

Inference in first-order logic

Chapter 9

Outline

- 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., $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields:

-

-

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

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

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., $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ yields:

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

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

Reduction to propositional inference

Suppose the KB contains just the following:

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

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

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

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

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

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

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

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

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

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

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

$\text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}), \text{etc.}$

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

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 ∞ 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.
- 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 *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)*
-

$\theta = \{x/\text{John}, y/\text{John}\}$ works

<ul style="list-style-type: none"> • $\text{Unify}(\alpha, \beta) = \theta$ 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

• $Unify(\alpha, \beta) = \theta$ 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

• $Unify(\alpha, \beta) = \theta$ 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

<ul style="list-style-type: none"> • $Unify(\alpha, \beta) = \theta$ 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

<ul style="list-style-type: none"> • $Unify(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$ • 		
p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, OJ)$	$\{fail\}$

Unification

- To unify $Knows(John, x)$ and $Knows(y, z)$,
- $\theta = \{y/John, x/z\}$ or $\theta = \{y/John, x/John, z/John\}$
- The first unifier is **more general** than the second.
-
- 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, θ) **returns** a substitution to make x and y identical

inputs: x , a variable, constant, list, or compound

y , a variable, constant, list, or compound

θ , the substitution built up so far

if $\theta = \text{failure}$ **then return failure**

else if $x = y$ **then return** θ

else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)

else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)

else if COMPOUND?(x) **and** COMPOUND?(y) **then**

return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))

else if LIST?(x) **and** LIST?(y) **then**

return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))

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)

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

p_1' is <i>King(John)</i>	p_1 is <i>King(x)</i>
p_2' is <i>Greedy(y)</i>	p_2 is <i>Greedy(x)</i>
θ is $\{x/\text{John}, y/\text{John}\}$	q is <i>Evil(x)</i>
$q \theta$ is <i>Evil(John)</i>	

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

Soundness of GMP

- Need to show that

-

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

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

- Lemma: For any sentence 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.

2. $p_1', \dots, p_n' \models p_1' \wedge \dots \wedge p_n' \models p_1'\theta \wedge \dots \wedge p_n'\theta$

3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

- 4.

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
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)$ 
      for each  $\theta$  such that  $(p_1 \wedge \dots \wedge p_n)\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$ 
        for some  $p'_1, \dots, p'_n$  in  $KB$ 
           $q' \leftarrow \text{SUBST}(\theta, q)$ 
          if  $q'$  is not a renaming of a sentence already in  $KB$  or new then do
            add  $q'$  to new
             $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
            if  $\phi$  is not fail then return  $\phi$ 
    add new to  $KB$ 
  return false
```

Forward chaining proof

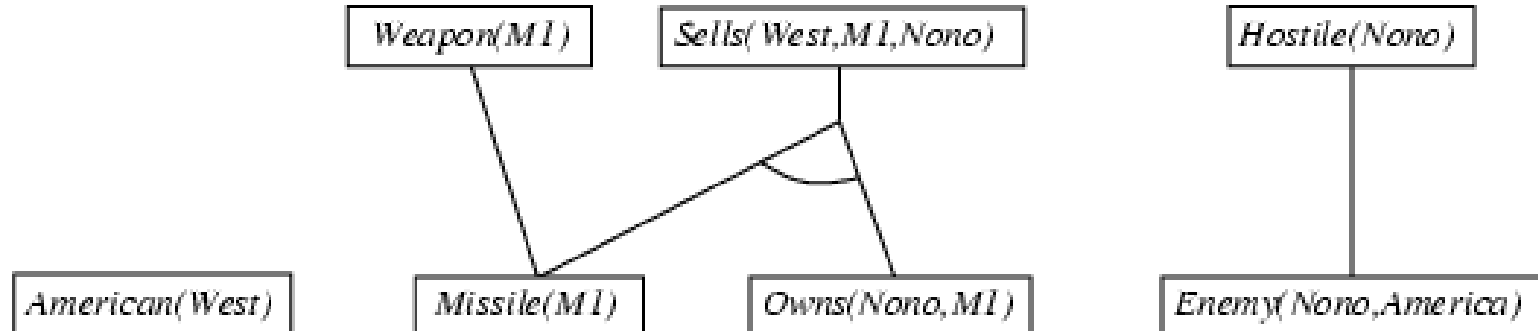
American(West)

Missile(M1)

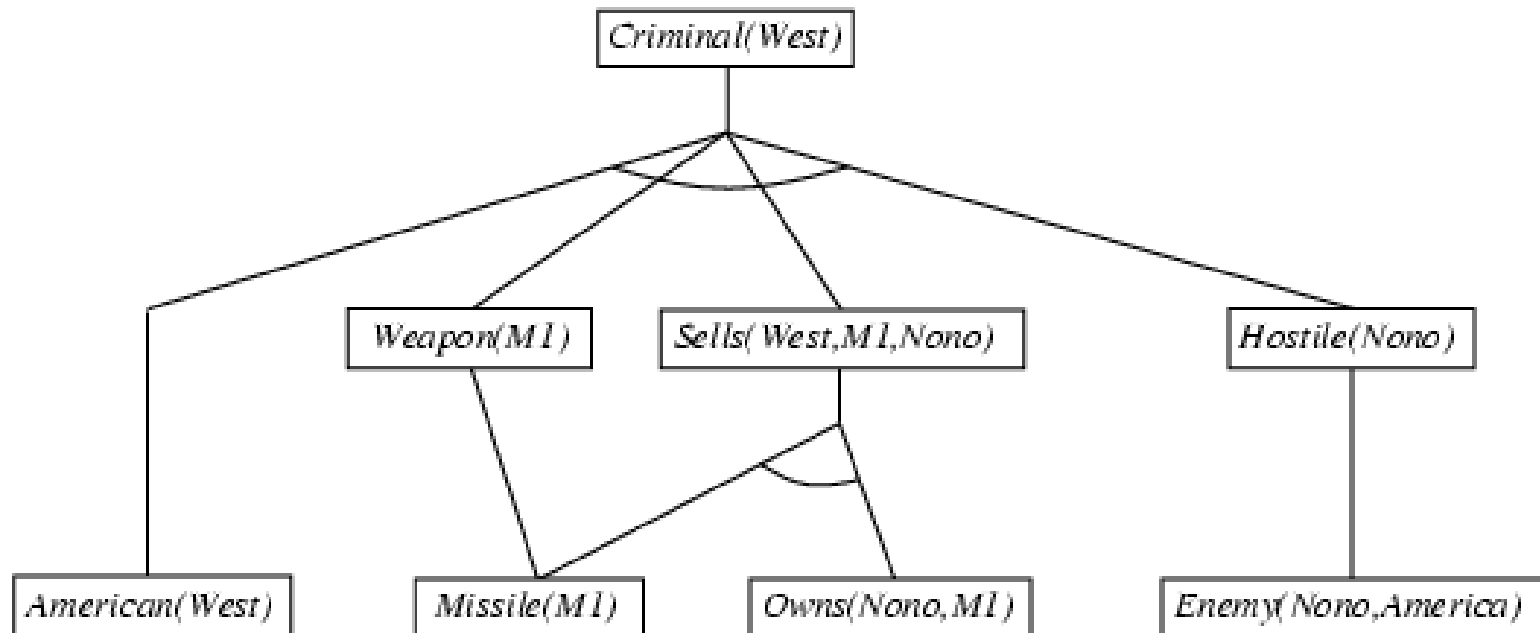
Owns(Nono,M1)

Enemy(Nono,America)

Forward chaining proof



Forward chaining proof



Properties of forward chaining

- Sound and complete for first-order definite clauses
-
- **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
-
- This is unavoidable: entailment with definite clauses is semidecidable
-

Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration k if a premise wasn't added on iteration $k-1$
⇒ match each rule whose premise contains a newly added positive literal

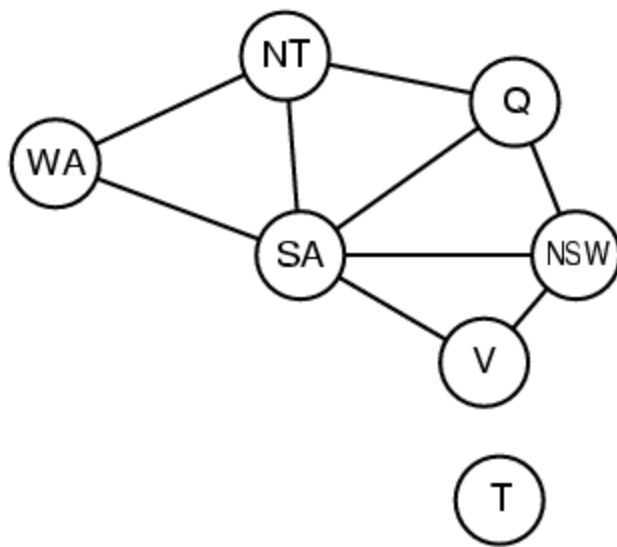
Matching itself can be expensive:

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



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

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

- $Colorable()$ 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

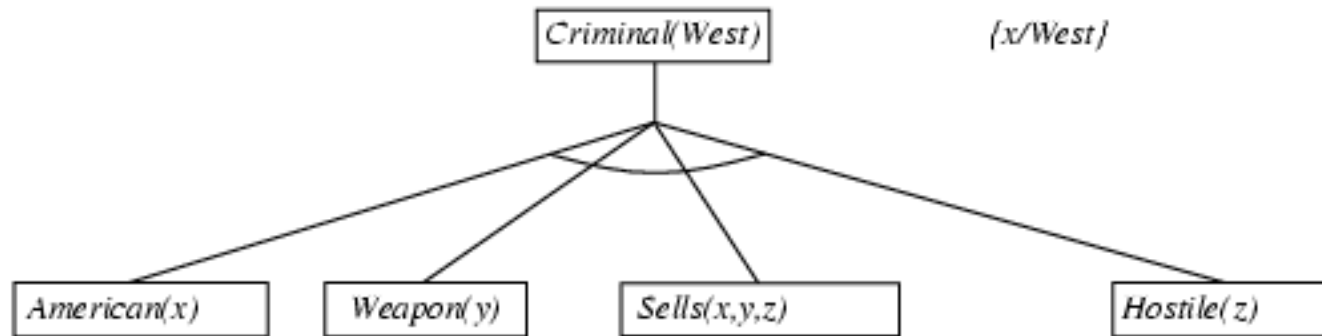
  if  $goals$  is empty then return  $\{ \theta \}$ 
   $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$ 
  for each  $r$  in  $KB$  where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
       $ans \leftarrow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

$$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$$

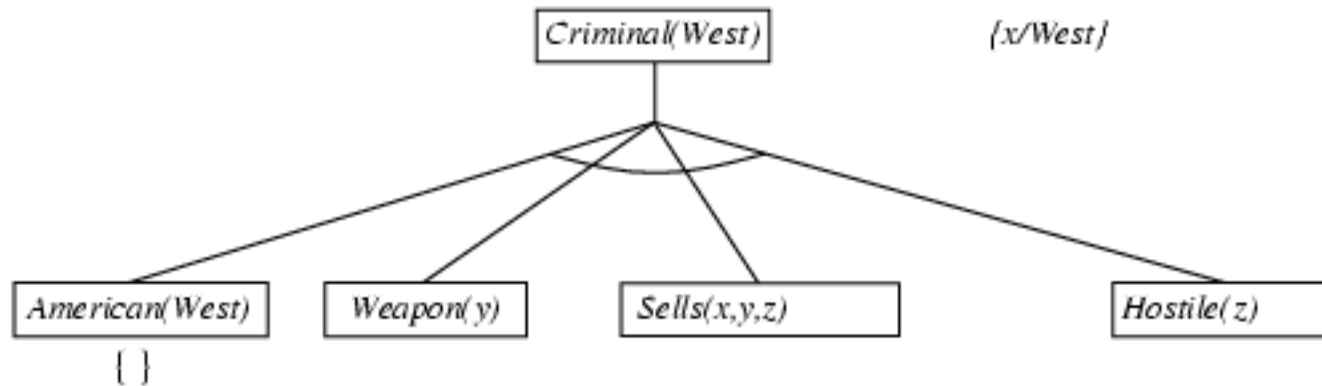
Backward chaining example

Criminal(West)

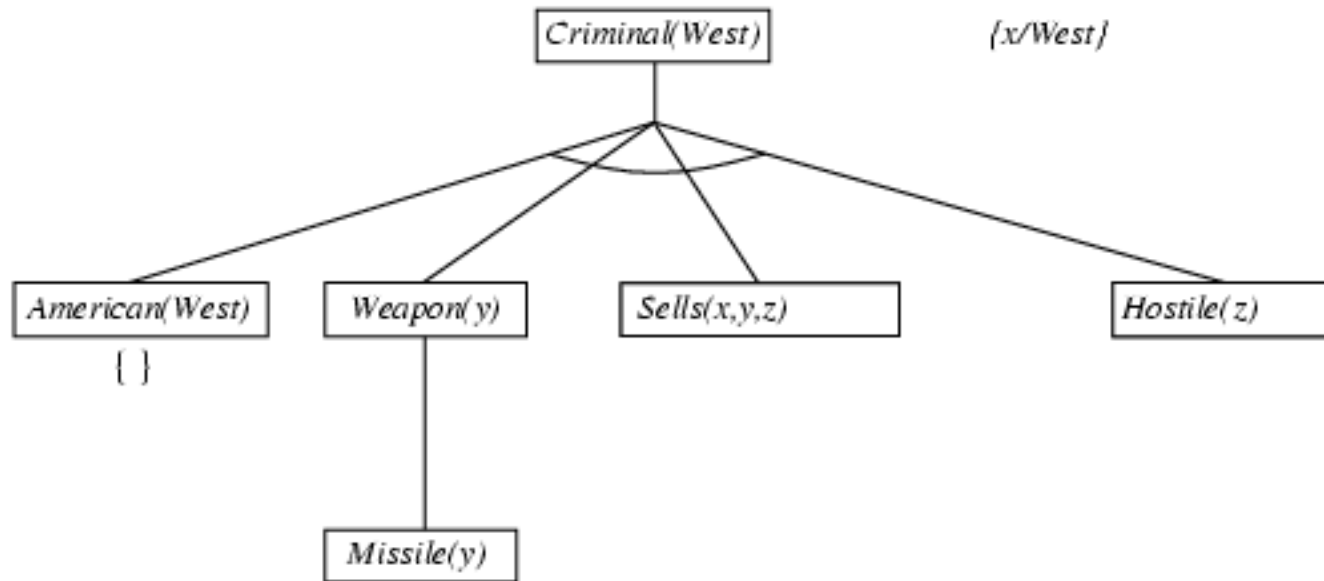
Backward chaining example



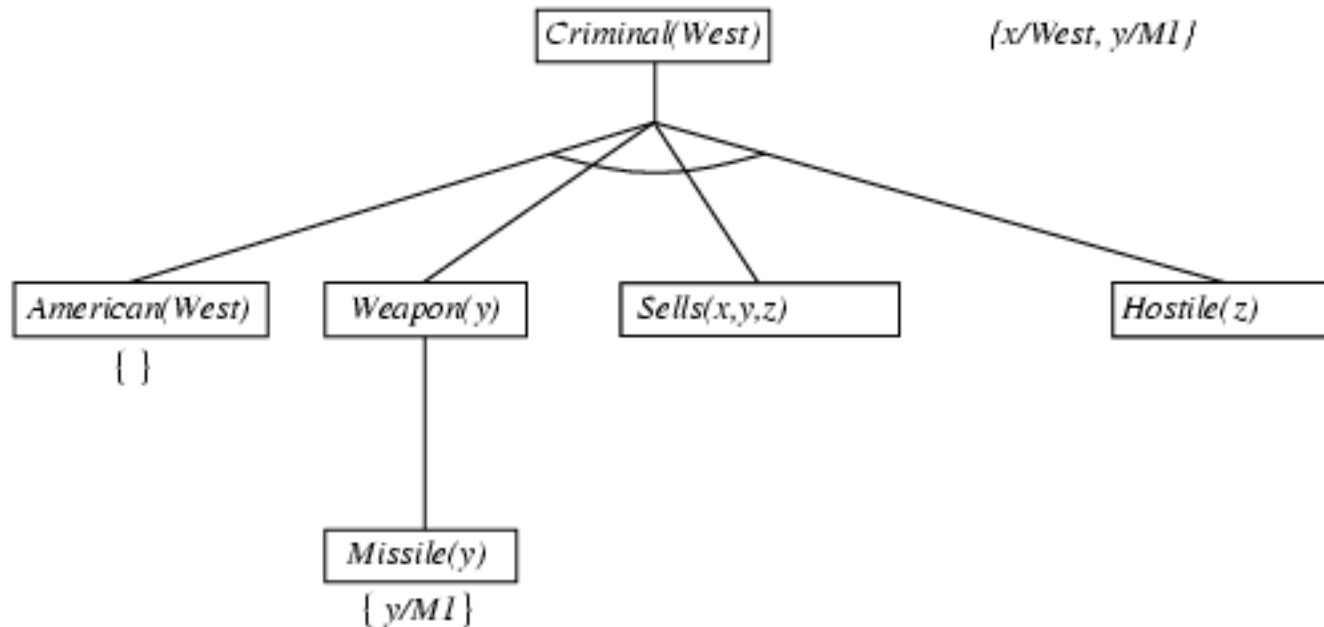
Backward chaining example



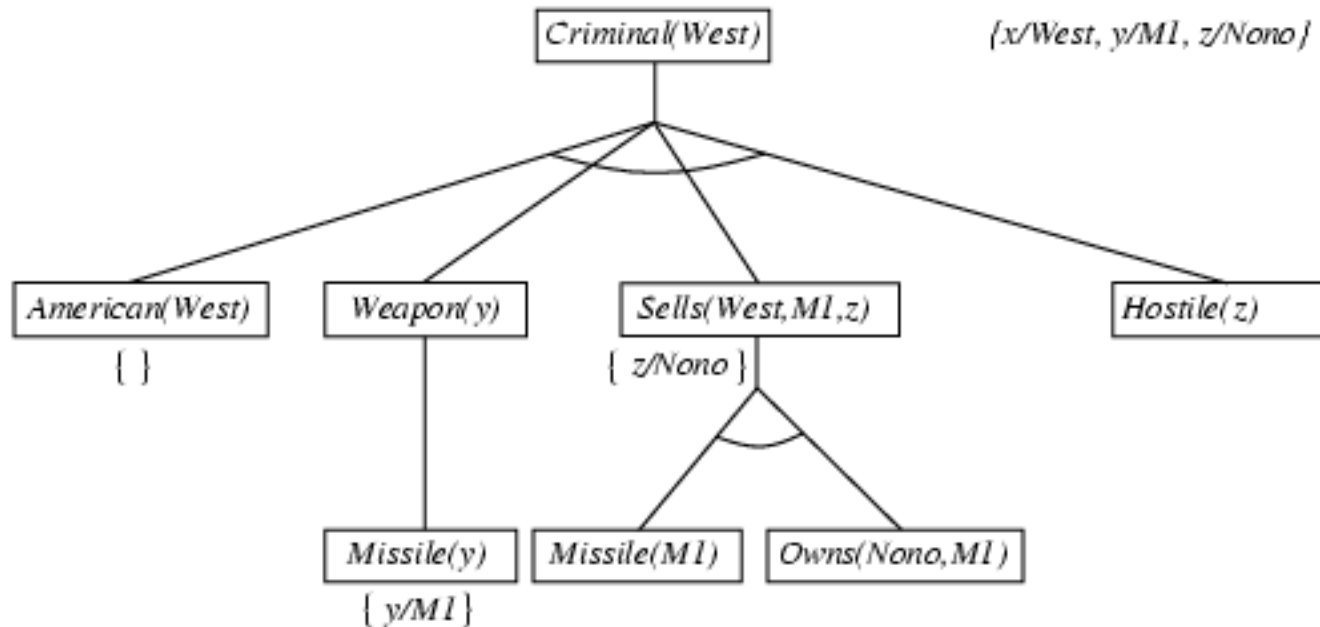
Backward chaining example



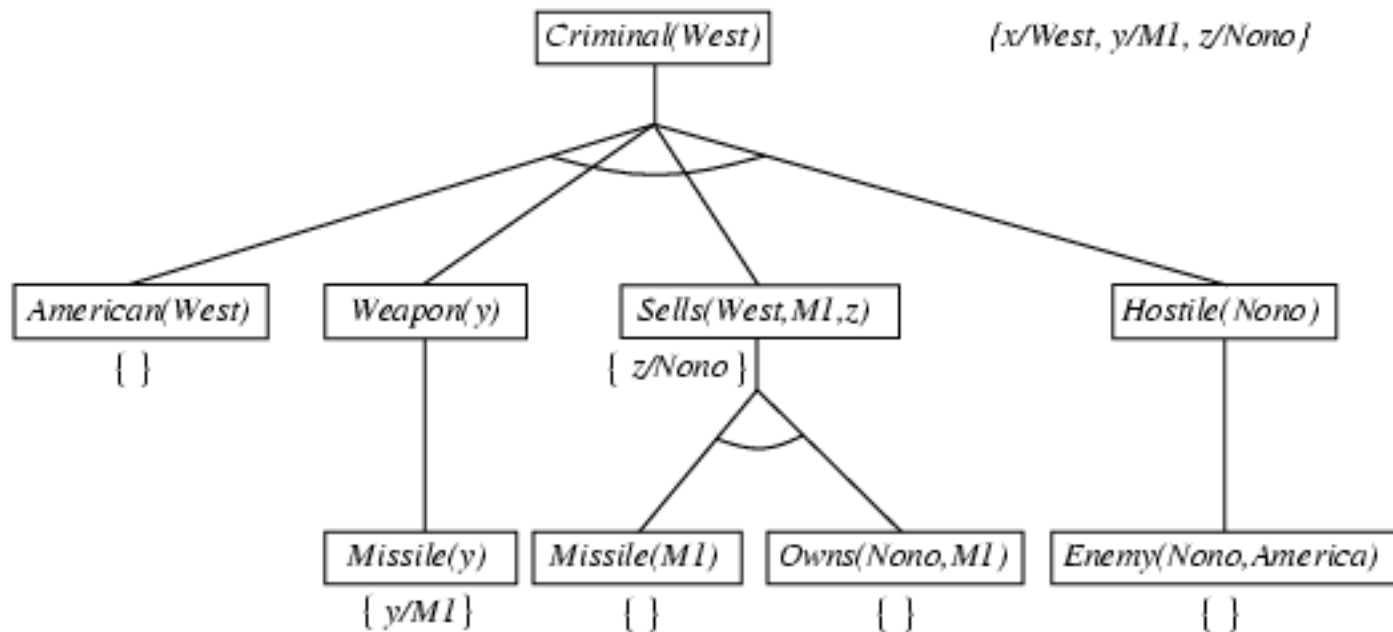
Backward chaining example



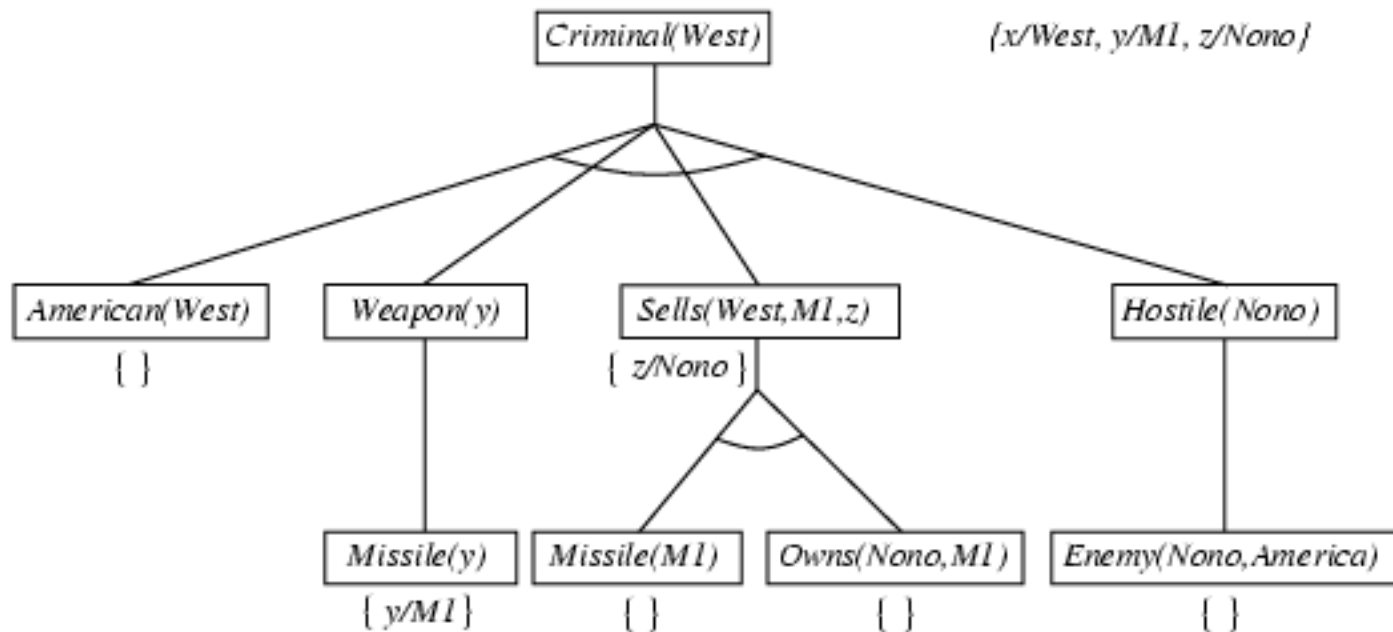
Backward chaining example



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
- - \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)

Logic programming: Prolog

- Algorithm = Logic + Control
-
- Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques \Rightarrow 60 million LIPS
- Program = set of clauses = `head :- literal1, ... literaln.`
- `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`
- Built-in predicates that have side effects (e.g., input and output)
-
- predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
= e.g. `given_alive(X) :- not dead(X)`

Prolog

- 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:

-

$$\ell_1 \vee \dots \vee \ell_k, \quad m_1 \vee \dots \vee m_n$$

$$(\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta$$

where $\text{Unify}(\ell_i, \neg m_j) = \theta$.

- The two clauses are assumed to be standardized apart so that they share no variables.

-

- For example,

-

$$\begin{array}{c} \neg \text{Rich}(x) \vee \text{Unhappy}(x) \\ \text{Rich}(\text{Ken}) \\ \text{Unhappy}(\text{Ken}) \end{array}$$

with $\theta = \{x/\text{Ken}\}$

Conversion to CNF

- Everyone who loves all animals 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 biconditionals and implications

-

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

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

-

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

Conversion to CNF contd.

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

4.

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

4. Skolemize: a more general form of existential instantiation.

Each existential variable is replaced by a **Skolem function** of the enclosing universally quantified variables:

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

5. Drop universal quantifiers:

6.

6.

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

6. Distribute \vee over \wedge :

Resolution proof: definite clauses

