Last update: November 10, 2012

Inference in first-order logic

CMSC 421: Chapter 9

Outline

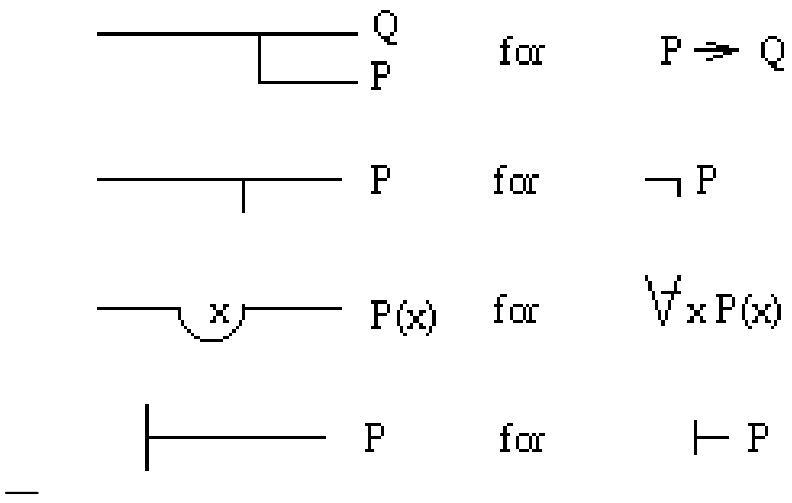
- ♢ Reducing first-order inference to propositional inference
- ♢ Unification
- ♢ Generalized Modus Ponens
- ♢ Forward and backward chaining
- ♢ Logic programming
- ♢ Resolution

A brief history of first-order logic

1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

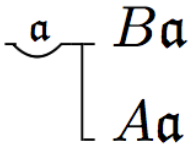
Frege's notation for FOL

♢ In Frege's notation, formulas looked like tree structures



♢ Example: $\forall x(A(x) \rightarrow B(x))$

- Frege would have written



Universal instantiation (UI)

♢ Every instantiation of a universally quantified sentence is entailed by it

♢ For every variable v and ground term g, if θ is the substitution $\{v = g\}$ then

$$\frac{\forall\, v \;\; \alpha}{\alpha\, \theta}$$

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

- $King(John) \wedge Greedy(John) \;\Rightarrow\; Evil(John)$
- $King(Richard) \wedge Greedy(Richard) \;\Rightarrow\; Evil(Richard)$
- $King(father(John)) \wedge Greedy(father(John)) \;\Rightarrow\; Evil(father(John))$

$\ldots$

Existential instantiation (EI)

♢ For any sentence α, variable v, and constant symbol k **that doesn't appear elsewhere in the knowledge base**, if $\theta = \{v = k\}$ then

$$\frac{\exists v \;\; \alpha}{\alpha\,\theta}$$

♢ E.g., from $\exists x \;\; Crown(x) \wedge OnHead(x, John)$ we can infer

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

where C_1 is a new constant symbol (doesn't already appear somewhere)

♢ In words:

- If there is a crown on John's head, then we can call the crown C_1

♢ C_1 is 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

♢ Mathematicians use these techniques informally every day

♢ Example: proofs involving limits

- Suppose $\lim_{x \to 5} f(x) = 2$. Then
  $\forall\, \epsilon > 0 \;\; \exists\, \delta > 0 \;\; \forall\, x \;\; |x - 5| < \delta \;\Rightarrow\; |f(x) - 2| < \epsilon$.
- Let ϵ be any number > 0. Then
  $\exists\, \delta > 0 \;\; \forall\, x \;\; |x - 5| < \delta \;\Rightarrow\; |f(x) - 2| < \epsilon$.
- Let $\delta_1 > 0$ be such that $\forall x \; |x - 5| < \delta, \;\; |f(x) - 2| < \epsilon$.
- Let x be any number such that $|x - 5| < \delta_1$. Then $|f(x) - 2| < \epsilon$.
- ...

Reduction to propositional inference

♢ Suppose the KB contains just the following:

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

♢ New KB: instantiate the universal sentence in **all possible** ways:

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

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

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

♢ The new KB preserves entailment of all ground sentences

- A ground sentence is entailed by new KB iff entailed by original KB

Reduction to propositional inference (continued)

♢ Every FOL KB can be propositionalized so as to preserve entailment of all ground sentences

- Propositionalize KB and query, apply resolution, return result

♢ Problem 1: propositionalization can create lots of irrelevant sentences.

- E.g., suppose a KB contains

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

♢ To prove $Evil(John)$, use propositionalization to get $Greedy(John)$

- But we also get $Greedy(Richard)$ and $Greedy(Joanna)$

♢ With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations

Reduction to propositional inference (continued)

♢ Problem 2: with function symbols, propositionalization can create infinitely many sentences!

- $Greedy(John)$
- $Greedy(father(John))$
- $Greedy(father(father(John)))$
- $\ldots$

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

♢ Basic idea: For $n = 0$ to ∞ do

- create a propositional KB by instantiating with all terms of depth $\leq n$
  - ◇ (e.g., up to n nested occurrences of $Father$)
- see if α is entailed by this KB

♢ Problem: works if α is entailed, goes forever if α is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is *semidecidable*

Unification

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

♢ We can infer $Evil(John)$ immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

- e.g., $\theta = \{x = John, y = John\}$

♢ Such a substitution is called a *unifier*

Unification

- A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical
  - α and β are *unifiable* if such a θ exists
- What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, Joanna)$	
$Knows(John, x)$	$Knows(y, mother(y))$	
$Knows(John, x)$	$Knows(x, Joanna)$	
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	
$Knows(x, x)$	$Knows(z, mother(z))$	

Unification

♢ A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical

- α and β are *unifiable* if such a θ exists

♢ What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x = Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	
$Knows(John, x)$	$Knows(y, mother(y))$	
$Knows(John, x)$	$Knows(x, Joanna)$	
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	
$Knows(x, x)$	$Knows(z, mother(z))$	

Unification

- A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical
  - α and β are *unifiable* if such a θ exists
- What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x = Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	$\{x = Joanna, y = John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	
$Knows(John, x)$	$Knows(x, Joanna)$	
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	
$Knows(x, x)$	$Knows(z, mother(z))$	

Unification

- ♢ A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical
  - α and β are *unifiable* if such a θ exists
- ♢ What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x = Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	$\{x = Joanna, y = John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	$\{y = John, x = mother(John)\}$
$Knows(John, x)$	$Knows(x, Joanna)$	
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	
$Knows(x, x)$	$Knows(z, mother(z))$	

Unification

- A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical
  - α and β are *unifiable* if such a θ exists
- What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x = Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	$\{x = Joanna, y = John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	$\{y = John, x = mother(John)\}$
$Knows(John, x)$	$Knows(x, Joanna)$	$fail$
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	
$Knows(x, x)$	$Knows(z, mother(z))$	

Unification

♢ A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical

- α and β are *unifiable* if such a θ exists

♢ What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x = Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	$\{x = Joanna, y = John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	$\{y = John, x = mother(John)\}$
$Knows(John, x)$	$Knows(x, Joanna)$	$fail$
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	$\{x_{17} = John, x = Joanna\}$
$Knows(x, x)$	$Knows(z, mother(z))$	

♢ *Standardizing apart* eliminates overlap of variables, e.g., $Knows(x_{17}, Joanna)$

Unification

♢ A *unifier* for α and β is a substitution θ such that $\alpha\theta$ and $\beta\theta$ are syntactically identical

- α and β are *unifiable* if such a θ exists

♢ What is the unifier (if there is one) for each of the following?

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x = Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	$\{x = Joanna, y = John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	$\{y = John, x = mother(John)\}$
$Knows(John, x)$	$Knows(x, Joanna)$	$fail$
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	$\{x_{17} = John, x = Joanna\}$
$Knows(x, x)$	$Knows(z, mother(z))$	$fail$

♢ *Standardizing apart* eliminates overlap of variables, e.g., $Knows(x_{17}, Joanna)$

♢ Can't unify a variable with a term that contains the variable

Unification (continued)

♢ A *most general unifier (mgu)* for α and β is a substitution θ such that

- θ is a unifier for α and β, and
- for every unifier θ' of α and β and for every expression e, $e\theta'$ is a substitution instance of $e\theta$

♢ E.g., let $\alpha = Knows(w, father(x))$ and $\beta = Knows(mother(y), y)$

- $\theta_1 = \{w = mother(father(x))), y = father(x)\}$ is an mgu
- $\theta_2 = \{w = mother(father(v))), y = father(v), x = v\}$ is an mgu
- $\theta_3 = \{w = mother(father(John)), y = father(John)\}$ is a unifier, but it is not an mgu

♢ If θ and θ' are mgus for α and β, then they are identical except for renaming of variables

Algorithm to find an mgu

♢ Compare the expressions element by element, building up a substitution along the way

- I'll give the basic idea; the book gives additional details

♢ For each pair of corresponding elements:

- Apply the substitution we've built so far
- If the two elements are the same after substituting, then continue
- Else if one of them is a variable x and the other is an expression e, and if x doesn't appear anywhere in e (the *occur check*)
  - ♢ then incorporate $x = e$ into the substitution
- Else FAIL

$$\begin{array}{ccccccccccc} Knows & (& John & , & mother & (& z & &)) \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & & \updownarrow \\ Knows & (& y & , & mother & (& sister(y) & &)) \\ & & y = John & & & & z = sister(John) & & \end{array}$$

♢ Runs in quadratic time (would be linear time if we left out the occur check)

Generalized Modus Ponens (GMP)

♢ Inference rule:

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

where

- ◇ θ is a substitution such that $p_i'\theta = p_i\theta$ for all i
- ◇ all variables are assumed to be universally quantified.

♢ Example:

$$\frac{King(John), \quad Greedy(y), \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)}{Evil(John)}$$

- ◇ $\theta = \{x = John, y = John\}$
- ◇ $q\theta = Evil(x)\theta = Evil(John)$

♢ Equivalent formulation using *definite clauses* (exactly one positive literal)

$$\frac{p_1', \quad p_2', \quad \ldots, \quad p_n', \quad (\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee q)}{q\theta}$$

Soundness of GMP

Theorem. If θ is a substitution that unifies p'_i with p_i for all i, then

$$p'_1,\ \ldots,\ p'_n,\ \ (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\ \models\ q\theta$$

Proof.

- ♢ Suppose $p'_1,\ \ldots,\ p'_n$, and $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ are true.
- ♢ Let θ be as above. Then by applying θ, we get
  - $p'_1\theta,\ \ldots,\ p'_n\theta$, and $(p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
- ♢ For all i, $p'_i\theta$ and $p_i\theta$ are the same expression, so we have
  - $p_1\theta,\ \ldots,\ p_n\theta$, and $(p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
- ♢ From ordinary Modus Ponens, we get $q\theta$

Example knowledge base

♢ The law says it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles. All of its missiles were sold to it by Colonel West, who is American.

♢ Prove that Col. West is a criminal

Example knowledge base

♢ The law says it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles. All of its missiles were sold to it by Colonel West, who is American.

♢ Prove that Col. West is a criminal

♢ Need two more inference rules:

- Missiles are weapons.
- An enemy of America is a "hostile nation"

Example knowledge base

♢ ~~The law says~~ it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles. All of its missiles were sold to it by Colonel West, who is American.

♢ Prove that Col. West is a criminal

♢ Need two more inference rules:

- Missiles are weapons.
- An enemy of America is a "hostile nation"

♢ What would be the problem with an axiom like this?

- $\forall x \;\; LawSays(x) \Rightarrow x$

Example knowledge base, continued

♢ It is a crime for an American to sell weapons to hostile nations.

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

♢ The country Nono, an enemy of America, has some missiles.

- $Enemy(Nono, America)$
- $\exists x \;\; Owns(Nono, x) \wedge Missile(x)$ correct translation of "some"?
  - ◇ $Owns(Nono, M_1)$ and $Missile(M_1)$

♢ All of its missiles were sold to it by Colonel West, who is American.

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

♢ Missiles are weapons.

- $\forall x \;\; Missile(x) \Rightarrow Weapon(x)$

♢ An enemy of America is a "hostile nation"

- $\forall x \;\; Enemy(x, America) \;\Rightarrow\; Hostile(x)$

Forward chaining algorithm

♢ Like propositional forward-chaining, with these modifications:

- To make the inferences go through, must find unifiers
- To find unifiers, must standardize the variables

```
function FOL-FC-Ask(KB, α)
    repeat until new is empty
        new ← { }
        for each rule r in KB do
            (p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Variables(r)
            for each set of statements p'1, ..., p'n in KB that unify with p1, ..., pn
                θ ← the mgu
                q' ← qθ
                if q' is not a renaming of a sentence already in KB or new then
                    add q' to new
                    if q' is unifiable with α then return the unified expression
        add new to KB
    return false
```

Forward chaining proof

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

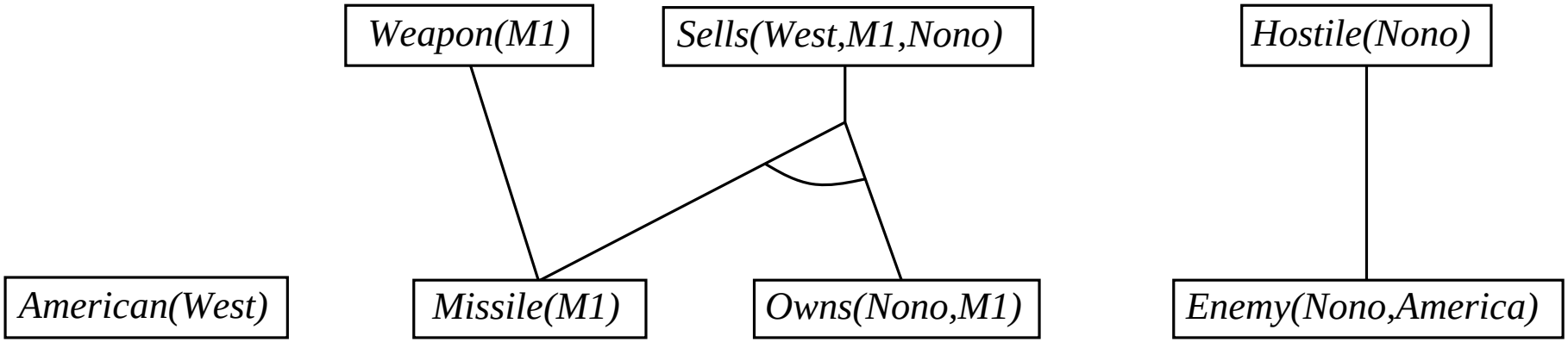
$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

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

$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x)$$

Forward chaining proof



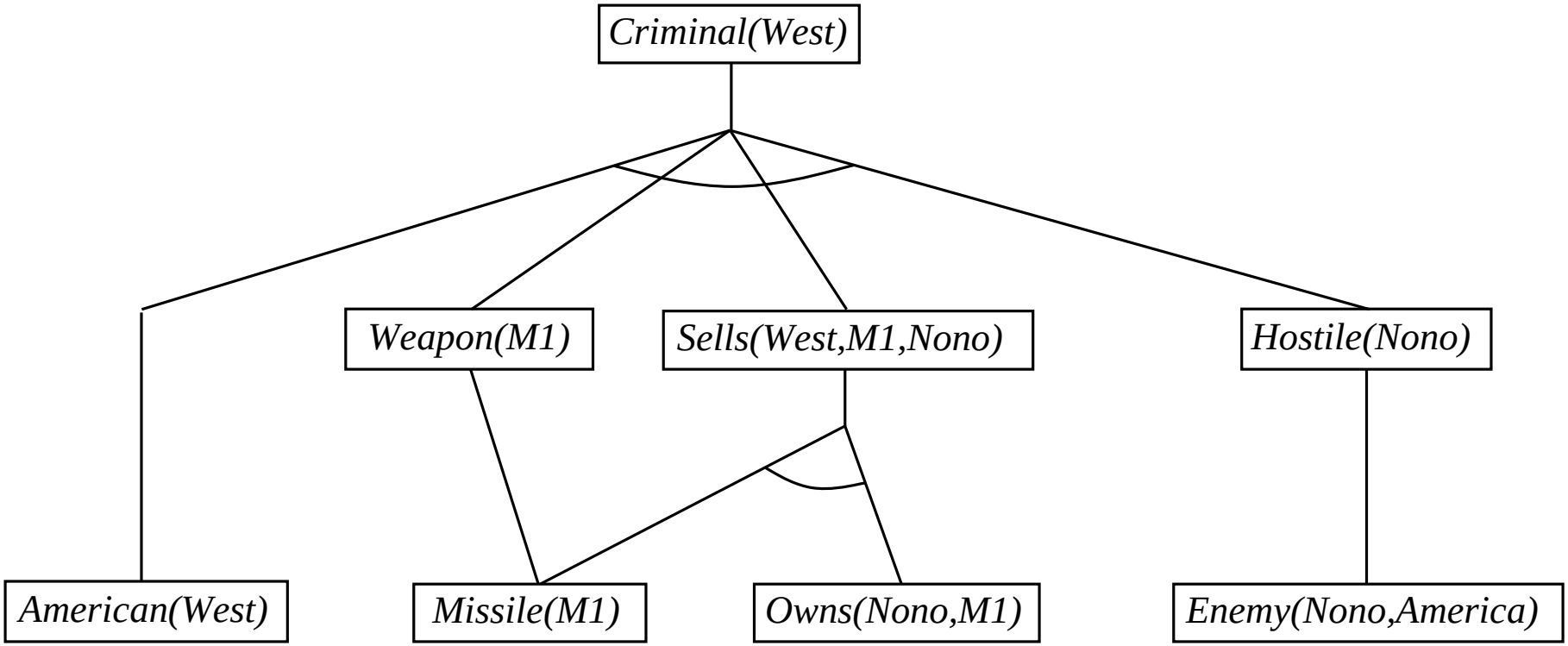
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

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

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

Forward chaining proof



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

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

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

Properties of forward chaining

♢ Sound and complete for first-order Horn clauses

- (proof similar to propositional proof)

♢ If α is not entailed, it might not terminate

- This is unavoidable because entailment is semidecidable (i.e., equivalent to the halting problem)

♢ Can guarantee termination if restrictions are satisfied

- *Datalog* = first-order Horn clauses + **no functions**
  - ◇ e.g., the Colonel West example
- FC terminates for Datalog in a polynomial number of iterations
  - ◇ at most $p \cdot n^k$ literals

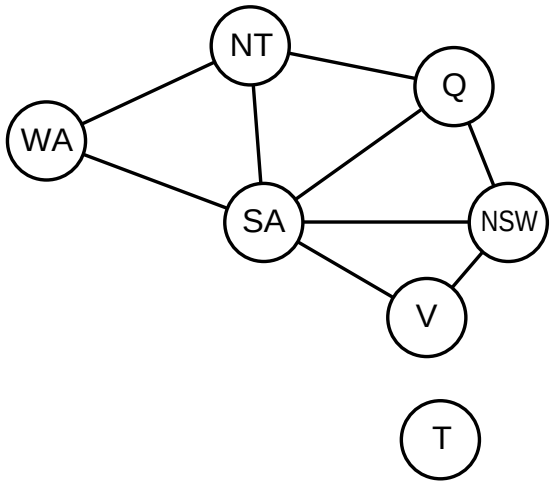
♢ Widely used (with some efficiency improvements) in *deductive databases* and *expert systems*

Efficiency of forward chaining

- Simple observation:
  - no need to look at a rule on iteration k
    unless iteration $k-1$ added at least one of its premises
- Only look at a rule if its premise contains a newly added literal
- For each such rule r, check whether all of r's premises are satisfied
  - Need to retrieve other premises
    - *Database indexing* allows $O(1)$ retrieval of known facts
    - e.g., query $Missile(x)$ retrieves $Missile(M_1)$
  - Efficiency problem:
    - Many combinations of facts may match **some** of r's premises
    - NP-hard to find one that matches **all** of r's premises
      - example on next page
  - Partial fix: store partial matches in data structures such as *rete networks*

Hard matching example

♢ Can write CSPs as datalog inference problems



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

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

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

$$Diff(wa, nt) \wedge Diff(wa, sa) \wedge Diff(nt, q) \wedge Diff(nt, sa) \wedge Diff(q, nsw) \wedge Diff(q, sa) \wedge Diff(nsw, v) \wedge Diff(nsw, sa) \wedge Diff(v, sa) \Rightarrow Colorable()$$

♢ Don't need statements like $nt = Red \vee nt = Blue \vee nt = Green$

- Why not?

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

- Need to try many combinations of variable values

♢ CSPs are NP-hard; 3SAT is a special case

Backward chaining algorithm

function FOL-BC-Ask($KB, \theta, [g_1, \ldots, g_k]$)
  inputs: KB, a knowledge base
    $g_1, \ldots, g_k$, goals: each is an atom with θ already applied to it
  $answers \leftarrow \{\}$ // *a set of substitutions, initially empty*
  if $k = 0$ (i.e., no goals) **then return** $\{\theta\}$
  for each sentence r **in** KB
    let $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ be a standardization of r
    if g_1 and q are unifiable
      $\theta' \leftarrow$ the mgu
      $answers \leftarrow answers \cup$ FOL-BC-Ask($KB, \theta\theta', [p_2\theta', \ldots, p_n\theta', g_1\theta', \ldots, g_k\theta']$)
  return *answers*

♢ What is $\theta\theta'$?

Backward chaining example

Criminal(West)

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

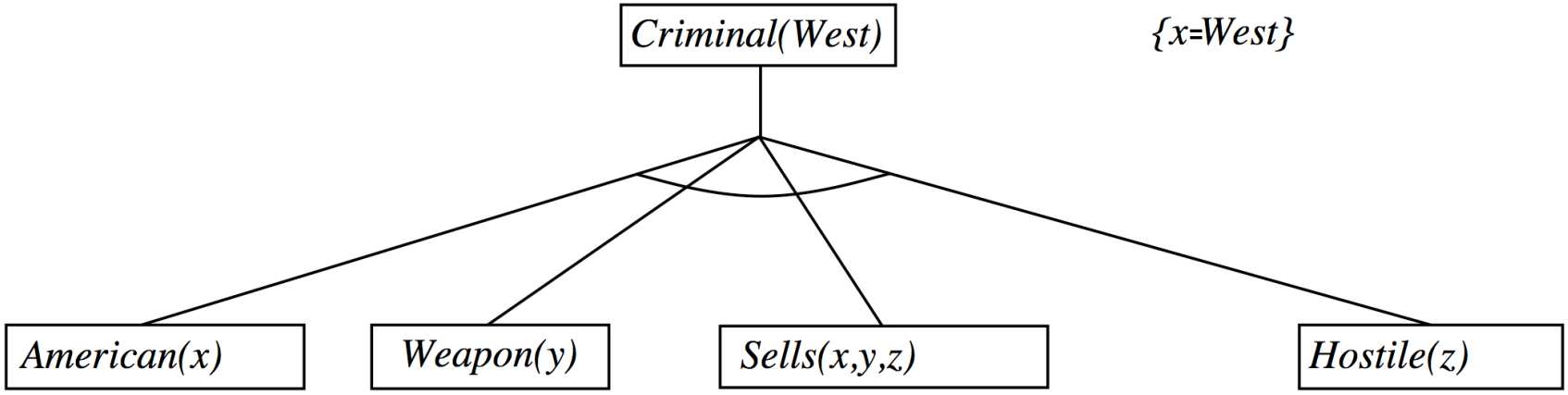
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

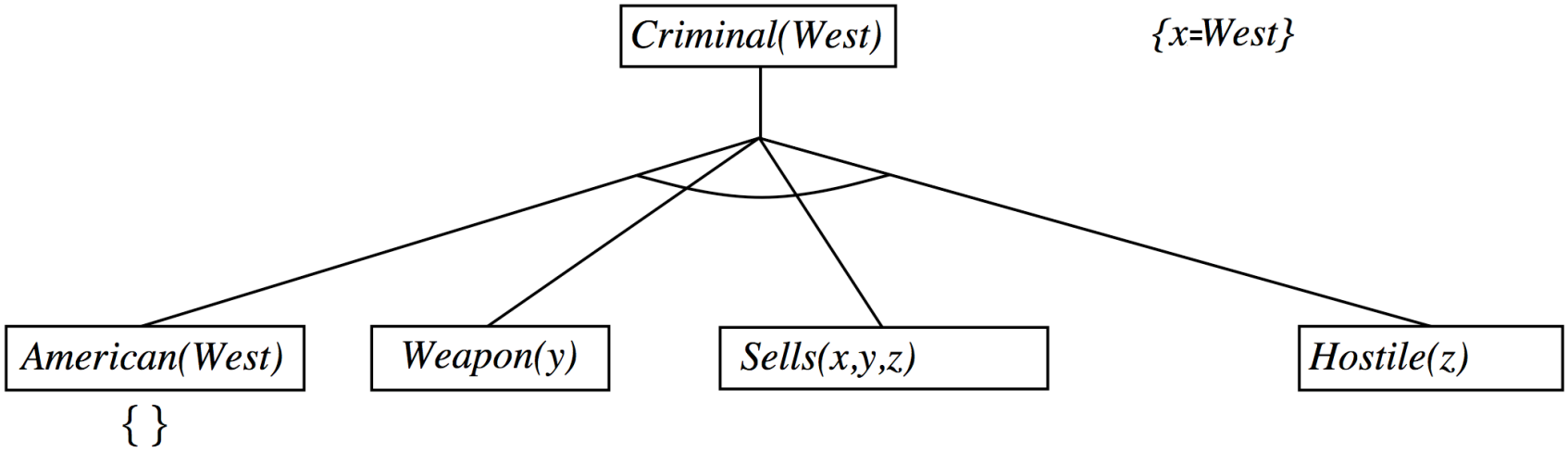
$American(West)$ $\quad$ $Enemy(Nono, America)$

Backward chaining example



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

Backward chaining example



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

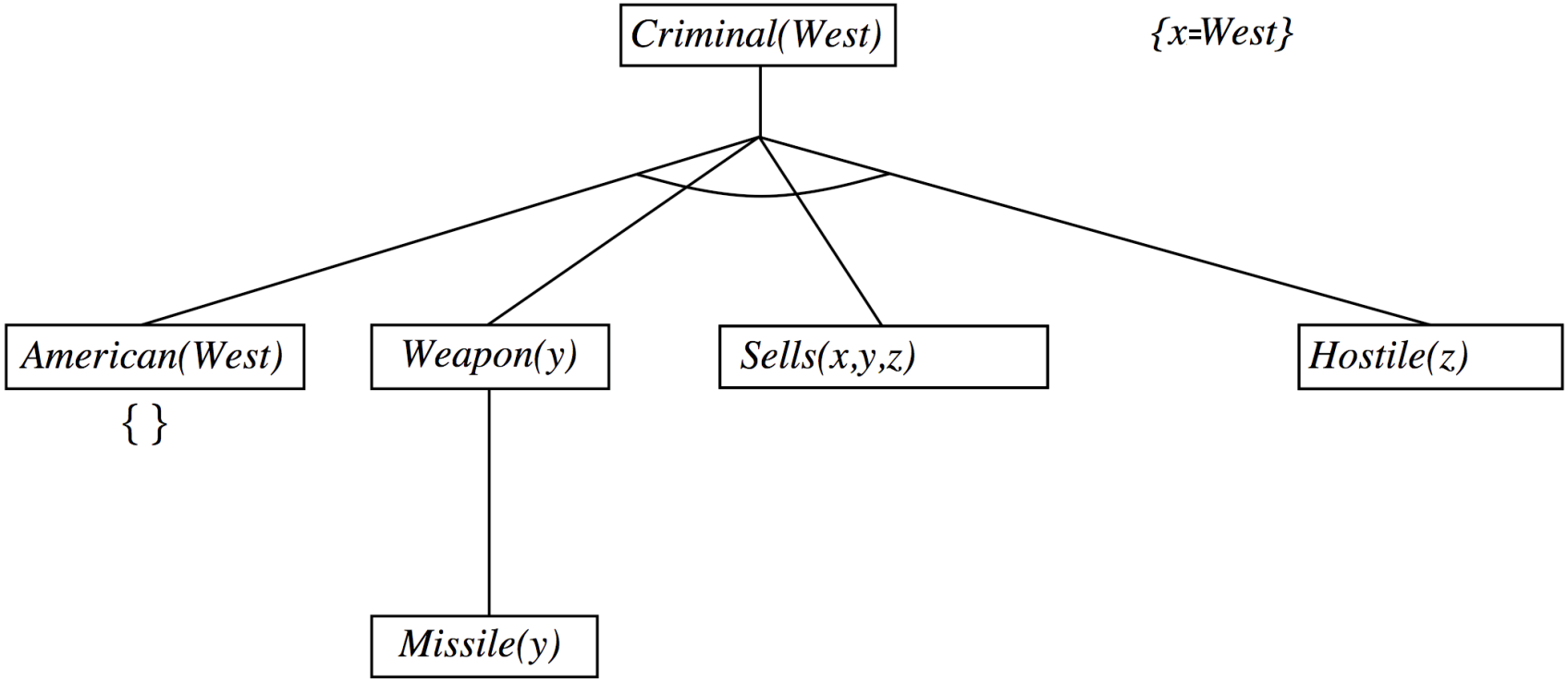
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

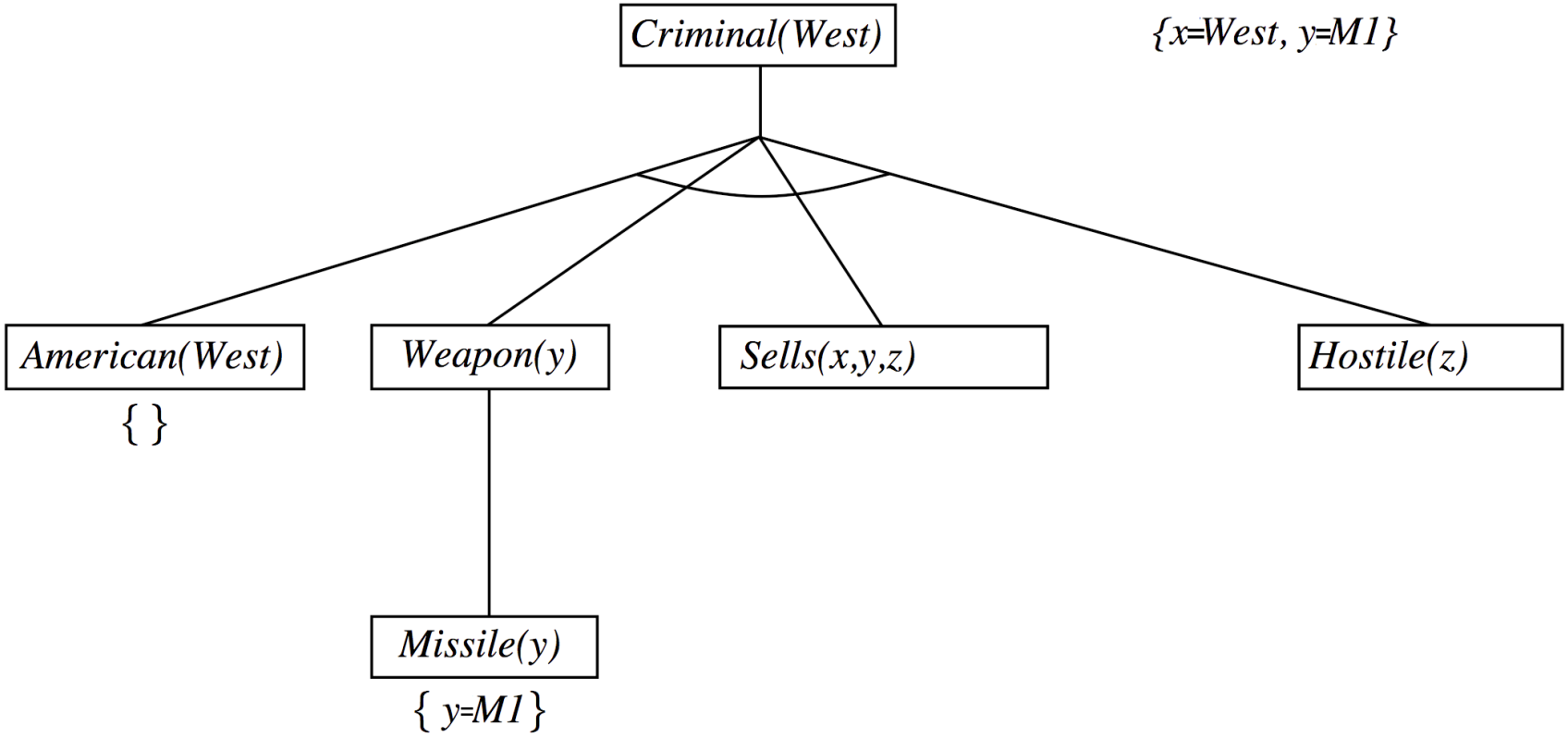
$American(West)$ $\quad$ $Enemy(Nono, America)$

Backward chaining example



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

Backward chaining example



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

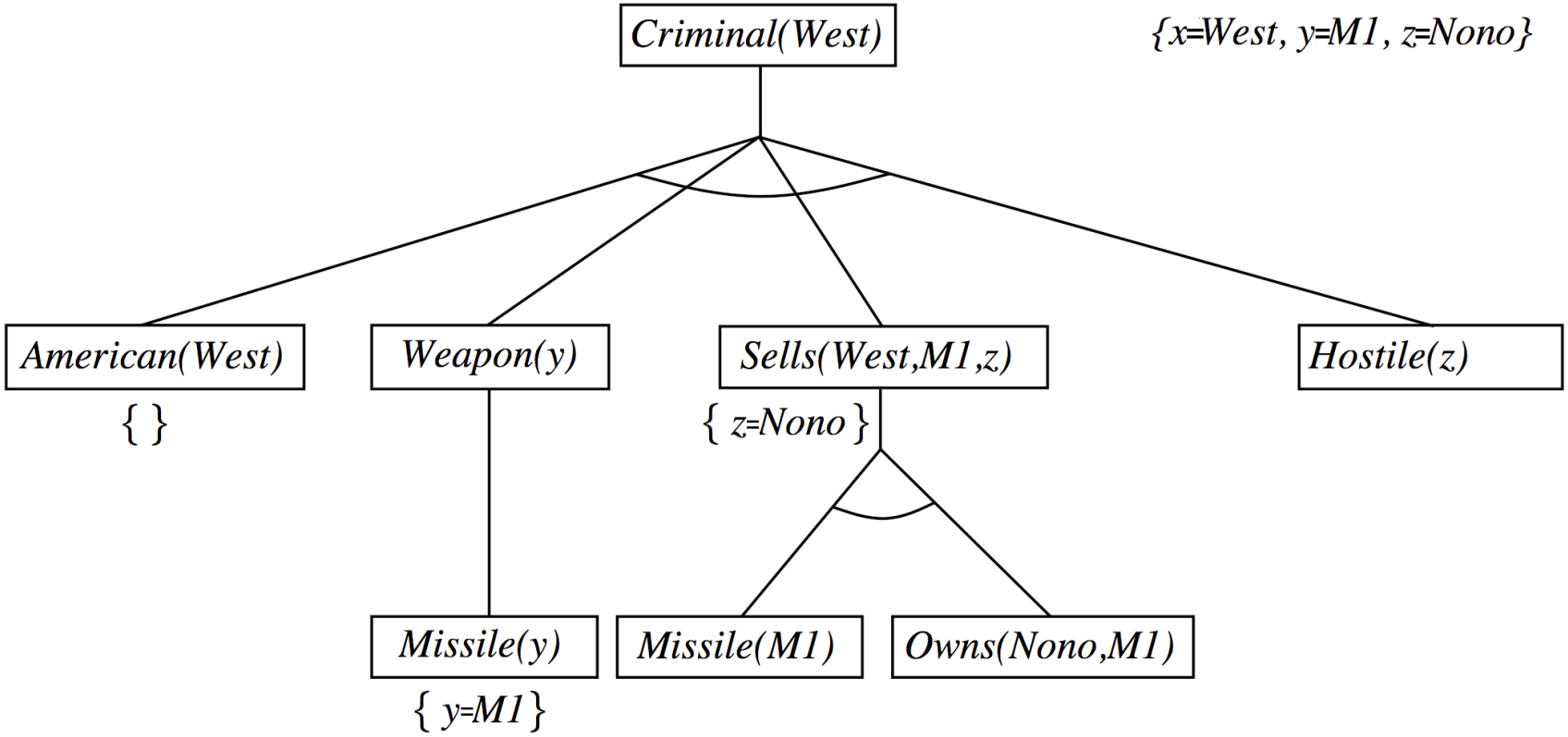
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

Backward chaining example



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

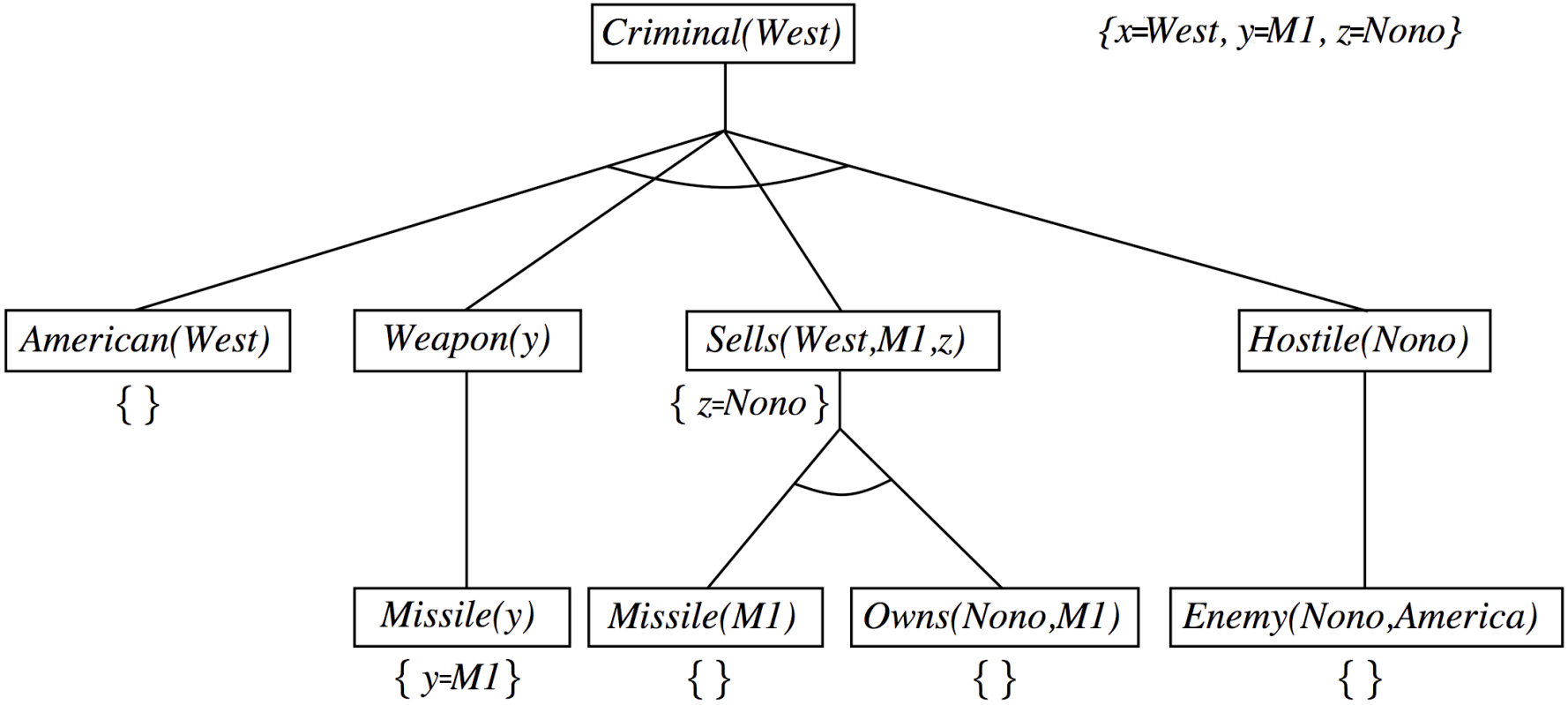
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

Backward chaining example



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

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

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

Properties of backward chaining

- ♢ Depth-first recursive proof search: space is linear in size of proof
- ♢ Incomplete due to infinite loops
  - Partial fix: check current goal against every goal on stack
  - This prevents looping here:
    - ◇ $P(x) \Rightarrow P(x)$
  - But it doesn't prevent looping here:
    - ◇ $Q(f(x)) \Rightarrow Q(x)$
- ♢ Inefficient due to repeated subgoals (both success and failure)
  - Fix using caching of previous results (extra space!)
- ♢ Widely used (without the above improvements!) for *logic programming*

- ♢ In the 1980s, an entire operating system was built around it
  - Japan's *fifth generation computer systems project*

Prolog

- Basis: backward chaining with Horn clauses
  - plus extras (e.g., built-in "predicates" for arithmetic, printing, etc.)
- Program = set of clauses having the following forms:
  - `head :- literal`$_1$`, ... literal`$_n$`.`
  - `head.`
  - Example

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- Capitalization is the opposite of what we were doing earlier
  - Capitalized words (e.g., `X`) are variables
  - lower-case words (e.g., `nono`) are constants
- Depth-first, left-to-right backward chaining
- Compilation techniques $\Rightarrow$ approaching a billion LIPS
  - Efficient unification by *open coding* (generate unification code inline)
  - Efficient retrieval of matching clauses by direct linking

Example

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

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

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
sells(west,X,nono) :- missile(X), owns(nono,X).
missile(m1).
owns(nono,m1).
weapon(X) :- missile(X)
hostile(X) :- enemy(X,america).
american(west).
enemy(nono,america).
```

♢ Goal: `:- criminal(West)`

- generates the same search tree as before
- Answer: `yes`

Depth-first search in Prolog

♢ Depth-first search from a start state `X`:

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

♢ Somewhat like the following procedure:

procedure dfs(X):
    if there is an mgu θ such that goal(Xθ) is true:
        return θ
    else
        for every mgu θ such that successor(Xθ,Sθ) is true:
            if there is an mgu θ' such that dfs(S$\theta\theta'$) is true:
                return $\theta\theta'$

♢ But there's automatic backtracking:

- Suppose we have this clause: `foo(X,Y) :- dfs(X), bar(X,Y).`
- If `dfs(X)` succeeds and `bar(X,Y)` fails, Prolog will backtrack to the **for** loop and continue it where it left off, to look for another `X`

Appending linked lists

♢ Procedure for concatenating two linked lists:

procedure append(U,V):
If U is empty, then return V
else
$Head$ = first element of U; $Tail$ = the other elements of U
Z = append($Tail$,V)
return the result of pushing $Head$ onto the front of Z

♢ Prolog notation: `[1,2,3]` is a linked list whose elements are 1, 2, and 3

- `[Head|Tail]` = list with 1st element `Head`, rest of list is `Tail`
  - ◇ e.g., if `L = [2,3,4,5]`, then `[1|L] = [1,2,3,4,5]`
- `[]` is the empty list, so `[Head|[]] = [Head]`

♢ Prolog code to concatenate two linked lists

- More general than the pseudocode above

```
append([],V,V).
append([Head|Tail],V,[Head|Z]) :- append(Tail,V,Z).
```

Appending linked lists in Prolog

♢ Same Prolog code as on the previous slide:

```
append([],V,V).
append([Head|Tail],V,[Head|Z]) :- append(Tail,V,Z).
```

♢ Given a goal `:- append(A1,A2,A3)`

if `append(A1,A2,A3)` unifies with `append([],V,V)`, return the mgu
if `append(A1,A2,A3)` unifies with `append([Head|Tail],V,[Head|Z])`:
let θ be the mgu
if there is a θ' such that `append(Tail,V,Z)`$\theta\theta'$ is true, return $\theta\theta'$
return failure

♢ Goal: `:- append([1,2],[3,4],L)`

- Answer: `L=[1,2,3,4]`

♢ Goal: `:- append(A,B,[1,2])`

- Answers:

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

Negation in Prolog

- We might want to make inferences such as $\neg Dead(x) \Rightarrow Alive(x)$
  - Not a Horn clause; the antecedent contains a negation
- Kludge: use "negation as failure"
- Example:
  - `alive(X) :- not dead(X).`
  - `:- alive(joe)`
  - Prolog tries to answer `:- dead(joe)`
    - If it succeeds, then it returns `no` for `alive(joe)`
    - Else it returns `yes` for `alive(joe)`
- Answer isn't necessarily correct:
  - will always return `yes` or `no`, even if there's no evidence either way
- Quantification problem:
  - `:- alive(Z)` ought to mean "is there someone who's alive?", but it ends up meaning "is everyone alive"?

Arithmetic in Prolog

♢ Consider the following statements:

```
Y = 4.
Z = 2.
X = (Y + Z)/2.
```

♢ We'd like to infer `X = 3.`

♢ Don't want to have to write axioms for how to evaluate every possible numeric expression

♢ Kludge: built-in binary predicate called `is`

- Written using infix notation, like this: `X is (Y + Z)/2.`
  - ◇ Not like this: `is(X,(Y + Z)/2).`
- Prolog computes the value of the expression on the right-hand side

♢ Only works if the right-hand side of the formula is completely instantiated

Resolution in FOL

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

- where θ is any substitution that unifies ℓ_i and $\neg m_j$

♢ Example: $\forall x \;\; Rich(x) \Rightarrow Unhappy(x)$

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

- with $\theta = \{x = Ken\}$

♢ To prove that $KB \models \alpha$

- convert $KB \wedge \neg\alpha$ to CNF
- use resolution try to reach a contradiction

♢ This is a complete proof procedure for FOL

- If there's a substitution θ such that $KB \models \theta\alpha$, then it will find θ
- If there's no such θ, then the procedure won't necessarily terminate

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, 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:

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

5. Drop universal quantifiers:

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

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

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

Example

♢ Original clauses:

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

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

$Owns(Nono, M_1)$

$Missile(M_1)$

$\forall x \;\; Missile(x) \Rightarrow Weapon(x)$

$\forall x \;\; Enemy(x, America) \;\Rightarrow\; Hostile(x)$

$American(West)$

$Enemy(Nono, America)$

♢ CNF:

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

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

$Owns(Nono, M_1)$

$Missile(M_1)$

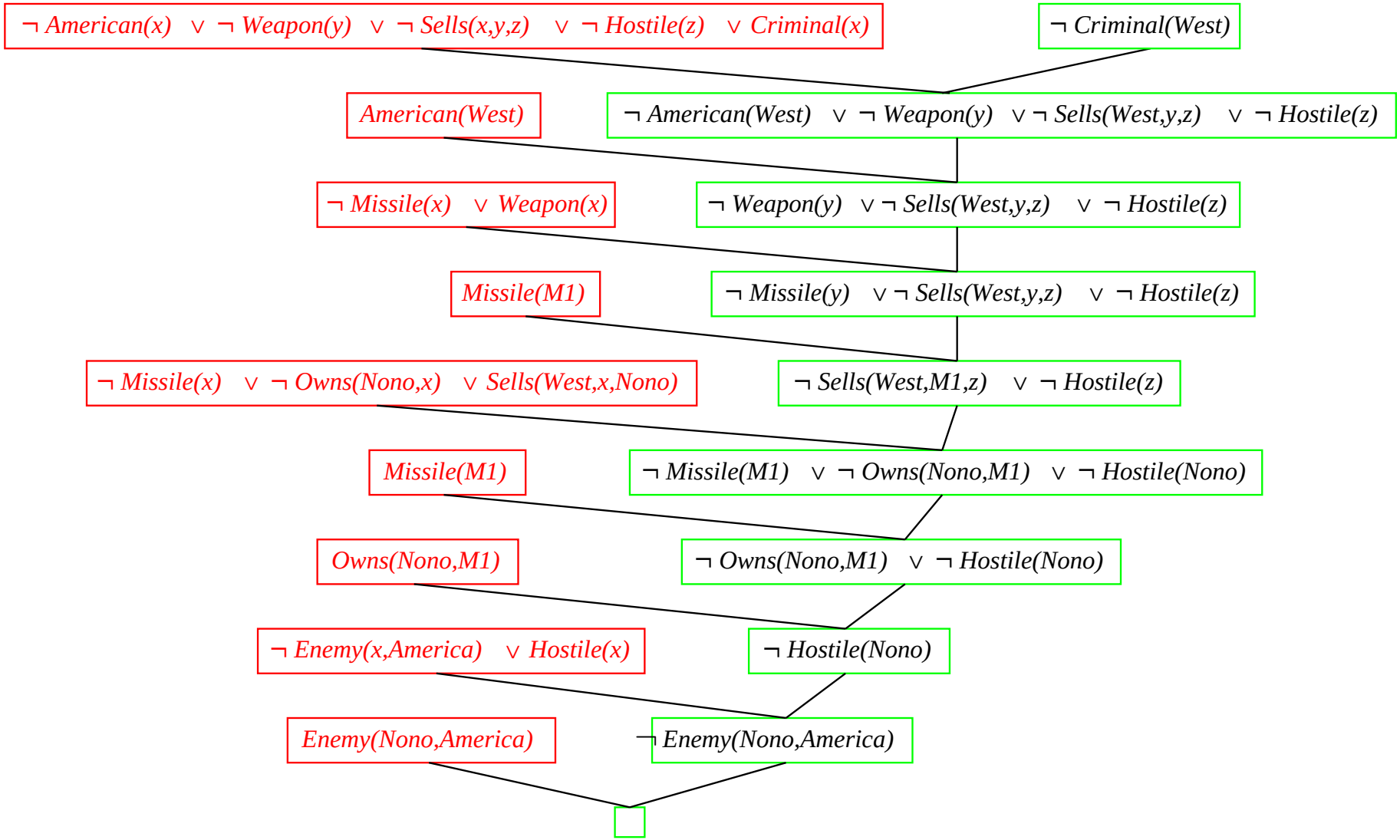
$\neg Missile(x) \vee Weapon(x)$

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

$American(West)$

$Enemy(Nono, America)$

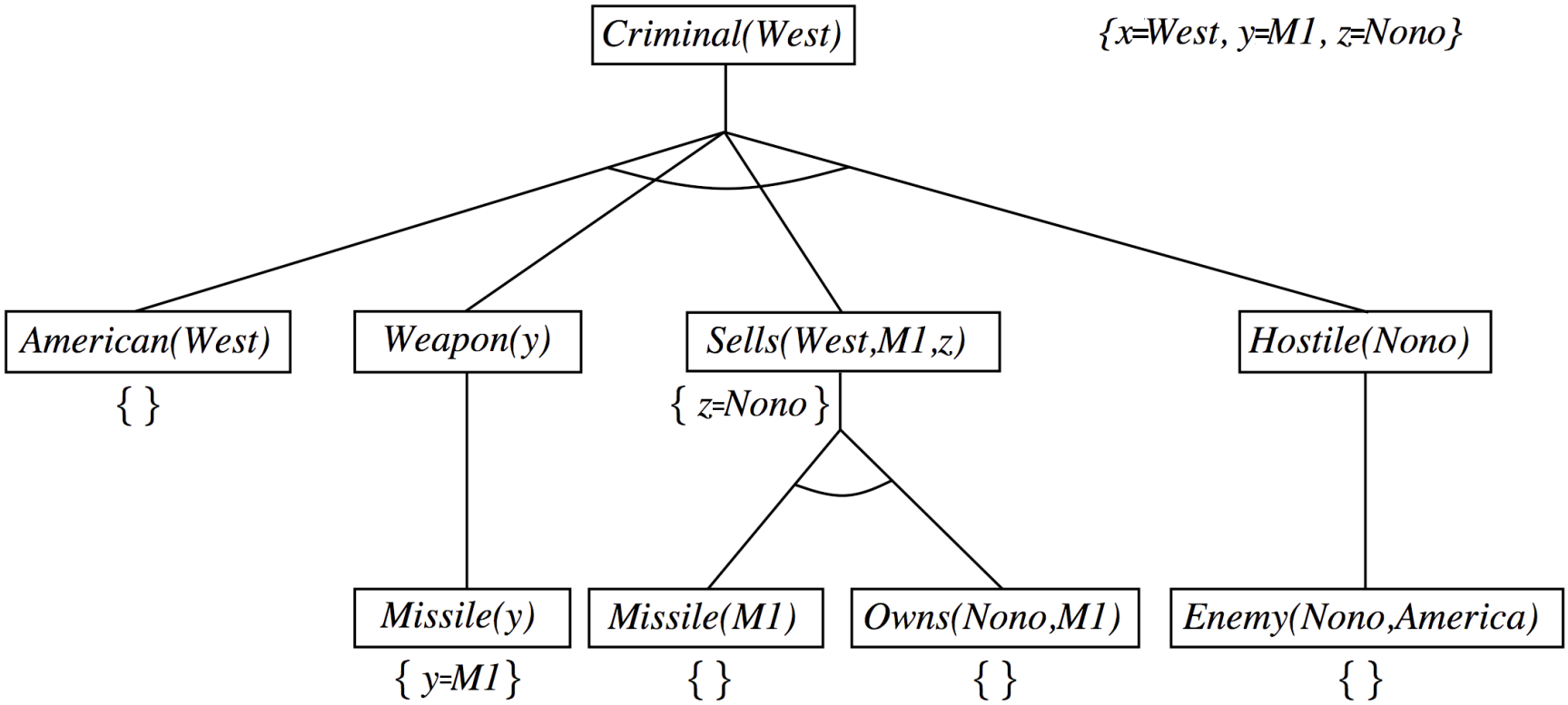
Resolution proof



♢ This figure omits all resolvents except the ones in the proof

Compare with the backward chaining example

♢ Does similar things, in a similar order



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

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

$Owns(Nono, M_1)$ $\qquad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\qquad$ $Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\qquad$ $Enemy(Nono, America)$

Homework 5

♢ Five problems: 7.12, 8.9, 9.4, 9.12, 9.23

- 10 points each, 50 points total
- Due Nov. ~~15~~ 20