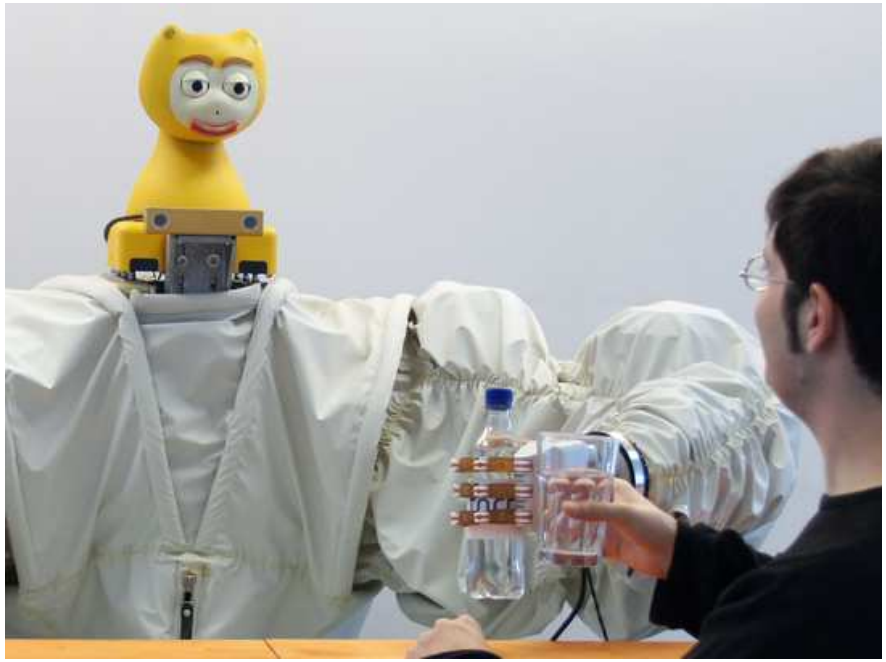
# Vorlesung

# Grundlagen der Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055



Wintersemester 2012/13 30.11.2012

# Chapter 8/9 (3rd ed.)

# Frist-Order Logic and Inference

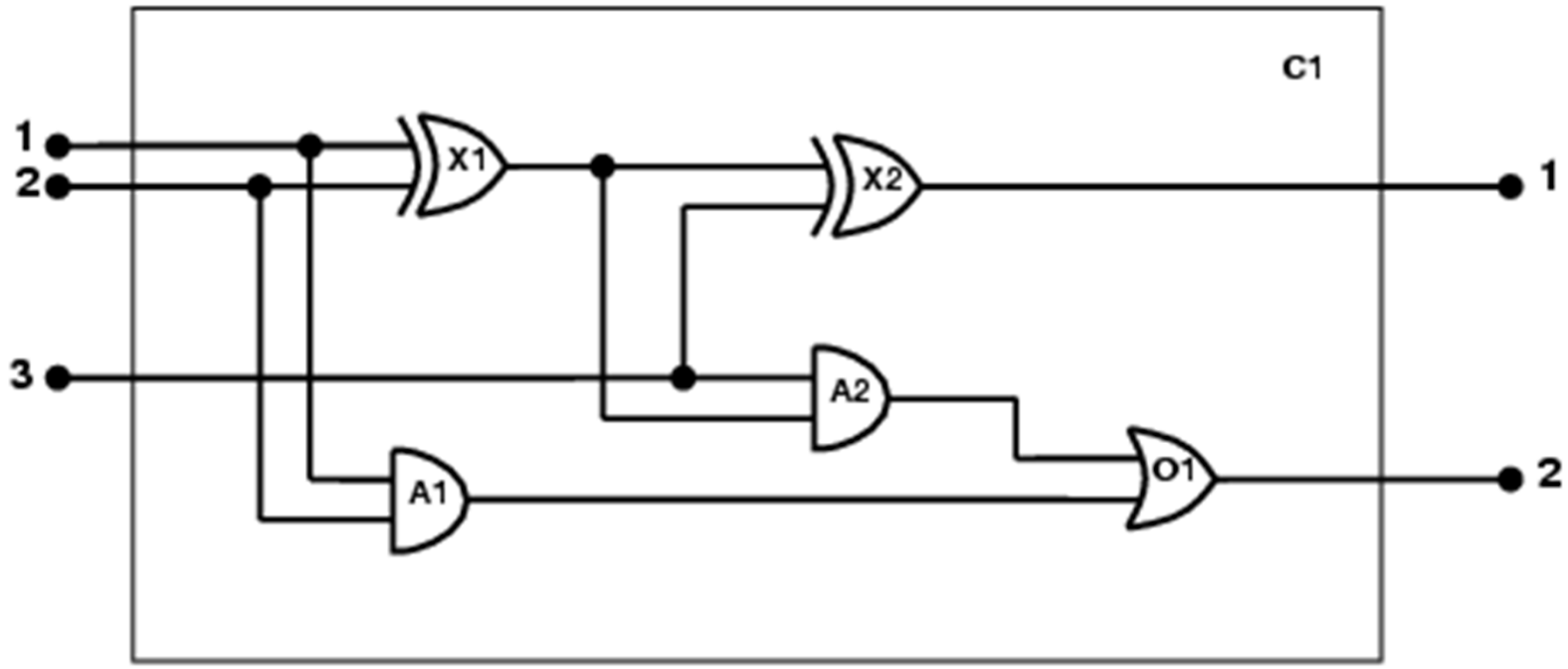
# Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base



# The electronic circuits domain

One-bit full adder



# The electronic circuits domain

1. Identify the task
   - Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge
   - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary
   - Type(X1) = XOR (or Type(X1, XOR) or XOR(X1))



# The electronic circuits domain

4. Encode general knowledge of the domain
   Assume t, $t_1$,$t_2$ are terminals, i.e. Terminal ($t_1$) …

   – $\forall t_1,t_2$ Connected($t_1$, $t_2$) $\Rightarrow$ Signal($t_1$) = Signal($t_2$)
   – $\forall t$ Signal(t) = 1 $\vee$ Signal(t) = 0

   – 1 ≠ 0

   – $\forall t_1,t_2$ Connected($t_1$, $t_2$) $\Rightarrow$ Connected($t_2$, $t_1$)

   – $\forall g$ Type(g) = OR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ $\exists n$ Signal(In(n,g)) = 1
   – $\forall g$ Type(g) = AND $\Rightarrow$ Signal(Out(1,g)) = 0 $\Leftrightarrow$ $\exists n$ Signal(In(n,g)) = 0

   – $\forall g$ Type(g) = XOR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$
     Signal(In(1,g)) ≠ Signal(In(2,g))

   – $\forall g$ Type(g) = NOT $\Rightarrow$ Signal(Out(1,g)) ≠ Signal(In(1,g))



# The electronic circuits domain

5. Encode the specific problem instance

$Type(X_1) = XOR$
$Type(A_1) = AND$
$Type(O_1) = OR$

$Type(X_2) = XOR$
$Type(A_2) = AND$

$Connected(Out(1,X_1),In(1,X_2))$
$Connected(Out(1,X_1),In(2,A_2))$
$Connected(Out(1,A_2),In(1,O_1))$
$Connected(Out(1,A_1),In(2,O_1))$
$Connected(Out(1,X_2),Out(1,C_1))$
$Connected(Out(1,O_1),Out(2,C_1))$

$Connected(In(1,C_1),In(1,X_1))$
$Connected(In(1,C_1),In(1,A_1))$
$Connected(In(2,C_1),In(2,X_1))$
$Connected(In(2,C_1),In(2,A_1))$
$Connected(In(3,C_1),In(2,X_2))$
$Connected(In(3,C_1),In(1,A_2))$



# The electronic circuits domain

6. Pose queries to the inference procedure

Which input values lead to “sum bit of $C_1$= 0 and carry bit of $C_1$ = 1” ?

$\exists i_1,i_2,i_3$ Signal(In(1,$C_1$)) = $i_1$ $\wedge$ Signal(In(2,$C_1$)) = $i_2$
$\wedge$ Signal(In(3,$C_1$)) = $i_3$
$\wedge$ Signal(Out(1,$C_1$)) = 0 $\wedge$ Signal(Out(2,$C_1$)) = 1

What are the possible sets of values of all the terminals for the adder circuit?

$\exists i_1,i_2,i_3,o_1,o_2$ Signal(In(1,$C_1$)) = $i_1$ $\wedge$ Signal(In(2,$C_1$)) = $i_2$ $\wedge$ Signal(In(3,$C_1$)) = $i_3$ $\wedge$ Signal(Out(1,$C_1$)) = $o_1$ $\wedge$ Signal(Out(2,$C_1$)) = $o_2$



# The electronic circuits domain

7. Debug the knowledge base, example XOR

$\exists i_1, i_2, o$ Signal(In(1,$C_1$)) = $i_1 \wedge$ Signal(In(2,$C_1$)) = $i_2$
$\wedge$ Signal(Out(1,$C_1$)) = $X_1$

Consider definition of XOR

Signal(Out(1, $X_1$) = 1 $\Leftrightarrow$ Signal(Out(1, $X_1$) $\neq$ Signal(Out(2, $X_1$)

Example Input is 0 and 1:

Signal(Out(1, $X_1$) = 1 $\Leftrightarrow$ 1 $\neq$ 0

May have omitted assertions like 1 $\neq$ 0, then the statement
Signal(Out(1, $X_1$) = 1 cannot be inferred



# Inference in First-Order Logic (chap. 9)

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- 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., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

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



# 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., $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields:

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

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



# 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 contd.

- Every FOL KB can be propositionalized so as to preserve entailment
- (A ground sentence is entailed by new KB iff entailed by original KB)
- 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*)))



# Reduction contd.

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

Idea: For $n = 0$ to $\infty$ 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 for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)



# Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

- Example:

  $\forall x\ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  $\text{King}(\text{John})$
  $\forall y\ \text{Greedy}(y)$
  $\text{Brother}(\text{Richard},\text{John})$

- it seems obvious that *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.



# 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)*

  θ = {x/John,y/John} works: Subst(θ, King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x), Greedy(y))

- Unify(α,β) = θ if Subst(θ, α) = Subst(θ, β)

| p | q | θ |
|---|---|---|
| Knows(John,x) | Knows(John,Jane) | {x/Jane}} |
| Knows(John,x) | Knows(y, Bill) | {x/Bill,y/John}} |
| Knows(John,x) | Knows(y,Mother(y)) | {y/John,x/Mother(John)}} |
| Knows(John,x) | Knows(x,Elisabeth) | {fail} |

- Standardizing apart eliminates overlap of variables, e.g., Knows($z_{17}$,Elisabeth)



# Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,

  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

- The first unifier is more general than the second, because there are fewer constraints on the variables.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.

  MGU = { y/John, x/z }



# The unification algorithm

**function** UNIFY($x, y, \theta$) **returns** a substitution to make $x$ and $y$ identical
  **inputs**: $x$, a variable, constant, list, or compound
    $y$, a variable, constant, list, or compound
    $\theta$, the substitution built up so far

  **if** $\theta$ = failure **then return** failure
  **else if** $x = y$ **then return** $\theta$
  **else if** VARIABLE?($x$) **then return** UNIFY-VAR($x, y, \theta$)
  **else if** VARIABLE?($y$) **then return** UNIFY-VAR($y, x, \theta$)
  **else if** COMPOUND?($x$) **and** COMPOUND?($y$) **then**
    **return** UNIFY(ARGS[$x$], ARGS[$y$], UNIFY(OP[$x$], OP[$y$], $\theta$))
  **else if** LIST?($x$) **and** LIST?($y$) **then**
    **return** UNIFY(REST[$x$], REST[$y$], UNIFY(FIRST[$x$], FIRST[$y$], $\theta$))
  **else return** failure



# The unification algorithm

**function** UNIFY-VAR($var, x, \theta$) **returns** a substitution

**inputs**: $var$, a variable

$x$, any expression

$\theta$, the substitution built up so far

**if** $\{var/val\} \in \theta$ **then return** UNIFY($val, x, \theta$)

**else if** $\{x/val\} \in \theta$ **then return** UNIFY($var, val, \theta$)

**else if** OCCUR-CHECK?($var, x$) **then return** failure

**else return** add $\{var/x\}$ to $\theta$



# 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 $Subst(\theta, p_i') = Subst(\theta, p_i)$ 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 θ 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 $Subst(\theta, p_i') = Subst(\theta, p_i)$ for all *i*

- Lemma:
  For any sentence *p*, we have $p \models Subst(\theta, p)$ by UI

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

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

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

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

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 …

$Enemy(Nono,America)$



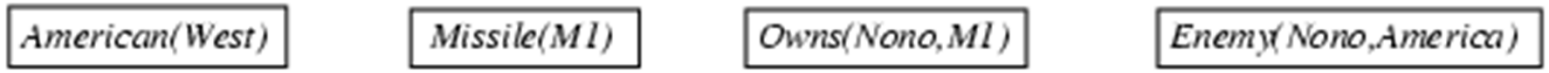
# Forward chaining algorithm

**function** FOL-FC-ASK($KB, \alpha$) **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** $\theta$ 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($\theta, q$)
      **if** $q'$ is not a renaming of a sentence already in $KB$ or *new* **then do**
        add $q'$ to *new*
        $\phi \leftarrow$ UNIFY($q', \alpha$)
        **if** $\phi$ is not *fail* **then return** $\phi$
  add *new* to $KB$
**return** *false*

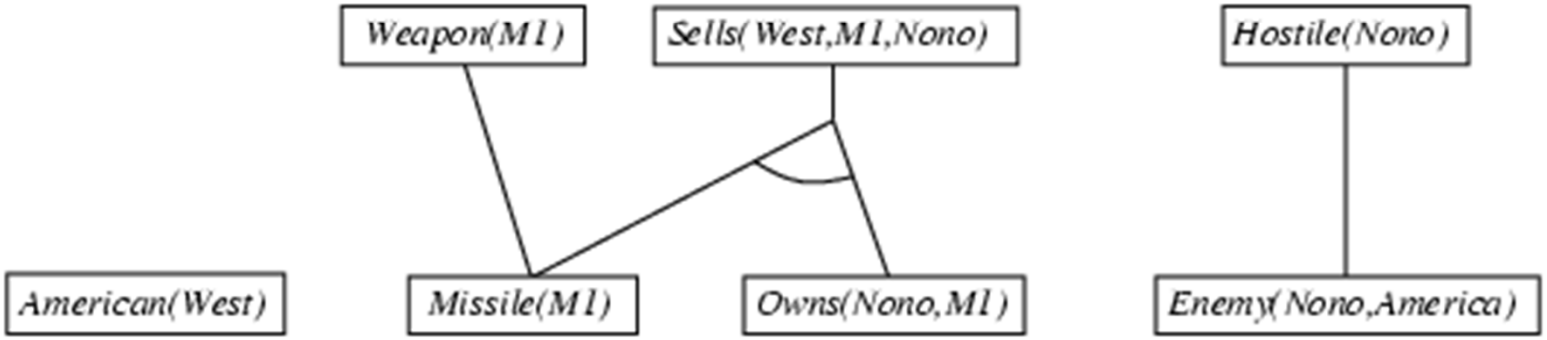


# Forward chaining proof

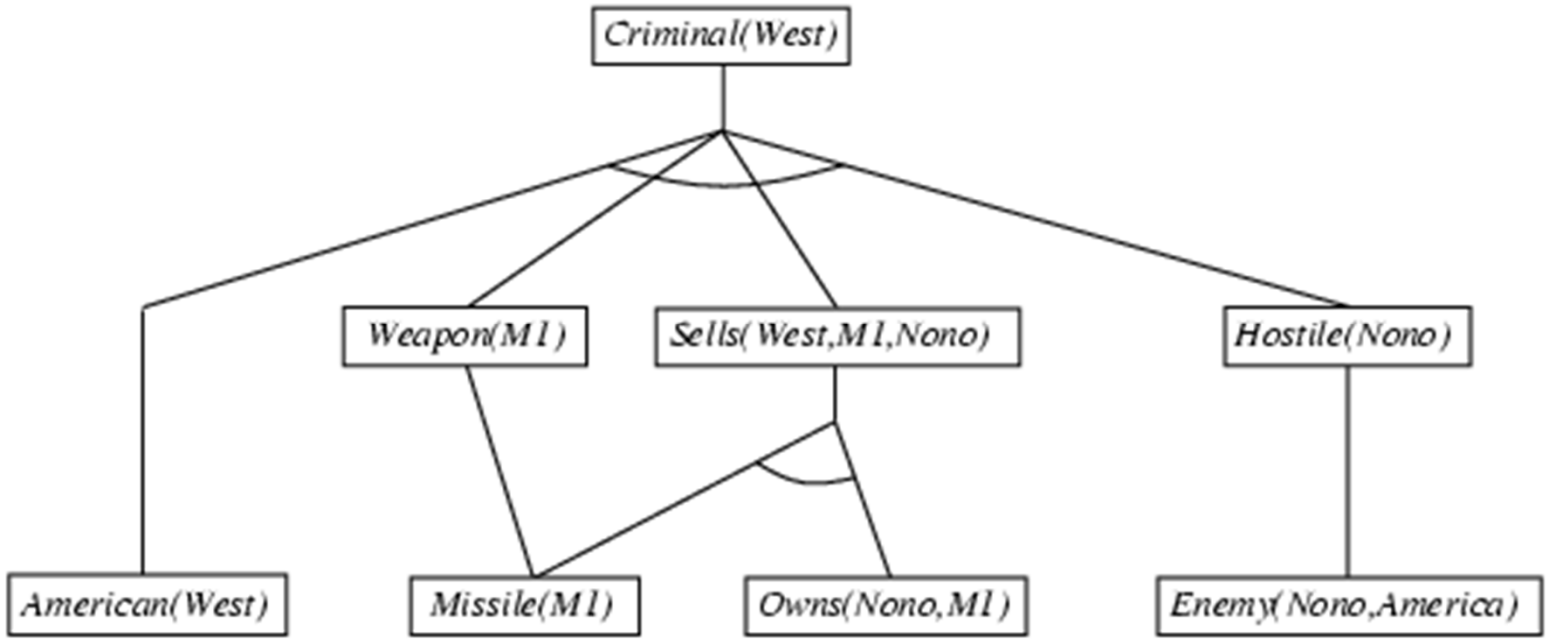





# Forward chaining proof



# Forward chaining proof



# Properties of forward chaining

- Sound and complete for first-order definite clauses

- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations

- May not terminate in general if α is not entailed



# Efficiency of forward chaining

Incremental forward chaining: 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 positive literal

Matching itself can be expensive:

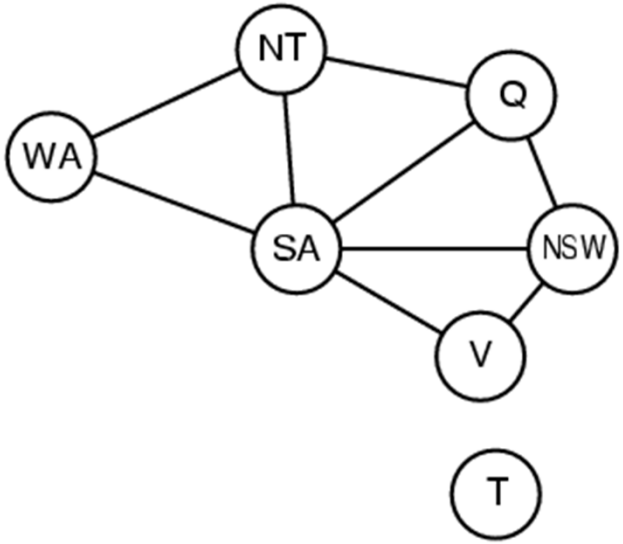
Database indexing allows O(1) retrieval of known facts

- e.g., query *Missile(x)* retrieves *Missile($M_1$)*

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(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, $\theta$) **returns** a set of substitutions
  **inputs**: $KB$, a knowledge base
    $goals$, a list of conjuncts forming a query
    $\theta$, the current substitution, initially the empty substitution $\{\}$
  **local variables**: $ans$, a set of substitutions, initially empty

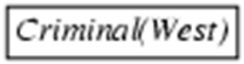
  **if** $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST($\theta$, FIRST($goals$))
  **for each** $r$ **in** $KB$ where STANDARDIZE-APART($r$) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY($q, q'$) succeeds
    $ans \leftarrow$ FOL-BC-ASK($KB$,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

SUBST(COMPOSE($\theta_1$, $\theta_2$), p) = SUBST($\theta_2$, SUBST($\theta_1$, p))



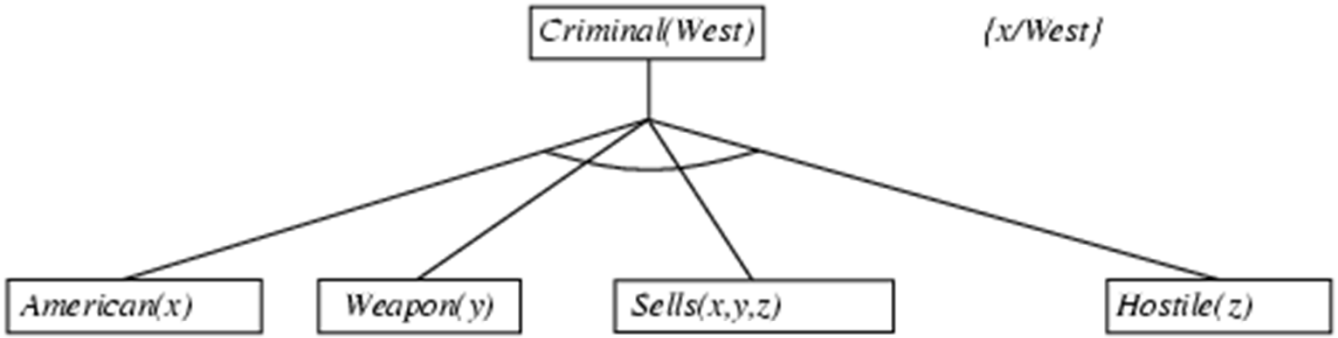
# Backward chaining example

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



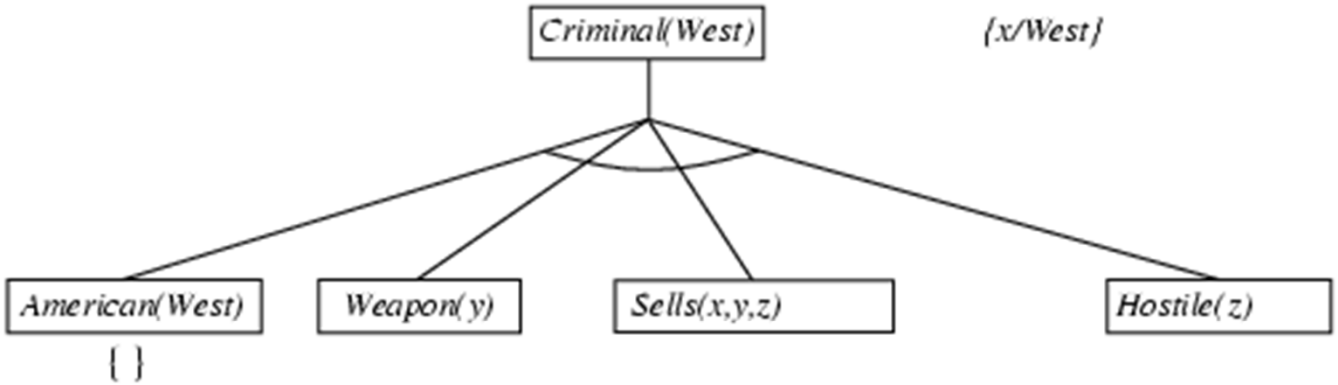



# Backward chaining example





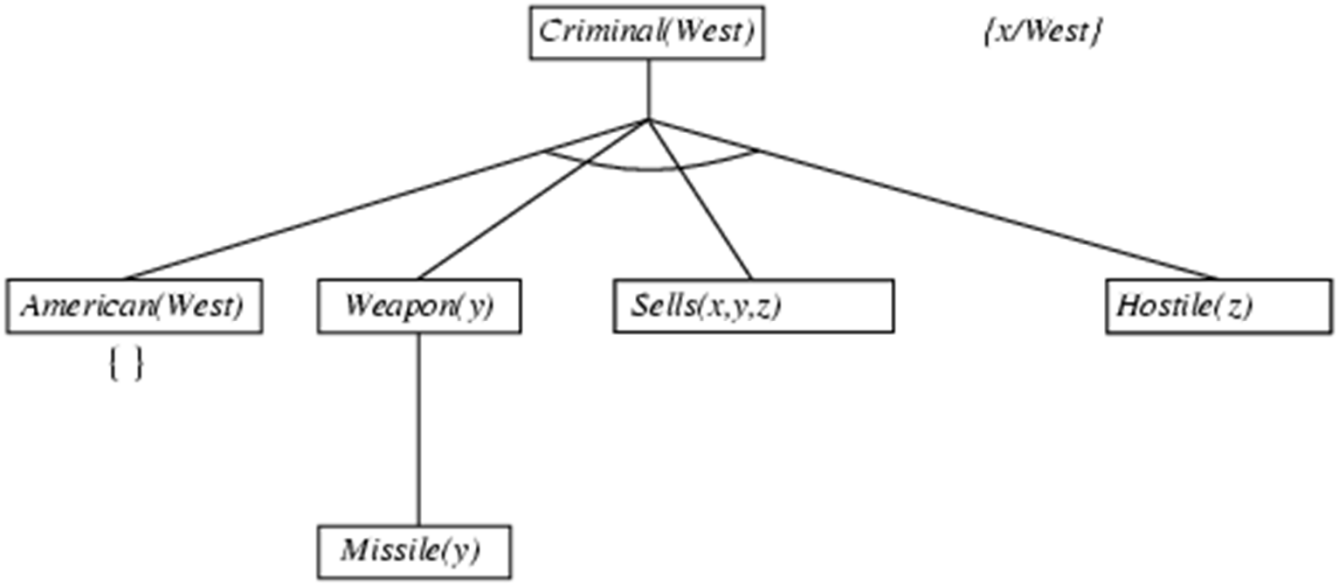

# Backward chaining example





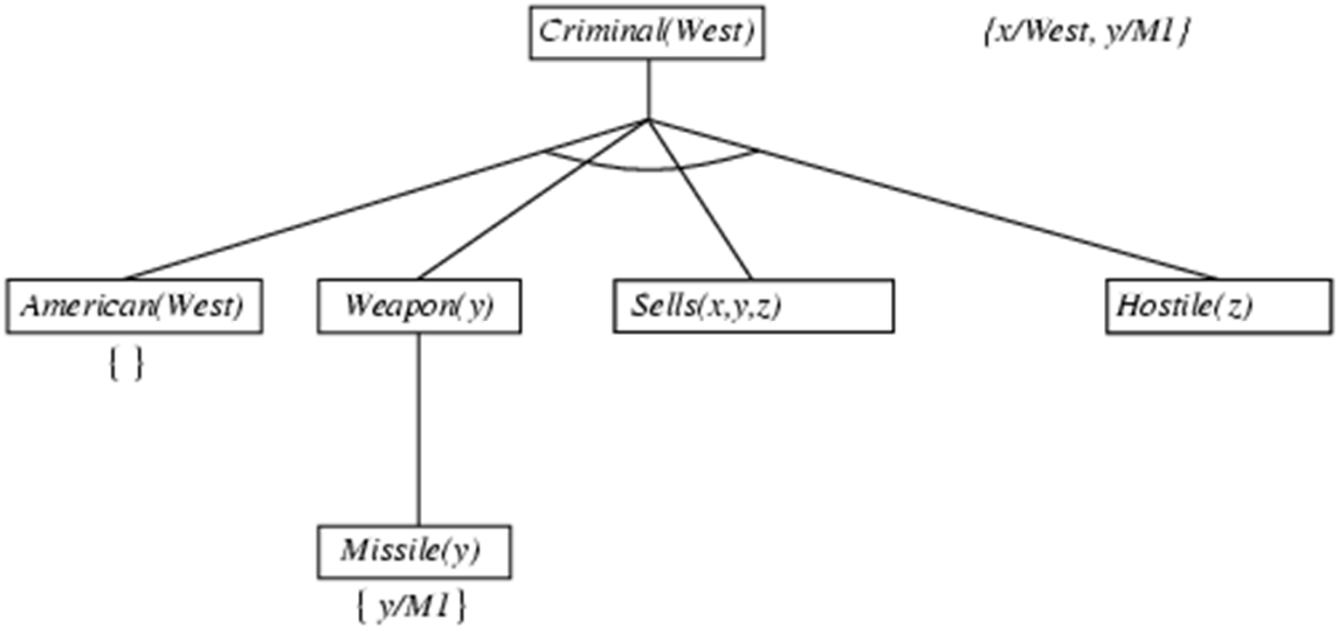

# Backward chaining example

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

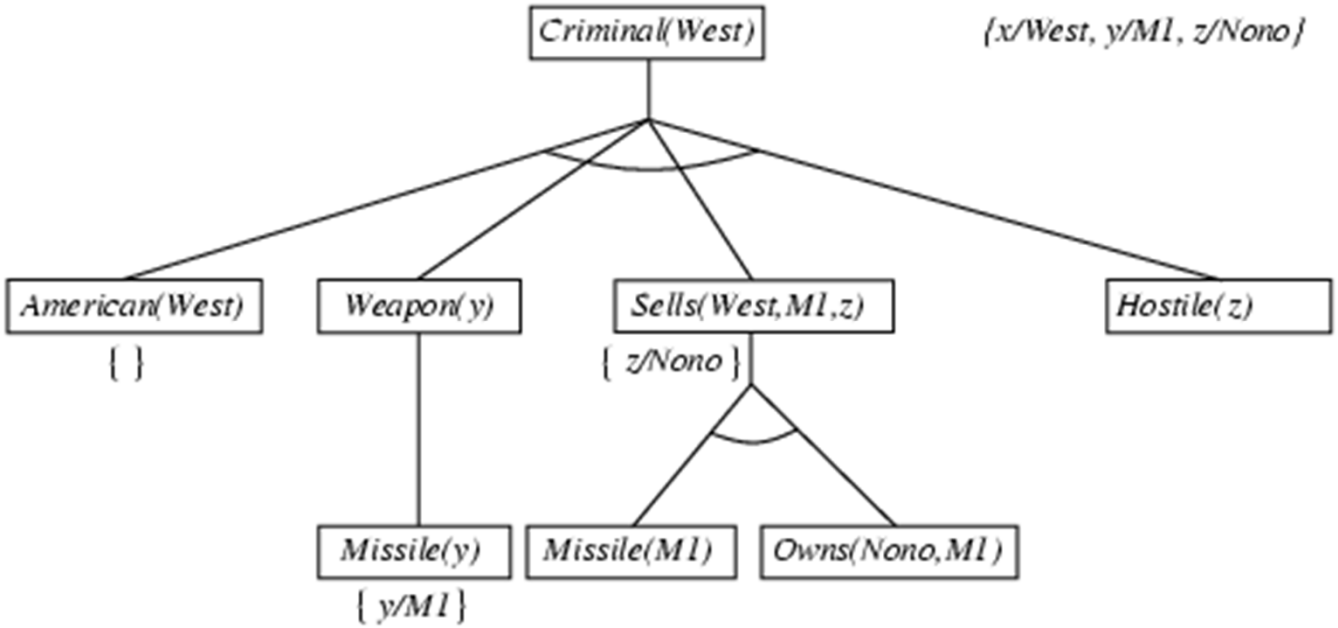


# Backward chaining example

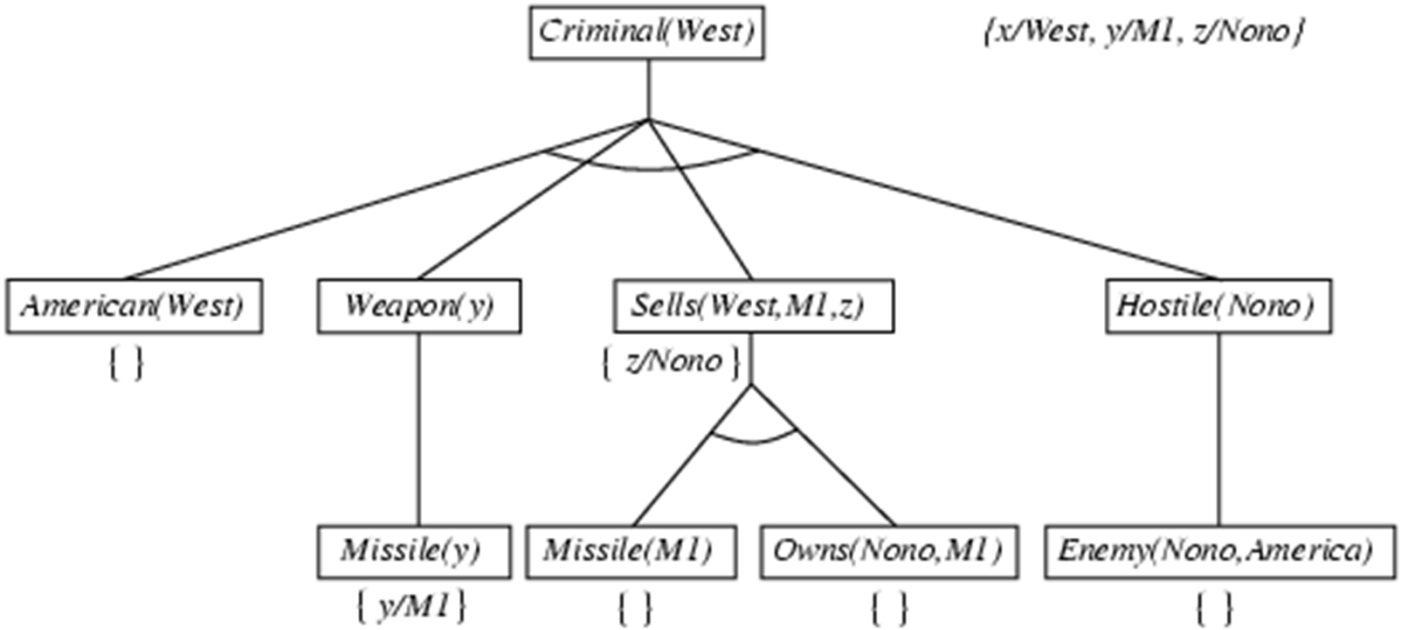
*Owns(Nono,$M_1$) and Missile($M_1$)*



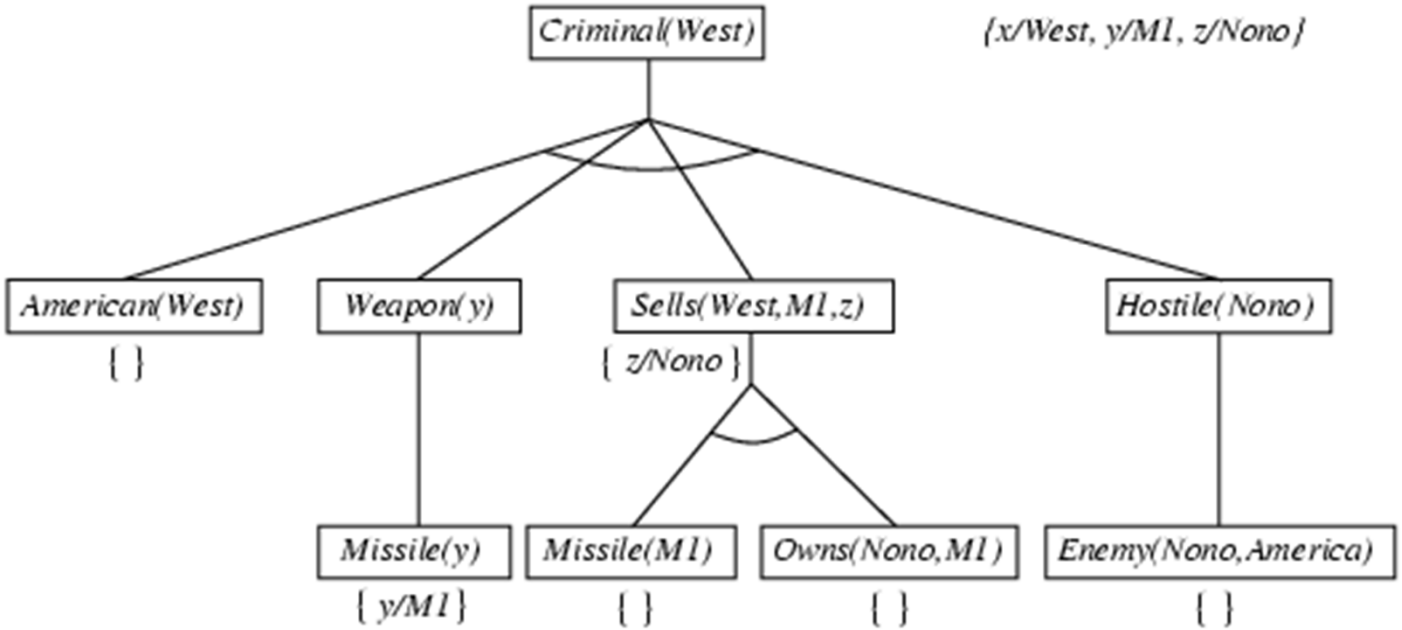
# Backward chaining example



# 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

  $\Rightarrow$ fix by checking current goal against every goal on stack

- Inefficient due to repeated subgoals (both success and failure)

  $\Rightarrow$ fix using caching of previous results (extra space)

- Widely used for logic programming



# Summary

- Inference in First-Order Logic
- Reduction to propositional inference
- Generalized Modus Ponens
- Unification
- Forward and Backward Chaining
- Prolog