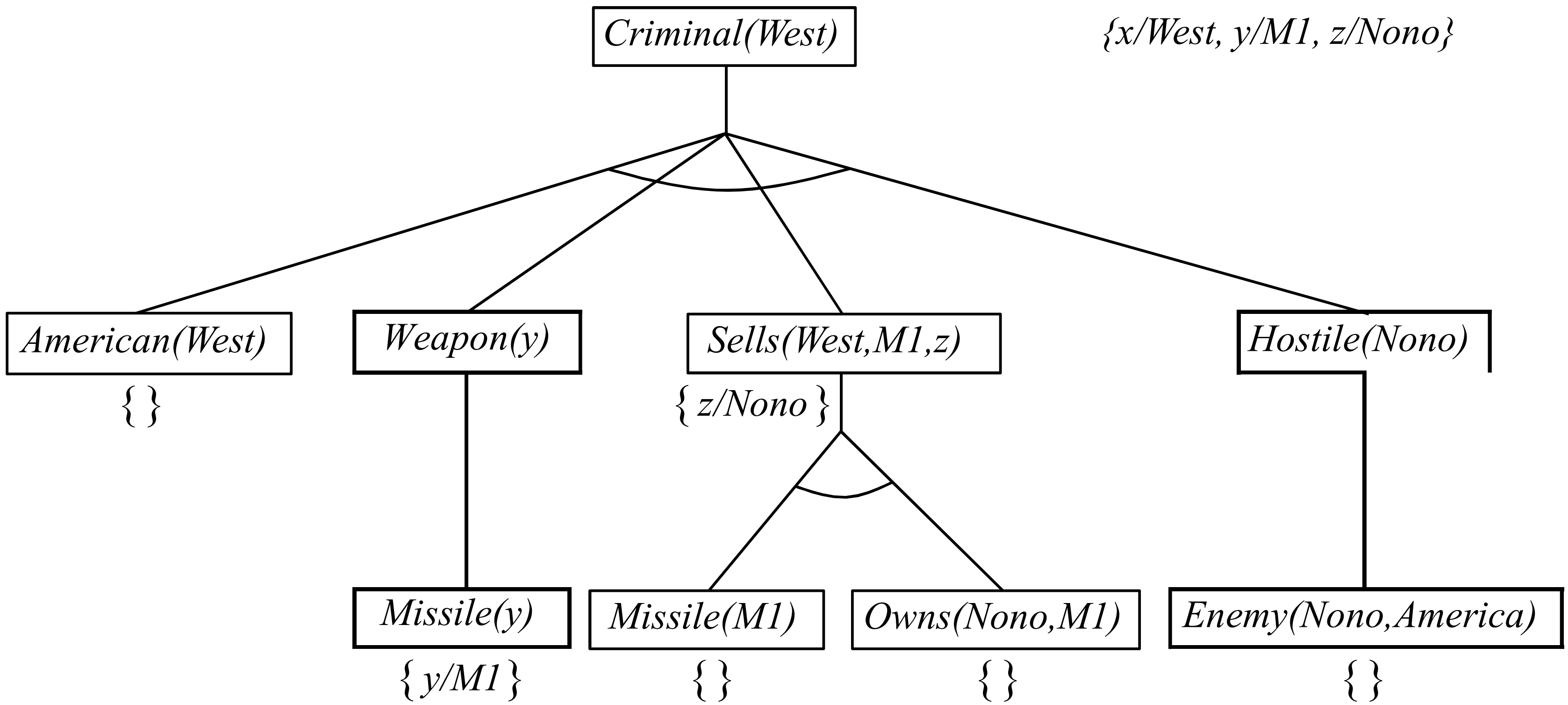
Backward chaining example



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)
Compilation techniques ⇒ approaching a billion LIPS

Program = set of clauses = head :- literal$_1$, ... literal$_n$.

```
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:

$$\frac{{}_1 \vee \cdots \vee {}_k, \qquad m_1 \vee \cdots \vee m_n}{({}_1 \vee \cdots \vee {}_{i-1} \vee {}_{i+1} \vee \cdots \vee {}_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\mathrm{Unify}({}_i, \neg m_j) = \theta$.

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

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

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

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

Conversion to CNF contd.

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:

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

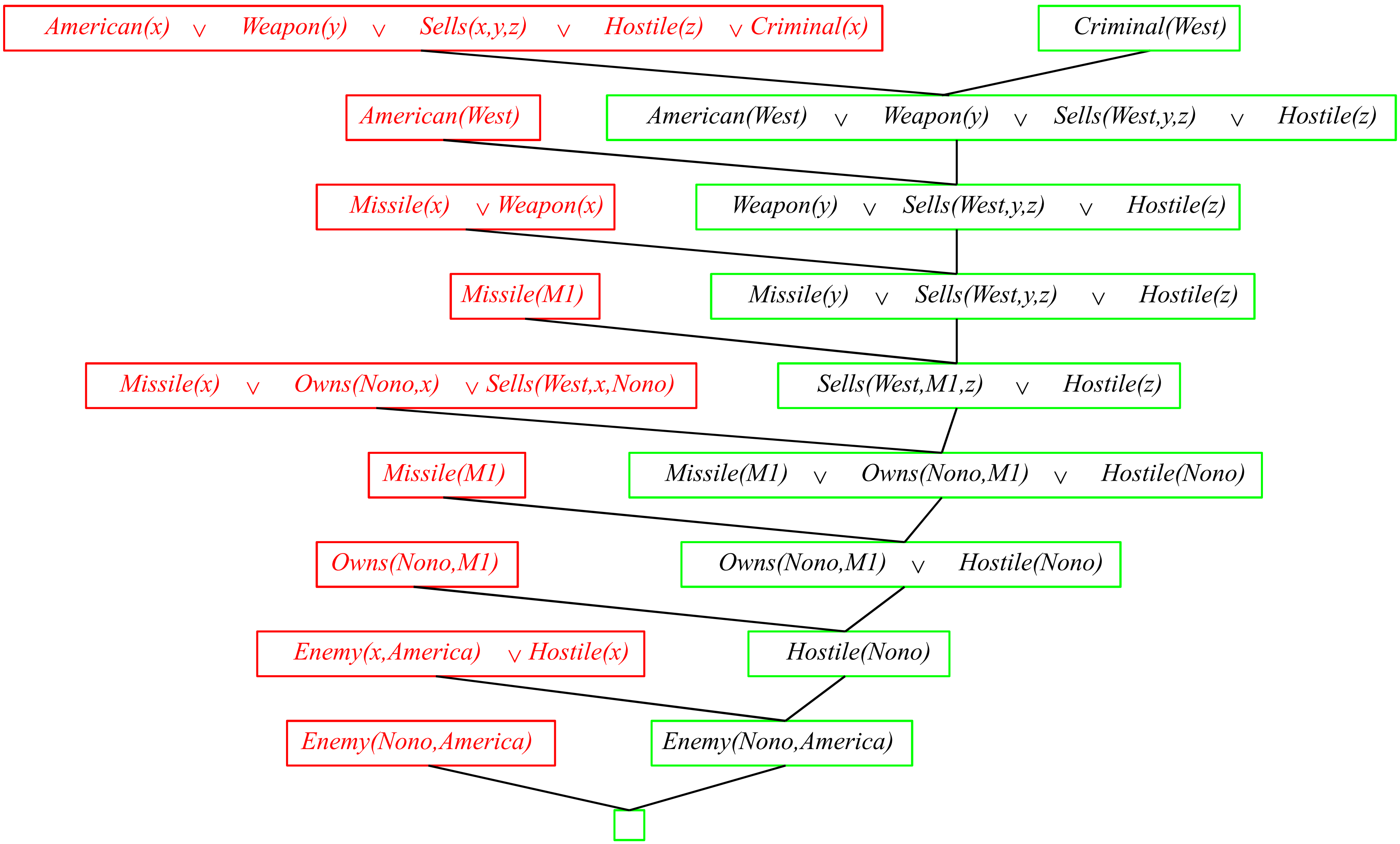
5. Drop universal quantifiers:

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

6. Distribute $\land$ over $\lor$:

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

Resolution proof: definite clauses



Gödel's Incompleteness Theorem

- There are true arithmetic sentences that cannot be proved
- For any set of true sentences of number theory, and in particular any set of basic axioms, there are other true sentences that cannot be proved from those axioms.
- We can never prove all the theorems of mathematics within any given system of axioms.

Resolution strategies

- **Unit preference:** prefers to do resolutions where one of the sentences is a single literal (unit clause)
- **Set of support:** every resolution step involve at least one element of a special set of clauses
- **Input resolution:** every resolution combines one of the KB input sentences with other sentences
- **Subsumption:** eliminates all sentences that are subsumed by KB sentences
- **Learning:** learning from experience (machine learning)

Summary

- **Unification** identify appropriate substitutions for variables eliminates the instantiation step in first-order proofs, making the process more efficient in many cases
- **Forward chaining** is used in deductive databases, where it can be combined with relational database operations. It is also used in production systems
- **Backward chaining** is used in logic programming systems, which employ sophisticated compiler technology to provide very fast inference
- **Prolog**, unlike first-order logic, uses a closed world with the unique names assumption and negation as failure.
- The generalized **resolution** inference rule provides a complete proof system for first order logic, using knowledge bases in conjunctive normal form.