

# CSCI 3230

## Fundamentals of Artificial Intelligence

Chapter 9, Chapter 10, Sect 2, 3

### INFERENCE IN FIRST-ORDER LOGIC

# Outline

- ▶ 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.	Stoics	Propositional logic, inference (may be)
322B.C.	Aristotle	“syllogisms” (inference rules), quantifier
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1849	Frege	First-Order Logic
1922	Wittgenstein	Proof by truth table
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

# Universal instantiation (UI)

for all...

$SUBST(\theta, \alpha)$  denotes the **result** of applying the substitution (or binding list)  $\theta$  to the sentence  $\alpha$ . For example:

$$SUBST(\{x/\text{Sam}, y/\text{Pam}\}, \text{Likes}(x, y)) = \text{Likes}(\text{Sam}, \text{Pam})$$

Every **instantiation** of a **universally** quantified sentence is entailed by it:

$$\frac{\forall v \quad \alpha}{SUBST(\{v/g\}, \alpha)}$$

for any sentence  $\alpha$ , any variable  $v$  and **no variable** 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 <sup>do not overload a constant name</sup> does not appear elsewhere in the knowledge base:

$$\frac{\exists v \quad \alpha}{SUBST(\{v / k\}, \alpha)}$$

- ▶ 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 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

一個公式可以被Skolem 化，就是說消除它的存在量詞並生成最初的公式的等價可滿足的公式

# Existential instantiation (EI)

- ▶ 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  
at least one case is true

# Reduction to propositional inference

Suppose the KB contains just the following:

that means all names in KB can be instantiated

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

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

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

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

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

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

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$  because x can be Richard in this case

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

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

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

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 to propositional inference

**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  $\alpha$  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  
see if  $\alpha$  is entailed by this KB

**Problem:** works if  $\alpha$  is entailed, loops if  $\alpha$  is not entailed

**Theorem:** Turing (1936), Church (1936), entailment in FOL is *semidecidable* (algorithm can say yes to every entailed sentence, cannot say no to every nonentailed sentence)



# 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!  $k$ : max # of arguments (var) in a predicate.

??So....??

# Unification

We can get the inference immediately if we can find a substitution  $\theta$  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

$\text{UNIFY}(p, q) = \theta$  where  $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$

p	q	$\theta$
Knows(John, x)	Knows(John, Jane)	$\{x/\text{Jane}\}$
Knows(John, x)	Knows(y, OJ)	$\{x/\text{OJ}, y/\text{John}\}$
Knows(John, x)	Knows(y, Mother(y))	$\{y/\text{John}, x/\text{Mother}(\text{John})\}$ <sup>recursive</sup>
Knows(John, x)	Knows(x, OJ)	<b>fail</b> <small>variable overloaded: use 1 variable to represent &gt;1 values</small>

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

# Generalized Modus Ponens (GMP)

For atomic sentences  $p_i, p_i'$ , and  $q$ , where there is a substitution  $\theta$  such that  $SUBST(\theta, p_i') = SUBST(\theta, p_i)$ , for all  $i$ :

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

$p_1'$  is *King(John)*

$p_2'$  is *Greedy(y)*

$p_1$  is *King(x)*

$p_2$  is *Greedy(x)*

$q$  is *Evil(x)*

$\theta$  is  $\{x/John, y/John\}$

$SUBST(\theta, q)$  is *Evil(John)*

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

*King(John)*

$\forall y \text{ Greedy}(y)$

*Brother(Richard, John)*

(p.9)

GMP used with KB of **definite clauses** (**exactly** one positive literal)  
All variables assumed universally quantified. (cf. **Horn clauses**)

# Soundness of GMP

Need to show that

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

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

$q\theta : SUBST(\theta, q)$  *substituted conclusion*

## 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. (陸軍上校)

Prove that Col. West is a criminal

# Example knowledge base

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

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

Nono... has some missiles, i.e.,  $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$ :

$\text{Owns}(\text{Nono}, M1)$  and  $\text{Missile}(M1)$

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

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

Missiles are weapons: (common sense)

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

An enemy of America counts as "hostile": (common sense)

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

West, who is American...

$\text{American}(\text{West})$

The country Nono, an enemy of America...

$\text{Enemy}(\text{Nono}, \text{America})$       4 rules & 4 facts

# Forward chaining algorithm

```
function FOL-FC-Ask( $KB$ ,  $\alpha$ ) returns a substitution or false  
  inputs:  $KB$ , a knowledge base, a set of first-order definite clauses  
            $\alpha$ , the query, an atomic sentence  
  local variables: new, the new sentences inferred on each iteration  
  
  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  $(p_1 \wedge \dots \wedge p_n)\theta = (p_1' \wedge \dots \wedge p_n')\theta$  //matched  
        for some  $p_1', \dots, p_n'$  in  $KB$  //facts  
           $q' \leftarrow \text{SUBST}(\theta, q)$  //a new fact (fire)  
          if  $q'$  is not a renaming of a sentence already in  $KB$  or new then do  
            add  $q'$  to new  
             $\phi \leftarrow \text{Unify}(q', \alpha)$  //query to be matched with new fact  
            if  $\phi$  is not fail then return  $\phi$   
    add new to  $KB$   
  return false
```

# Forward chaining proof

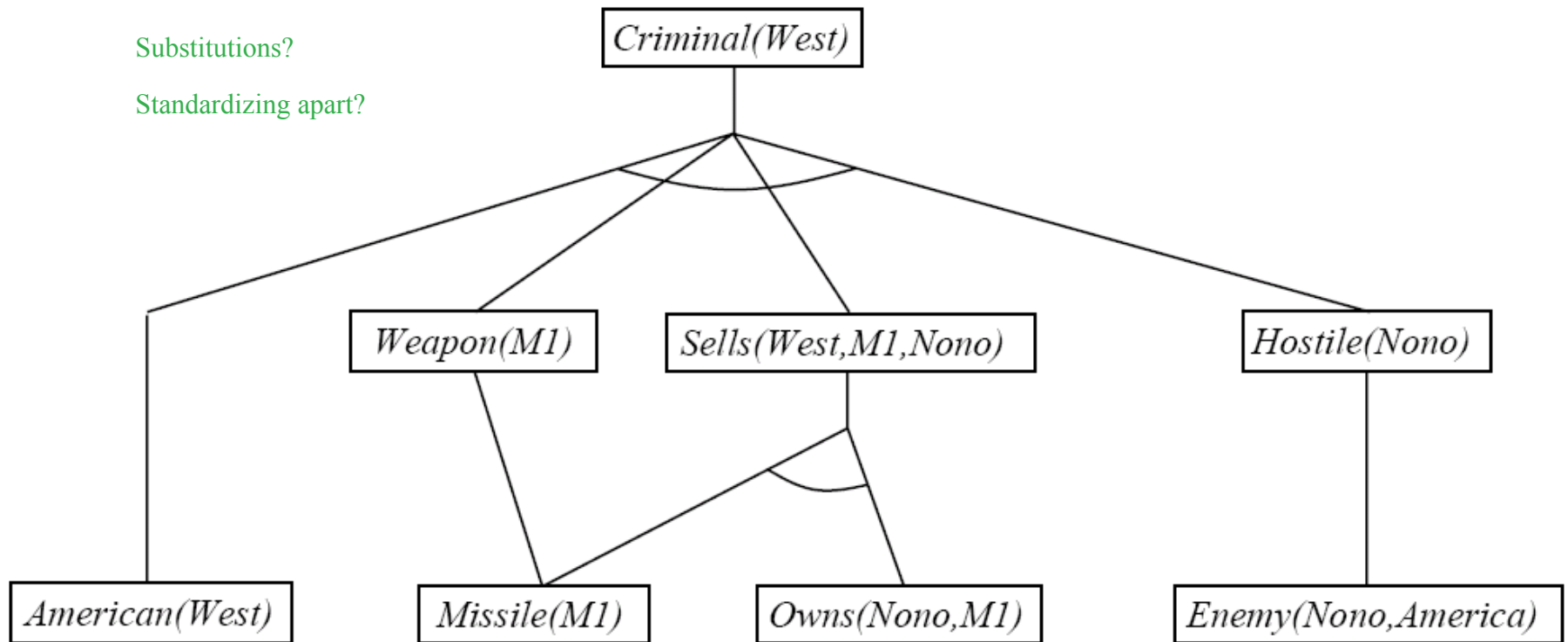
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

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

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

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

$American(West) \quad Missile(M1)$   
 $Owns(Nono, M1) \quad Enemy(Nono, America)$





# Properties of forward chaining

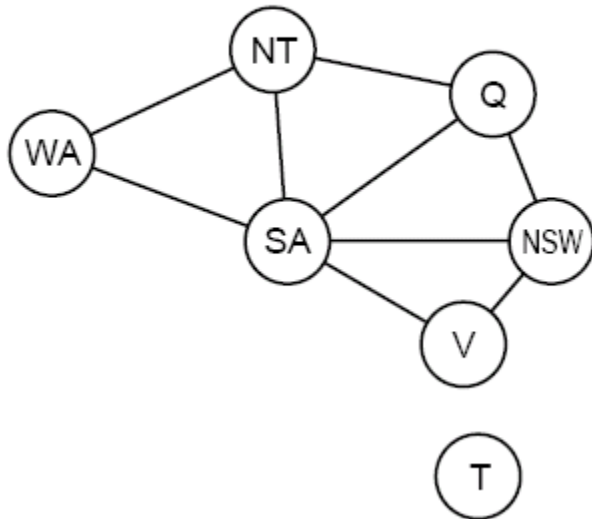
Sound and complete for first-order definite clauses  
(proof similar to propositional proof)

- ▶ **Datalog KB** = first-order definite clauses + (recursive) **no functions** (e.g. crime KB, p14)
- ▶ FC terminates for Datalog in poly iterations: at most  $p \cdot n^k$  literals  
(p k-ary predicates and n constants)
- ▶ May not terminate in general if  $\alpha$  is not entailed
- ▶ This is unavoidable: entailment with definite clauses is semidecidable (p.8)
- ▶ Datalog: a query & rule language for deductive databases (syntactically a subset of Prolog) – Database-logic

# 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
- ▶ <sup>unification</sup> Matching itself can be expensive
- ▶ Database indexing allows  $O(1)$  retrieval of known facts  
e.g., query `Missile(x)` retrieves `Missile(M)`
- ▶ Matching conjunctive premises against known facts is NP-hard
- ▶ Forward chaining is widely used in deductive databases

# Hard matching example



$$Diff(wa, nt) \wedge Diff(wa, sa) \wedge$$

$$Diff(nt, q) \wedge Diff(nt, sa) \wedge$$

$$Diff(q, nsw) \wedge Diff(q, sa) \wedge$$

$$Diff(nsw, v) \wedge Diff(nsw, sa) \wedge$$

$$Diff(v, sa) \Rightarrow Colorable()$$

$$Diff(\text{Red}, \text{Blue}) \quad Diff(\text{Red}, \text{Green})$$

$$Diff(\text{Green}, \text{Red}) \quad Diff(\text{Green}, \text{Blue})$$

$$Diff(\text{Blue}, \text{Red}) \quad Diff(\text{Blue}, \text{Green})$$

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

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

3-satisfiability problems

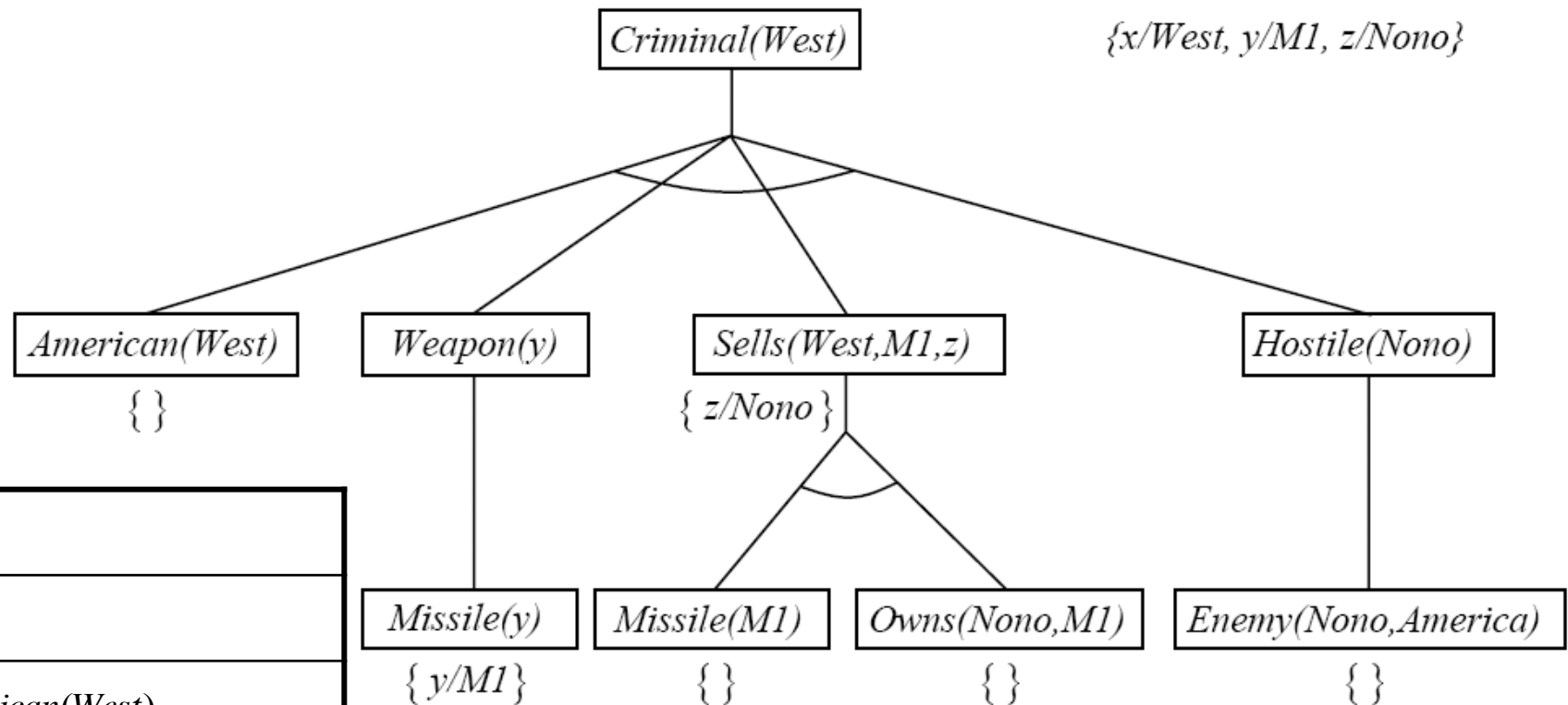
e.g. 3-coloring

# Backward chaining algorithm

```
function FOL-BC-Ask ( $KB$ ,  $goals$ ,  $\theta$ ) returns a set of substitutions  
  inputs:  $KB$ , a knowledge base  
            $goals$ , list of conjuncts forming a query  
            $\theta$ , the current substitution, initially the empty substitution  $\{\}$   
  local variable:  $ans$ , a set of substitutions, initially empty  
  if  $goals$  is empty then return  $\{\theta\}$   
   $q' \leftarrow \text{SUBST}(\theta, \text{First}(goals))$  //pop from goal stack  
  for each  $r$  in  $KB$  where  $\text{Standard-Apart}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$   
    and  $\theta' \leftarrow \text{Unify}(q, q')$  succeeds //(sub)-goal matches then-part or fact  
     $ans \leftarrow \text{FOL-BC-Ask}(KB, [p_1, \dots, p_n | \text{Rest}(goals)], \text{Compose}(\theta', \theta)) \cup ans$   
    //Depth first recursive call      push  
  return  $ans$ 
```

**Composition** of substitutions, apply each substitution in turn:  
 $\text{SUBST}(\text{Compose}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$

# Backward chaining example



$American(West).$

~~$Missile(M1).$~~

~~$Sells(West, M1).$~~

~~$Hostile(Nono, America).$~~

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

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

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

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

$American(West)$

$Missile(M1)$

$Owns(Nono, M1)$

$Enemy(Nono, America)$

# 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 sub-goals** (both success and failure)

⇒ fix using caching of previous results (extra space!)

Widely used (without improvements!) for **logic programming**

# Logic programming

Soundbite: 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 instance as facts
- 6 Ask queries
- 7 Find false facts

## Ordinary programming

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

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

# Prolog systems

Basis: backward chaining with Horn clauses + Control  
Widely used in Europe, Japan (basis of 5th Generation project)  
Compilation techniques  $\Rightarrow$  60 million LIPS

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

Depth-first, left-to-right backward chaining.

Built-in predicates for arithmetic etc., e.g., X is Y\*Z+3  
Does **not** solve: equation "5 is X+Y" fails

Closed-world assumption ("negation as failure") database is complete | if  
a fact is not in the database, then it is not true

e.g. given alive(X) :- not dead (X).  
alive(Joe) succeeds if dead(Joe) fails



# Prolog examples

Appending two lists to produce a third:

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

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

( `|` : adjoins)

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

(What 2 lists can be appended to give [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,

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

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

# Conversion to CNF (Conjunctive Normal Form)

Everyone who loves all animal 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 biconditional and implications  $(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$

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

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

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

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

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

3. Standardize variables: each quantifier should use a difference one

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

# Conversion to CNF

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

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

5. Drop universal quantifiers:  
 $[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$

6. Distribute  $\wedge$  over  $\vee$   
 $[\text{Animal}(F(x)) \vee \text{Loves}(G(x), x)]$   
 $\wedge [\neg \text{Loves}(x, F(x)) \vee \text{Loves}(G(x), x)]$

# Resolution proof: definite clauses

