

Inference in first-order logic

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

Existential instantiation (EI)

- For any sentence 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:

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
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 inference procedure, 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 ∞ 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 $Greedy(y)$

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

- $Unify(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	
$Knows(John, x)$	$Knows(x, OJ)$	

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

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)	

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

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
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	
$Knows(John, x)$	$Knows(x, OJ)$	

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

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))	$\{y/John, x/Mother(John)\}$
Knows(John,x)	Knows(x,OJ)	

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

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))	{y/John,x/Mother(John)}}}
Knows(John,x)	Knows(x,OJ)	{fail}

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

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 = $\{y/John, x/z\}$

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 *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

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 proof

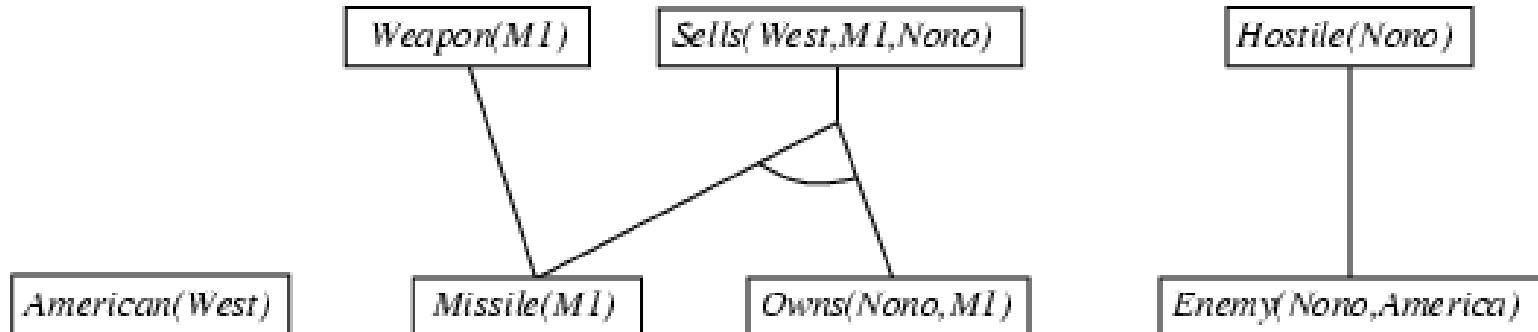
American(West)

Missile(M1)

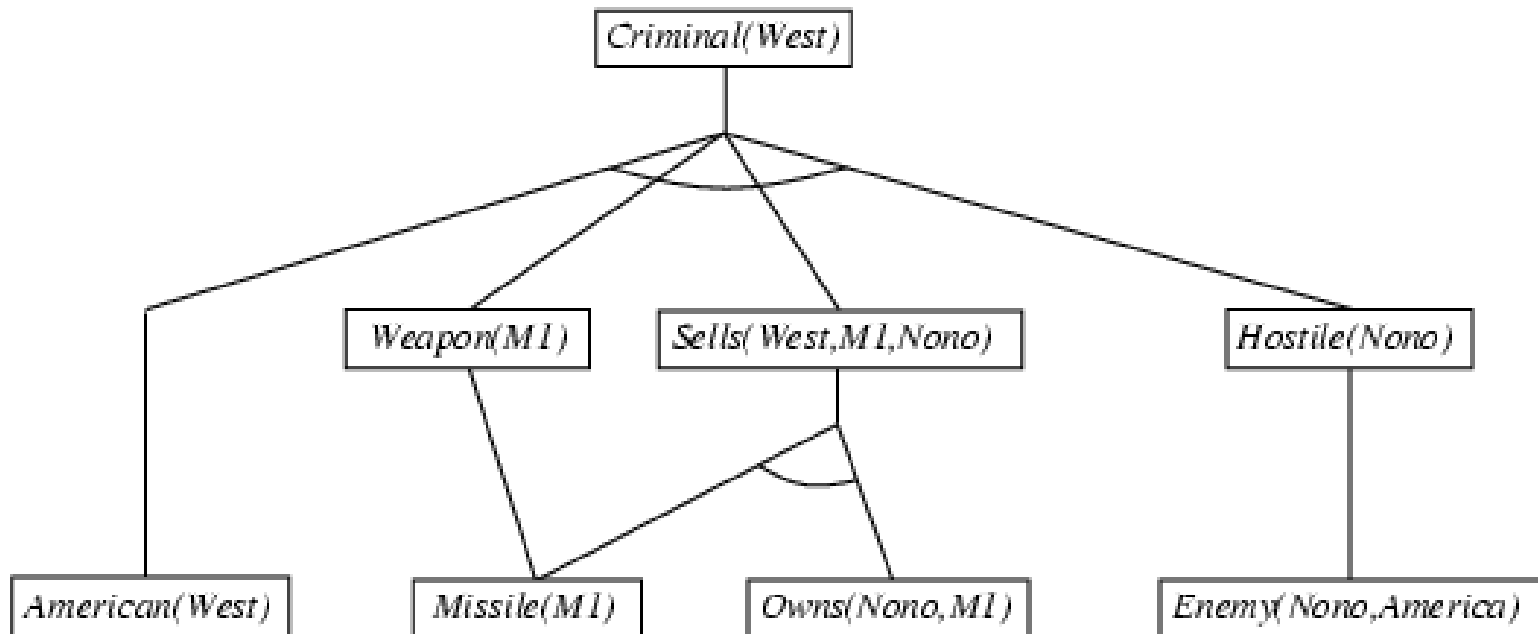
Owns(Nono,M1)

Enemy(Nono,America)

Forward chaining proof



Forward chaining proof



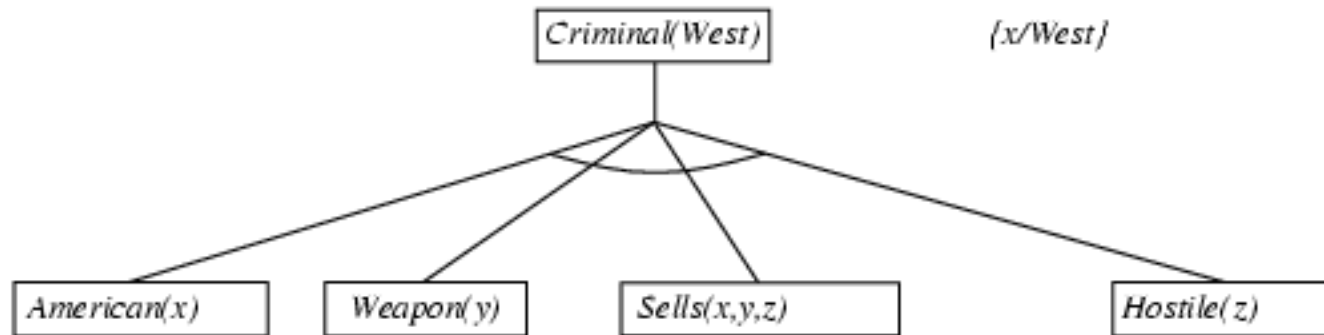
Properties of forward chaining

- 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

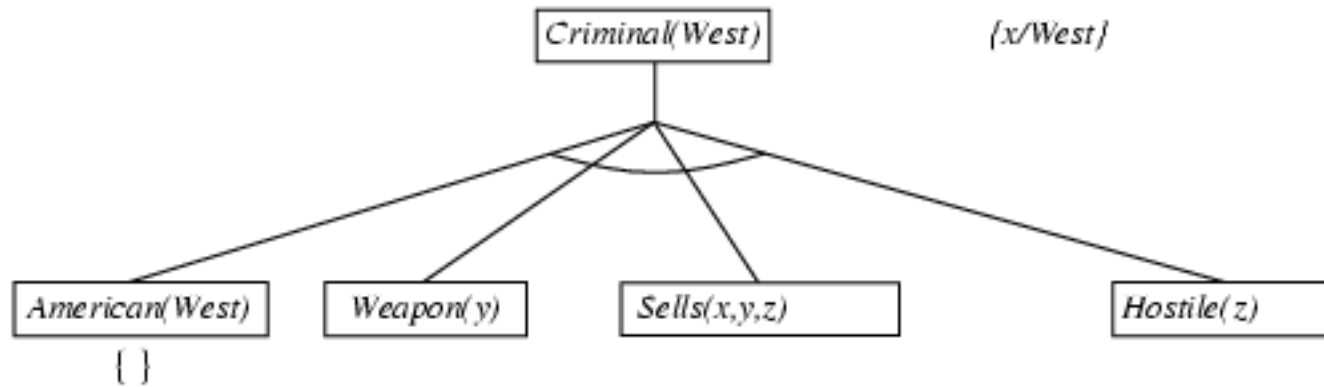
Backward chaining example

Criminal(West)

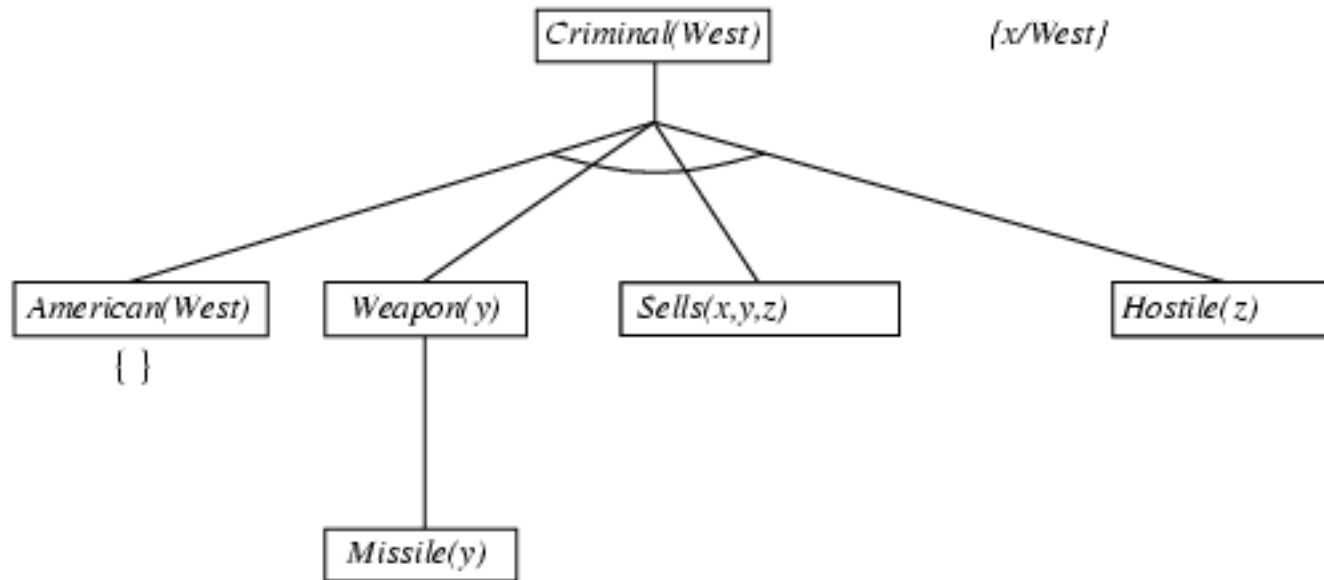
Backward chaining example



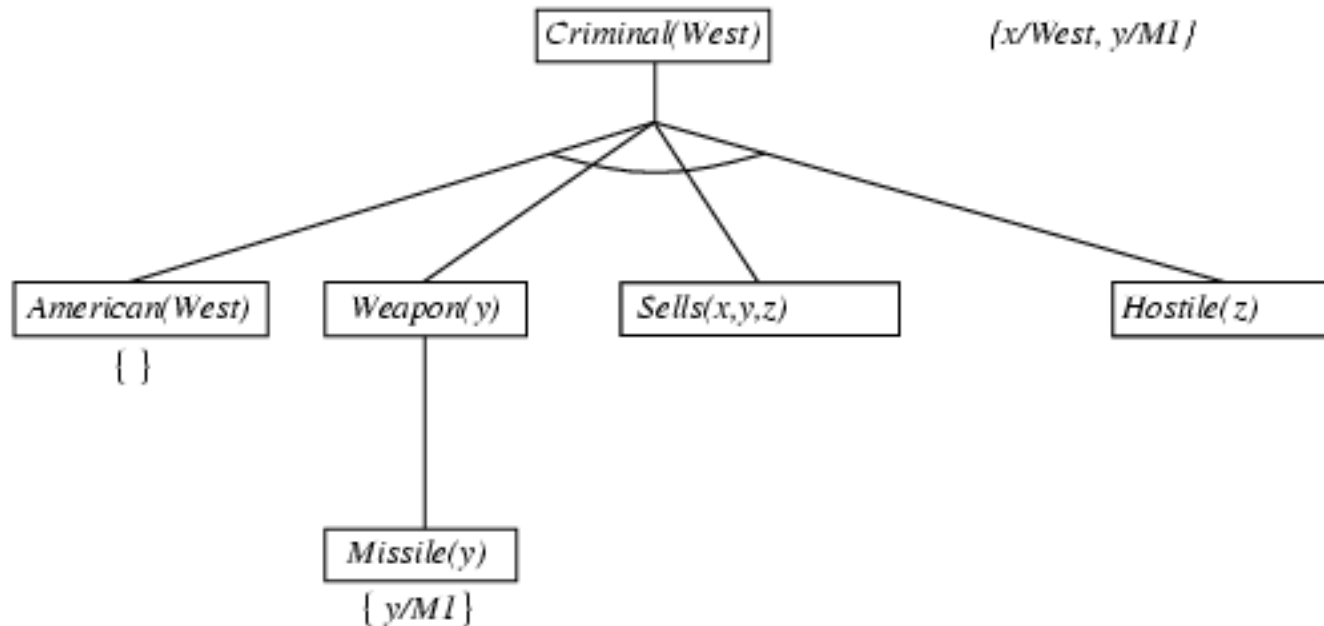
Backward chaining example



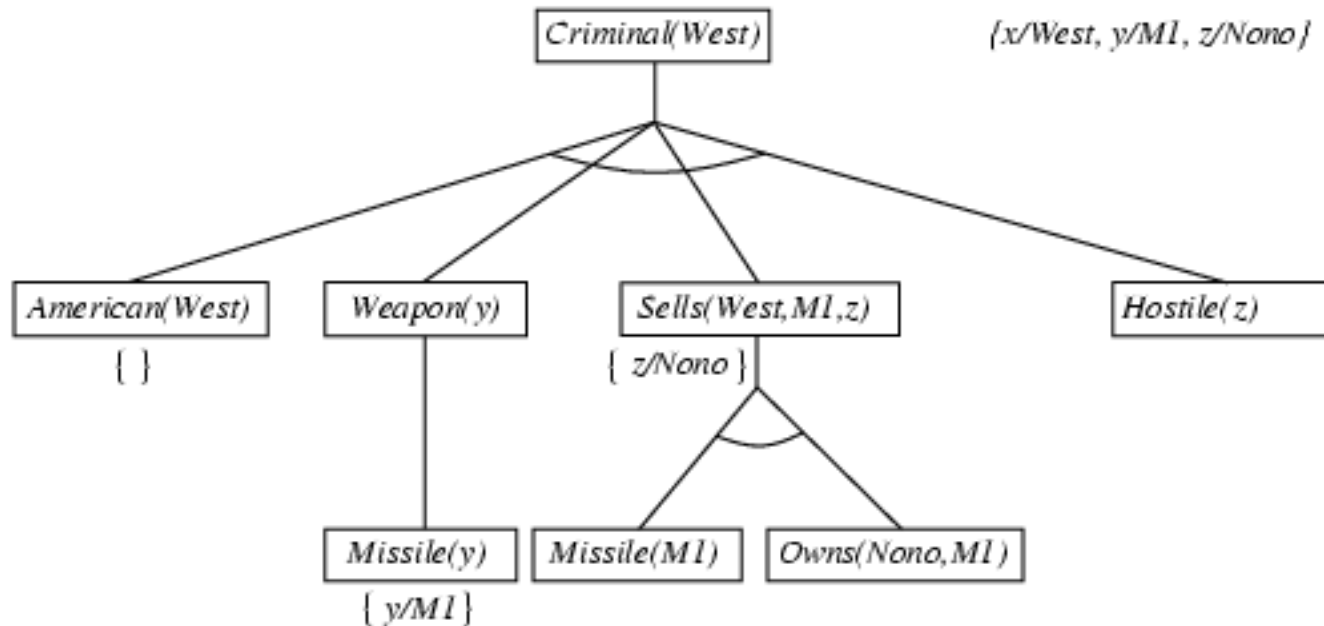
Backward chaining example



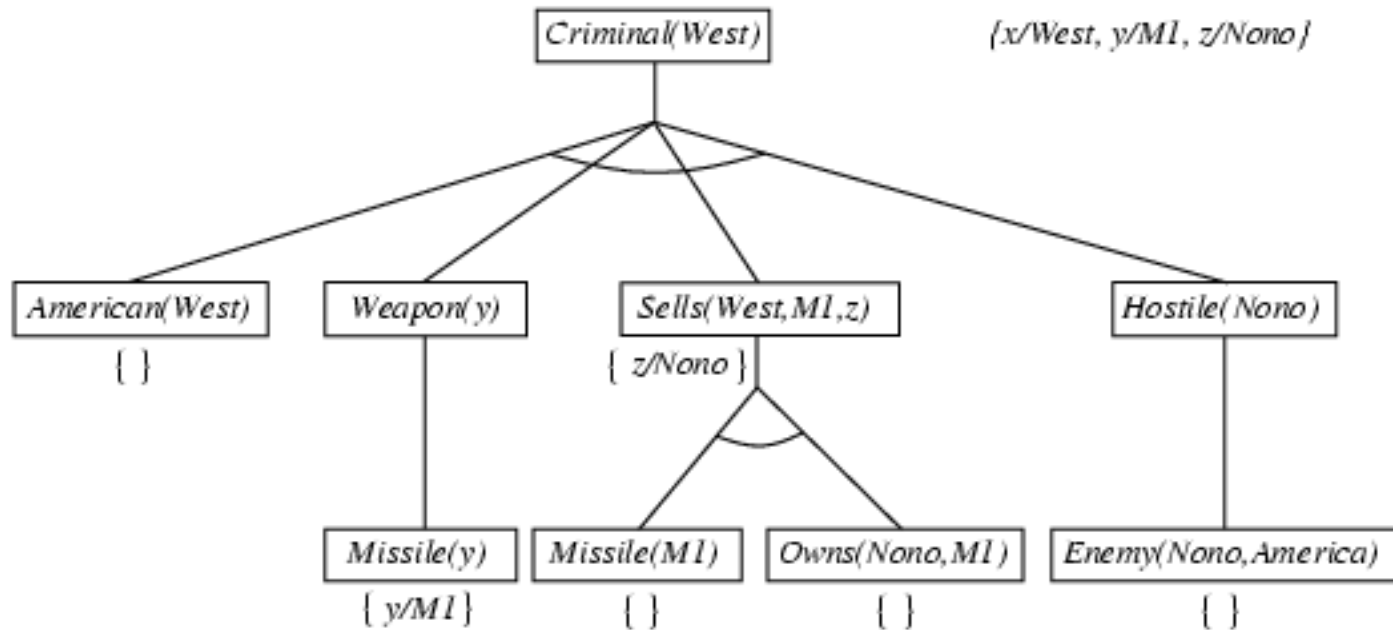
Backward chaining example



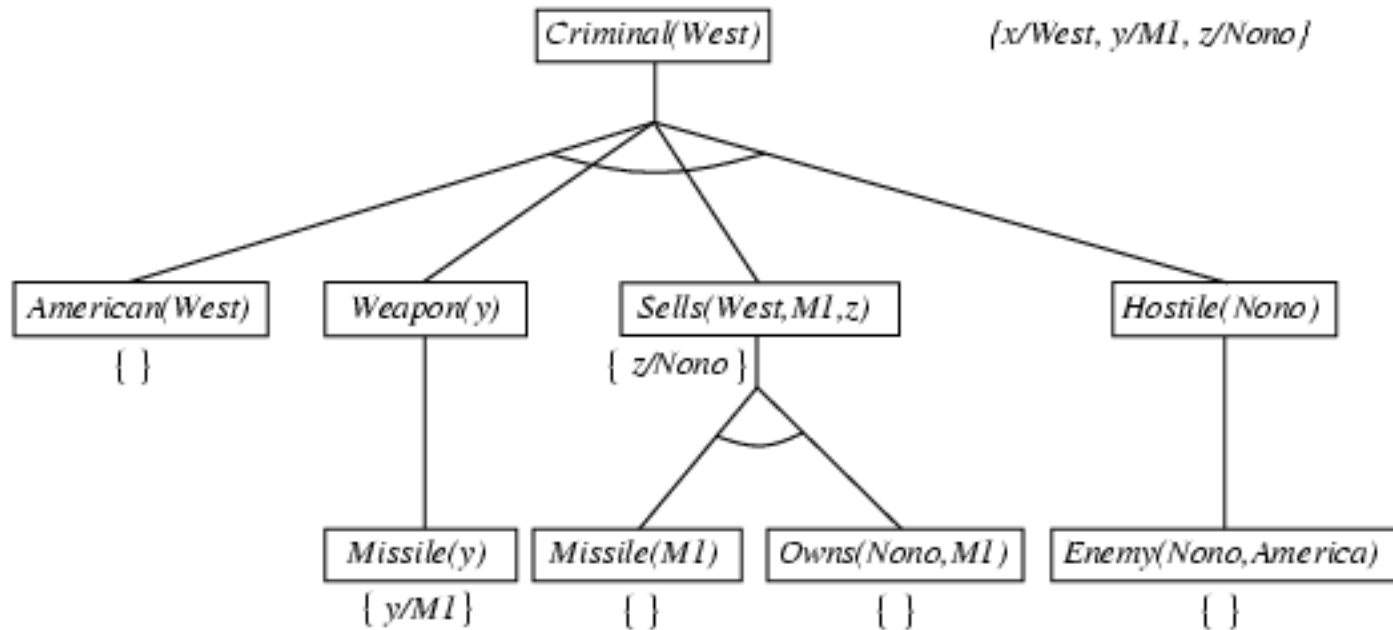
Backward chaining example



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)
- Widely used for **logic programming**

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
- Program = set of clauses = head :- literal₁, ... literal_n.
`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).`
 - `alive(joe)` succeeds if `dead(joe)` fails

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=[]`