Inference in First-Order Logic

CHAPTER 9
HASSAN KHOSRAVI
SPRING2011

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution

Universal instantiation (UI)

- Notation: Subst({v/g}, α) means the result of substituting g for v in sentence α

- 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

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$, **{x/John}**

$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$, **{x/Richard}**

$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$, **{x/Father(John)}**

Existential instantiation (EI)

- For any sentence α, variable *v*, and constant symbol *k* (that does not appear elsewhere in the knowledge base):

$$\frac{\exists v\ \alpha}{\mathrm{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields: $Crown(C_1) \wedge OnHead(C_1,John)$
-

  where C_1 is a new constant symbol, called a Skolem constant

- Existential and universal instantiation allows to "propositionalize" any FOL sentence or KB
  - EI produces one instantiation per EQ sentence
  - UI produces a whole set of instantiated sentences per UQ sentence
  -

Reduction to propositional form

Suppose the KB contains the following:

$\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
Father(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: propositional symbols are
  - King(John), Greedy(John), Evil(John), King(Richard), etc

Reduction continued

- 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 for doing inference in FOL:
  - propositionalize KB and query
  - apply resolution-based inference
  - return result
- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., *Father*(*Father*(*Father*(*John*))), etc

Reduction continued

Theorem: Herbrand (1930). If a sentence α is entailed by a 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

Example

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

Query Evil(X)?

- Depth 0

  $Father(John)$

  $Father(Richard)$

  $King(John)$

  $Greedy(Richard)$

  $Brother(Richard, John)$

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

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

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

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

- Depth 1

  Depth 0 +

  $Father(Father(John))$

  $Father(Father(John))$

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

Problems with Propositionalization

- Problem: works if α is entailed, loops if α is not entailed

- Propositionalization generates lots of irrelevant sentences
  - So inference may be very inefficient

- e.g., from:

  ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
  King(John)
  ∀y Greedy(y)
  Brother(Richard,John)

- It seems obvious that *Evil*(*John*) is entailed, 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
- Lets see if we can do inference directly with FOL sentences

Unification

- Recall: Subst(θ, p) = result of substituting θ into sentence p

- Unify algorithm: takes 2 sentences p and q and returns a unifier if one exists

  Unify(p,q) = θ where Subst(θ, p) = Subst(θ, q)

- Example:
  p = Knows(John,x)
  q = Knows(John, Jane)

  Unify(p,q) = {x/Jane}

Unification examples

- simple example: query = Knows(John,x), i.e., who does John know?

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}

- Last unification fails: only because x can’t take values John and OJ at the same time
- Problem is due to use of same variable x in both sentences
- Simple solution: Standardizing apart eliminates overlap of variables, e.g., Knows(z,OJ)

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.
- There is a single most general unifier (MGU) that is unique up to renaming of variables.
- MGU = { y/John, x/z }
- General algorithm in Figure 9.1 in the text

Recall our example...

$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

King(John)

$\forall y\ Greedy(y)$

Brother(Richard,John)

And we would like to infer Evil(John) without propositionalization

Generalized Modus Ponens (GMP)

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

where we can unify p_i' and p_i for all *i*

Example:

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

p_1' is *King*(*John*) p_1 is *King*(*x*)

p_2' is *Greedy*(*John*) p_2 is *Greedy*(*x*)

θ is {x/John} q is *Evil*(*x*)

Subst(θ,q) is *Evil*(*John*)

Completeness and Soundness of GMP

- GMP is sound
  - Only derives sentences that are logically entailed
  - See proof on p276 in text
- GMP is complete for a KB consisting of Horn clauses
  - Complete: derives all sentences that entailed

Horn Clauses

- Resolution in general can be exponential in space and time.

- If we can reduce all clauses to "Horn clauses" resolution is linear in space and time

A clause with at most 1 positive literal.
e.g. $A \vee \neg B \vee \neg C$

- Every Horn clause can be rewritten as an implication with a conjunction of positive literals in the premises and a single positive literal as a conclusion.

e.g. $B \wedge C \Rightarrow A$

- 1 positive literal: definite clause

- 0 positive literals: Fact or integrity constraint:
  e.g. $(\neg A \vee \neg B) \equiv (A \wedge B \Rightarrow False)$

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 $p_i'\theta = p_i\theta$ for all I

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

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
2.
2. $p_1', \;; \ldots, \;;p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens
4.

Storage and retrieval

- Storage(s): stores a sentence s into the knowledge base

- Fetch(q): returns all unifiers such that the query q unifies with some sentence.

- Simple naïve method. Keep all facts in knowledge base in one long list and then call unify(q,s) for all sentences to do fetch.
  - Inefficient but works

- Unification is only attempted on sentence with chance of unification. (knows(john, x) , brother(richard,john))
  - Predicate indexing
  - If many instances of the same predicate exist (tax authorities employer(x,y))
    - Also index arguments
    - Keep latice p280

Inference appoaches in FOL

- Forward-chaining
  - Uses GMP to add new atomic sentences
  - Useful for systems that make inferences as information streams in
  - Requires KB to be in form of first-order definite clauses
- Backward-chaining
  - Works backwards from a query to try to construct a proof
  - Can suffer from repeated states and incompleteness
  - Useful for query-driven inference
- Resolution-based inference (FOL)
  - Refutation-complete for general KB
    - Can be used to confirm or refute a sentence p (but not to generate all entailed sentences)
  - Requires FOL KB to be reduced to CNF
  - Uses generalized version of propositional inference rule
- Note that all of these methods are generalizations of their propositional equivalents

Knowledge Base in FOL

- 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.
-

Knowledge Base in FOL

- 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.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile“:

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Forward chaining algorithm

- Definite clauses → disjunctions of literals of which exactly one is positive.

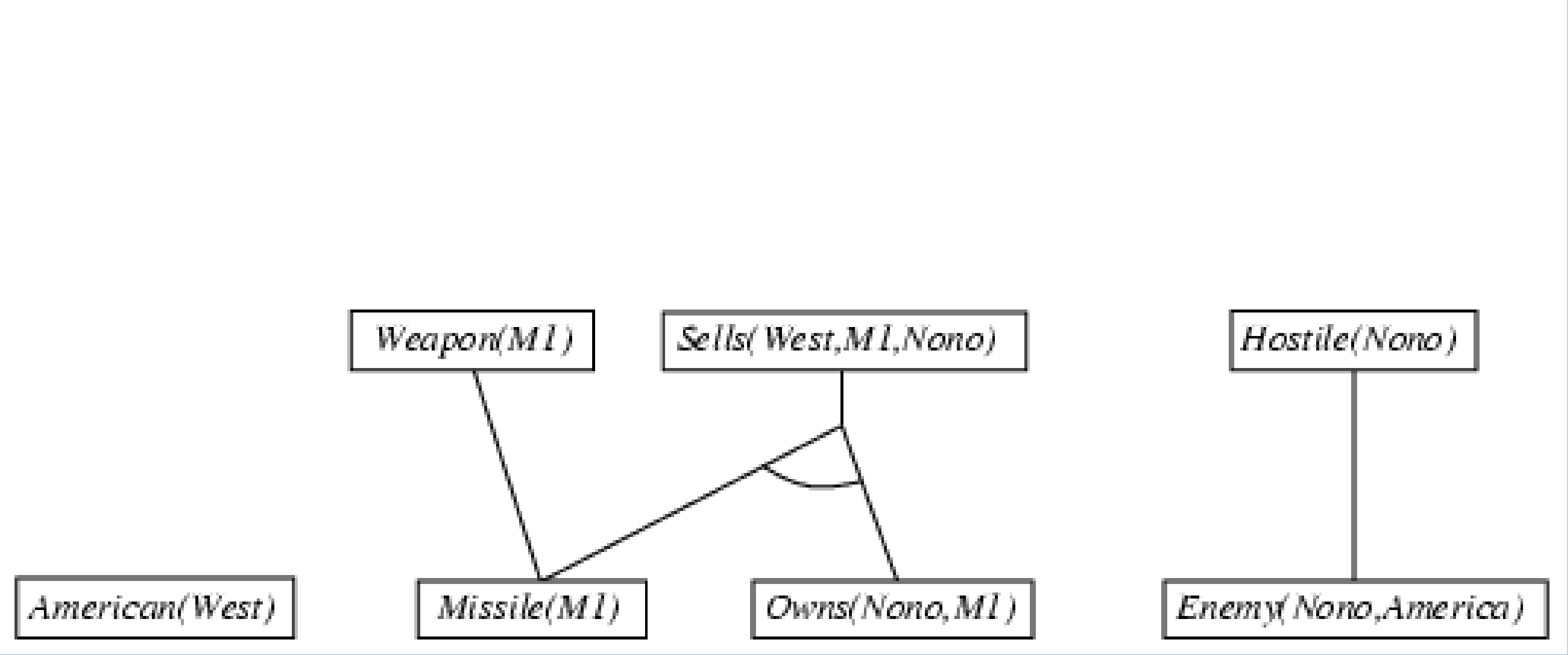
]

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

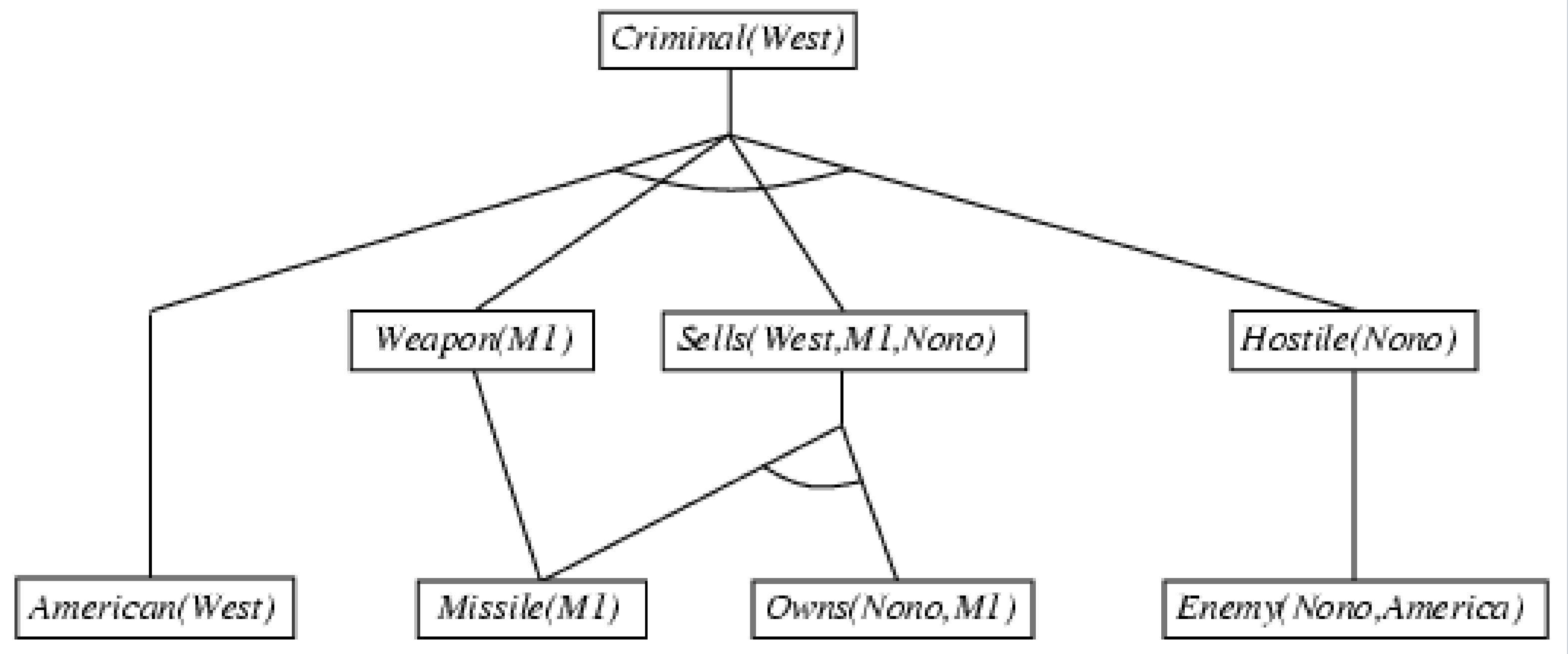
    repeat until *new* is empty
        $new \leftarrow \{\}$
        for each sentence r **in** KB **do**
            $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
            for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
                        for some $p'_1, \ldots, p'_n$ in KB
                $q' \leftarrow$ SUBST(θ, 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', α)
                    if ϕ is not *fail* **then return** ϕ
        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

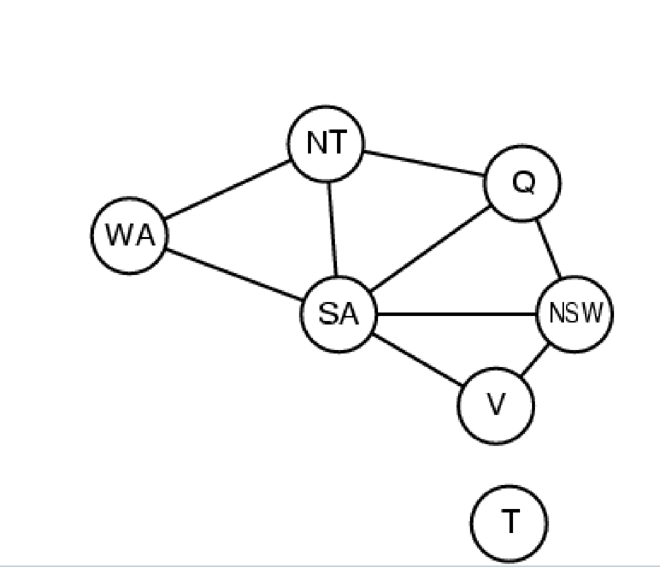
Matching itself can be expensive:

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



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

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

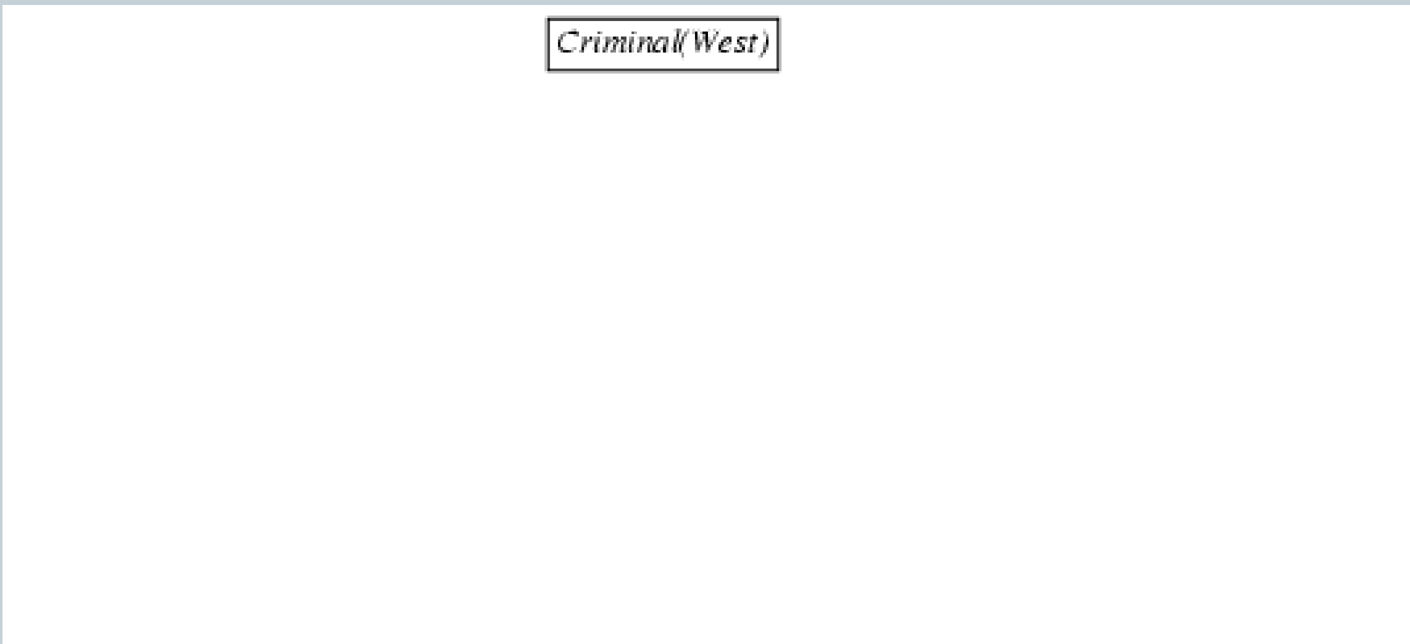
- *Colorable*() is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard
-

Backward chaining algorithm

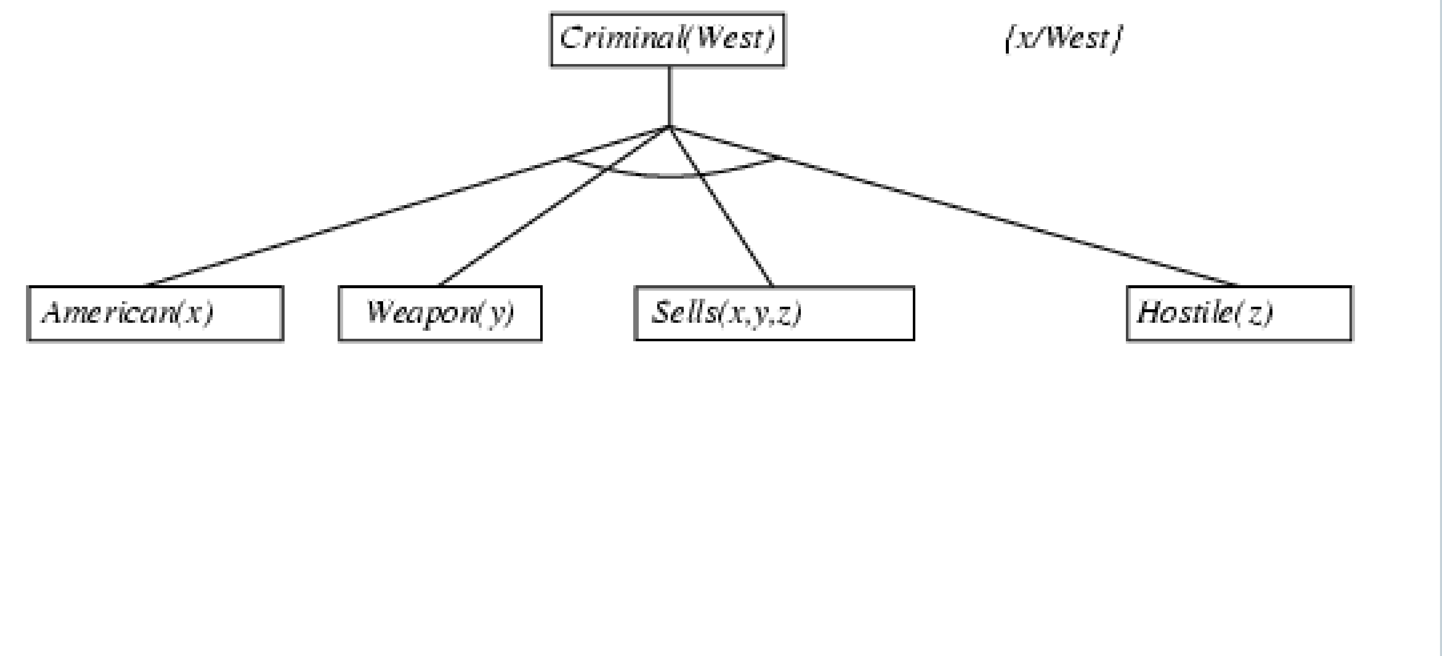
function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
inputs: KB, a knowledge base
$goals$, a list of conjuncts forming a query
θ, the current substitution, initially the empty substitution $\{\ \}$
local variables: ans, a set of substitutions, initially empty

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

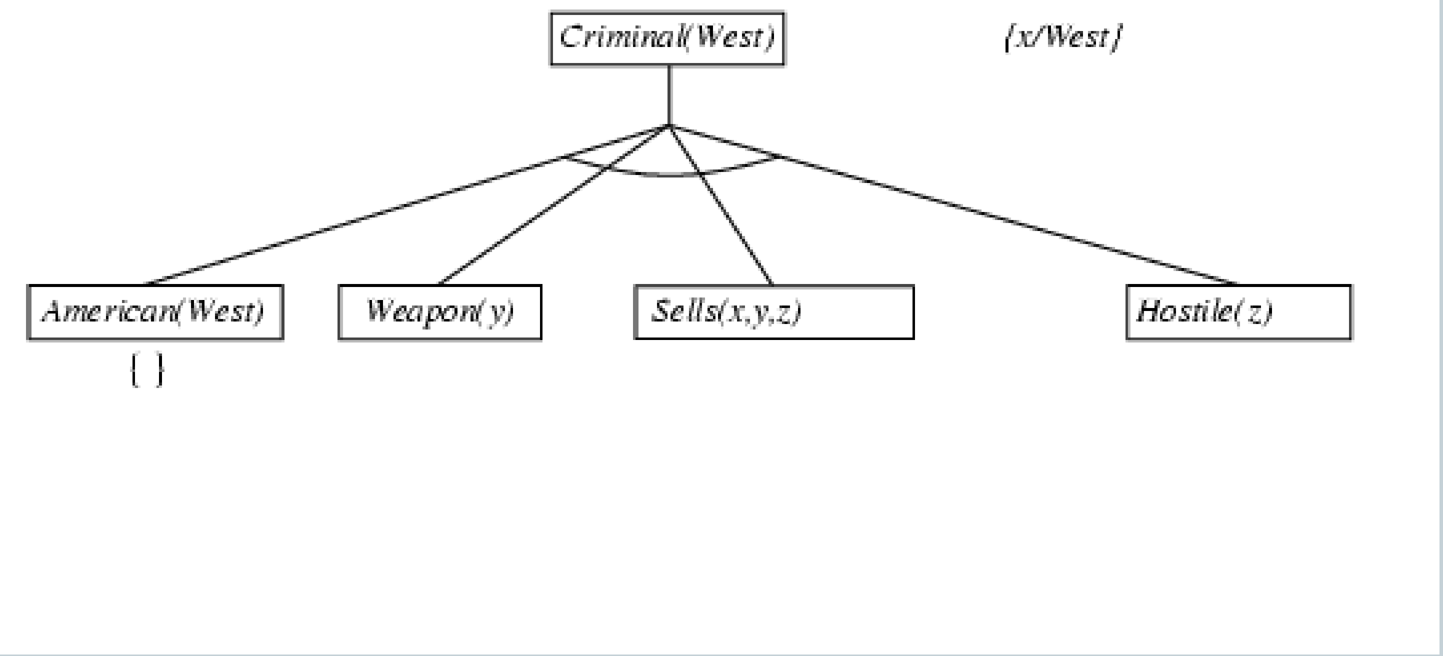
Backward chaining example



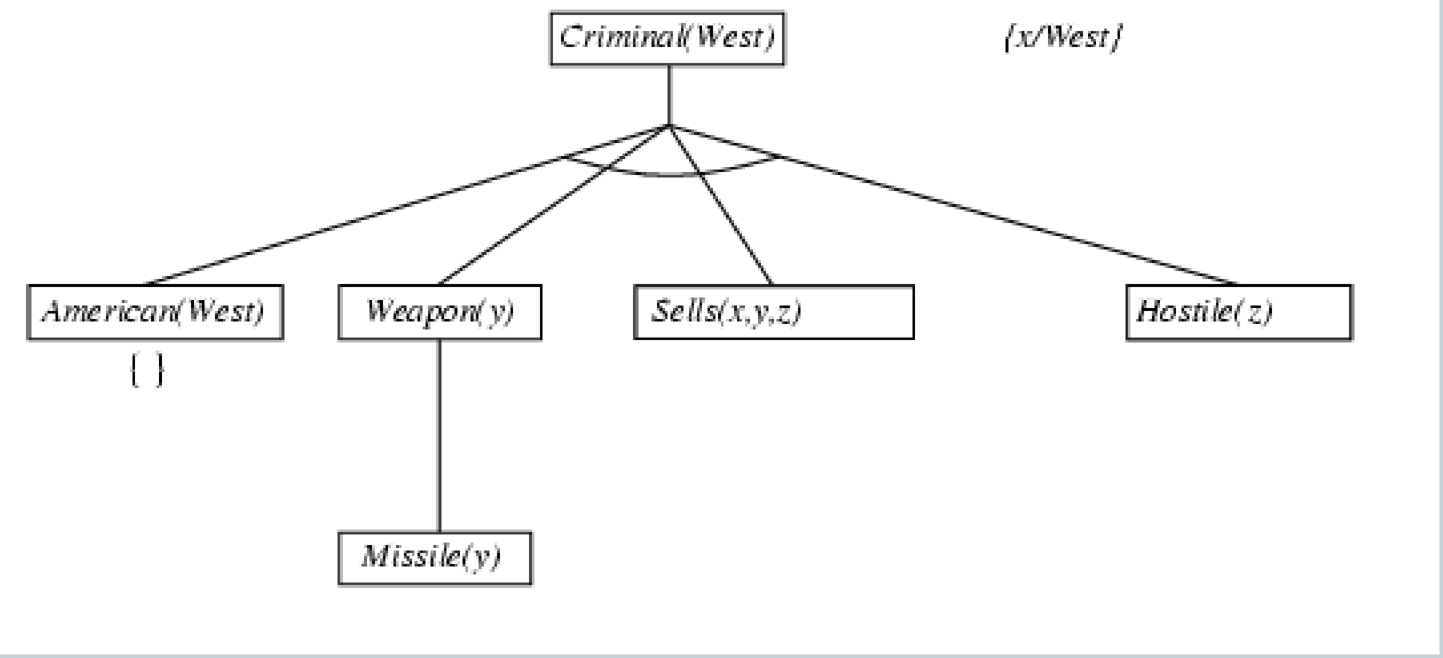
Backward chaining example



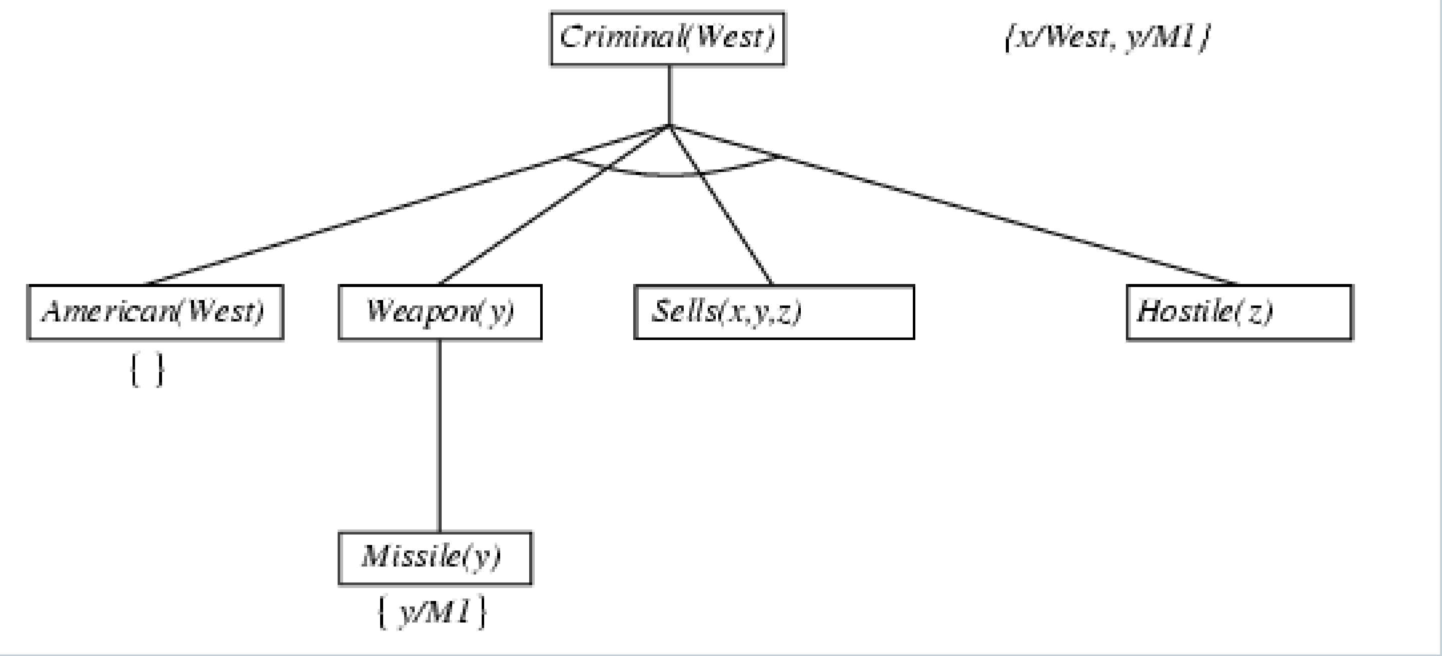
Backward chaining example



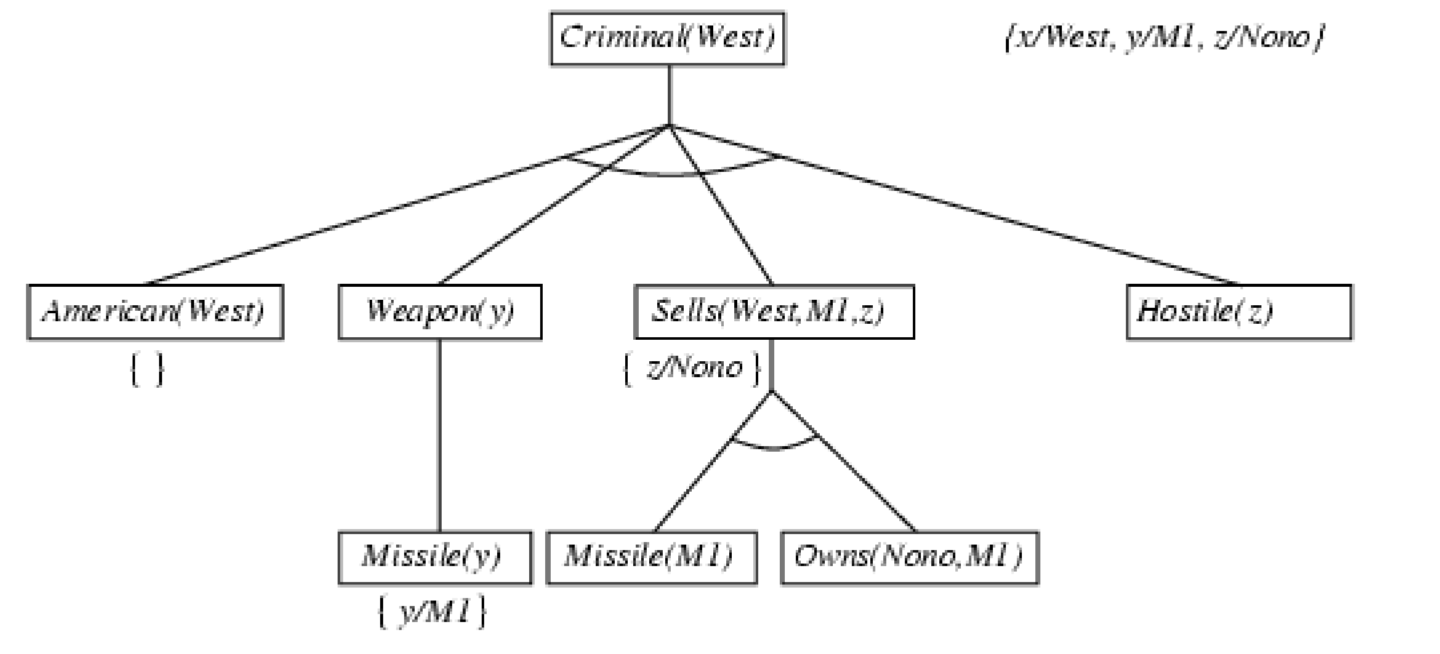
Backward chaining example



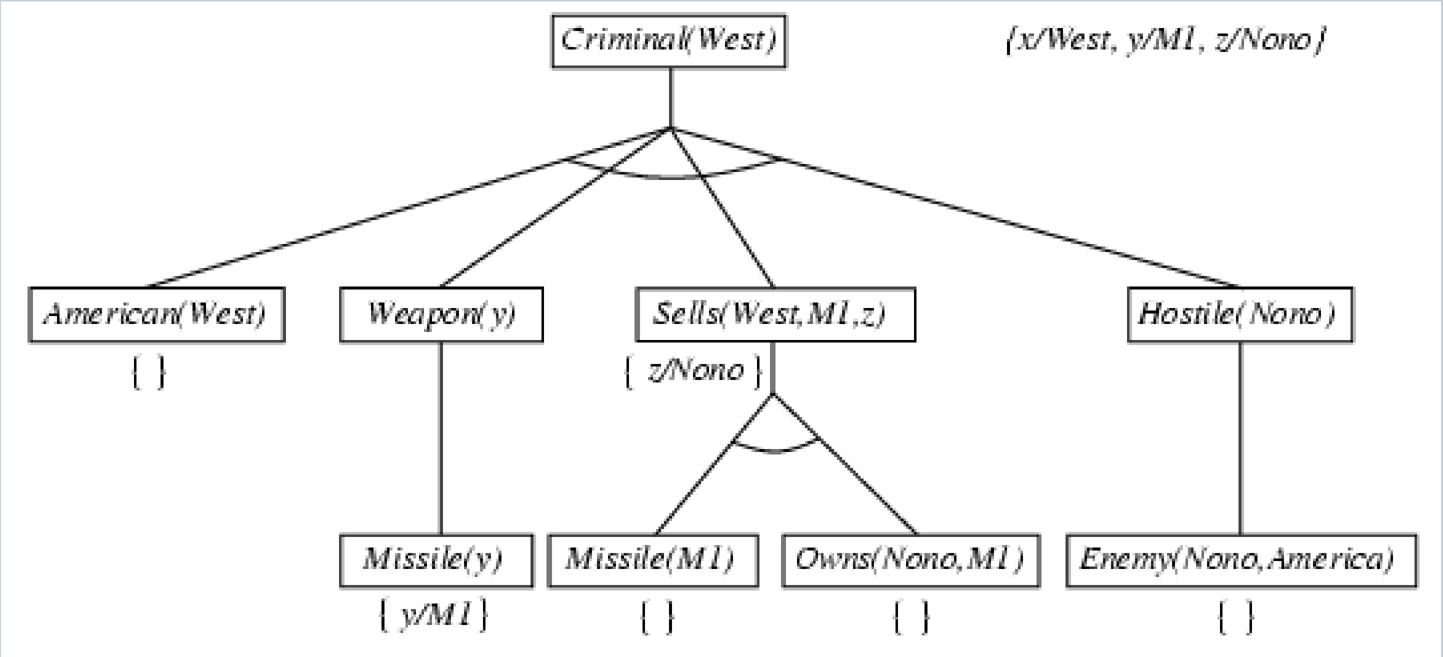
Backward chaining example



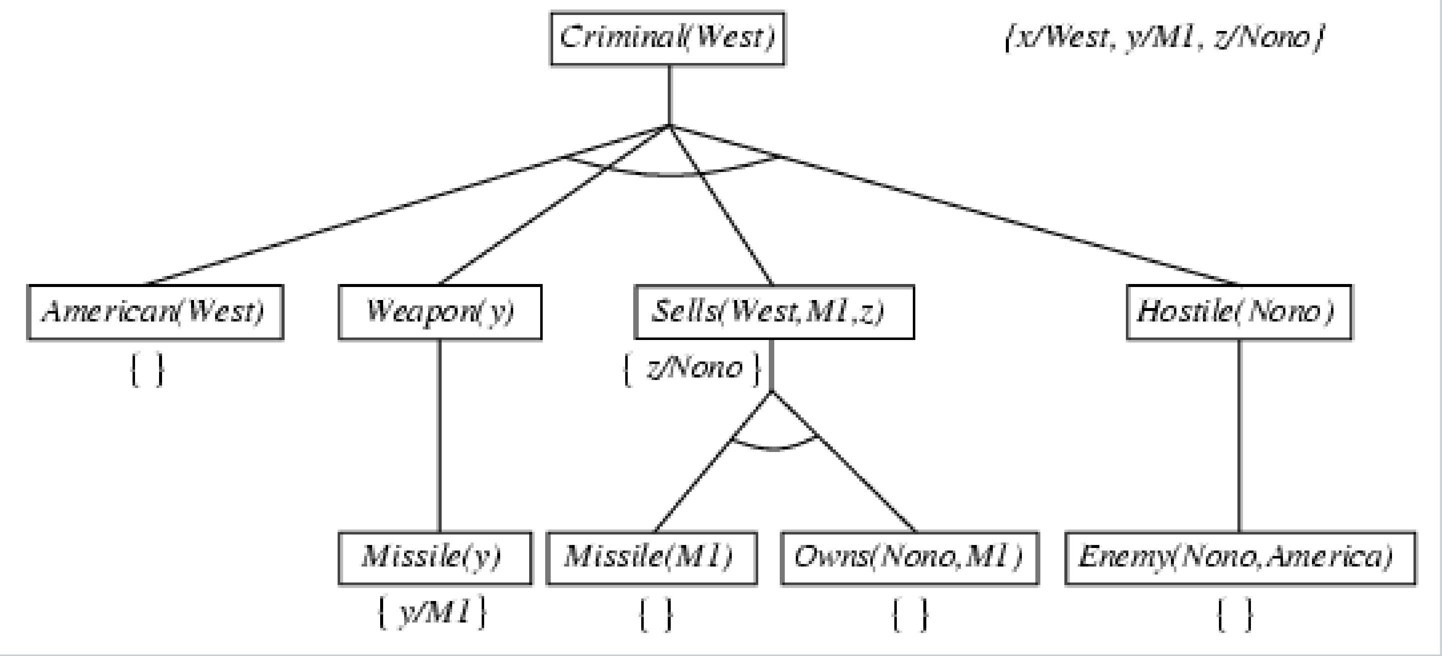
Backward chaining example



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

- Program = set of clauses = `head :- literal`$_1$`, … literal`$_n$.

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

`Query : criminial(west)?`

`Query: criminial(X)?`

- membership
  - member(X,[X|_]).
  - member(X,[_|T]):- member(X,T).
    - ?-member(2,[3,4,5,2,1])
    - ?-member(2,[3,4,5,1])

- subset
  - subset([],L).
  - subset([X|T],L):- member(X,L),subset(T,L).
    - ?- subset([a,b],[a,c,d,b]).

- Nth element of list
  - nth(0,[X|_],X).
  - nth(N,[_|T],R):- nth(N-1,T,R).
    - ?nth(2,[3,4,5,2,1],X)

Prolog

- Appending two lists to produce a third:
  - `append([],Y,Y).`
  - `append([X|L],Y,[X|Z]) :- append(L,Y,Z).`
    - query: `append([1,2],[3],Z) ?`
    - query: `append(A,B,[1,2]) ?`
- answers:

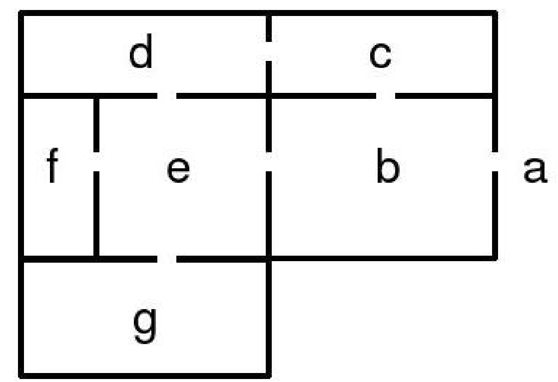
```
A=[]     B=[1,2]
A=[1]    B=[2]
A=[1,2]  B=[]
```

- Path between two nodes in a graph
  - `path(X,Z): link(X,Z)`
  - `path(X,Z): link(Y,Z), path(X,Y)`

```
What happens if?
path(X,Z): path(X,Y), link(X,Z)
path(X,Z): link(X,Z)
```

Searching in a Maze

- Searching for a telephone in a building:



- How do you search without getting lost?
- How do you know that you have searched the whole building?
- What is the shortest path to the telephone?

Searching in a Maze

- go(X,Y,T): Succeeds if one can go from room X to room Y. T contains the list of rooms visited so far.
- Facts in the knowledge base
  - Door(b,c)
  - hasphone(g):

- go(X,X,_).

  go(X,Y,T) :- door(X,Z), not(member(Z,T)), go(Z,Y,[Z|T]).

- go(X,Y,T) :- door(Z,X), not(member(Z,T)), go(Z,Y,[Z|T]).

- go(a,X,[]),hasphone(X) inefficient.
- hasphone(X),go(a,X,[])

Recall: Propositional Resolution-based Inference

We want to prove:

$$\boxed{\begin{array}{l} KB \models \alpha \\ \textit{equivalent to}: KB \wedge \neg\alpha \ \textit{unsatifiable} \end{array}}$$

We first rewrite $KB \wedge \neg\alpha$ into conjunctive normal form (CNF).

A "conjunction of disjunctions"

literals

$$\underbrace{(A \vee \neg B)}_{\text{Clause}} \wedge \underbrace{(B \vee \neg C \vee \neg D)}_{\text{Clause}}$$

- Any KB can be converted into CNF
- k-CNF: exactly k literals per clause

Resolution Examples (Propositional)

$(A \vee B \vee C)$

$(\neg A)$

$- - - - - - - - - - - - -$

$\therefore (B \vee C)$

$(A \vee B \vee C)$

$(\neg A \vee D \vee E)$

$- - - - - - - - - - - -$

$\therefore (B \vee C \vee D \vee E)$

Resolution Algorithm

- The resolution algorithm tries to prove: $KB \models \alpha$ *equivalent to*

  $KB \wedge \neg\alpha$ *unsatisfiable*

- Generate all new sentences from KB and the query.
- One of two things can happen:

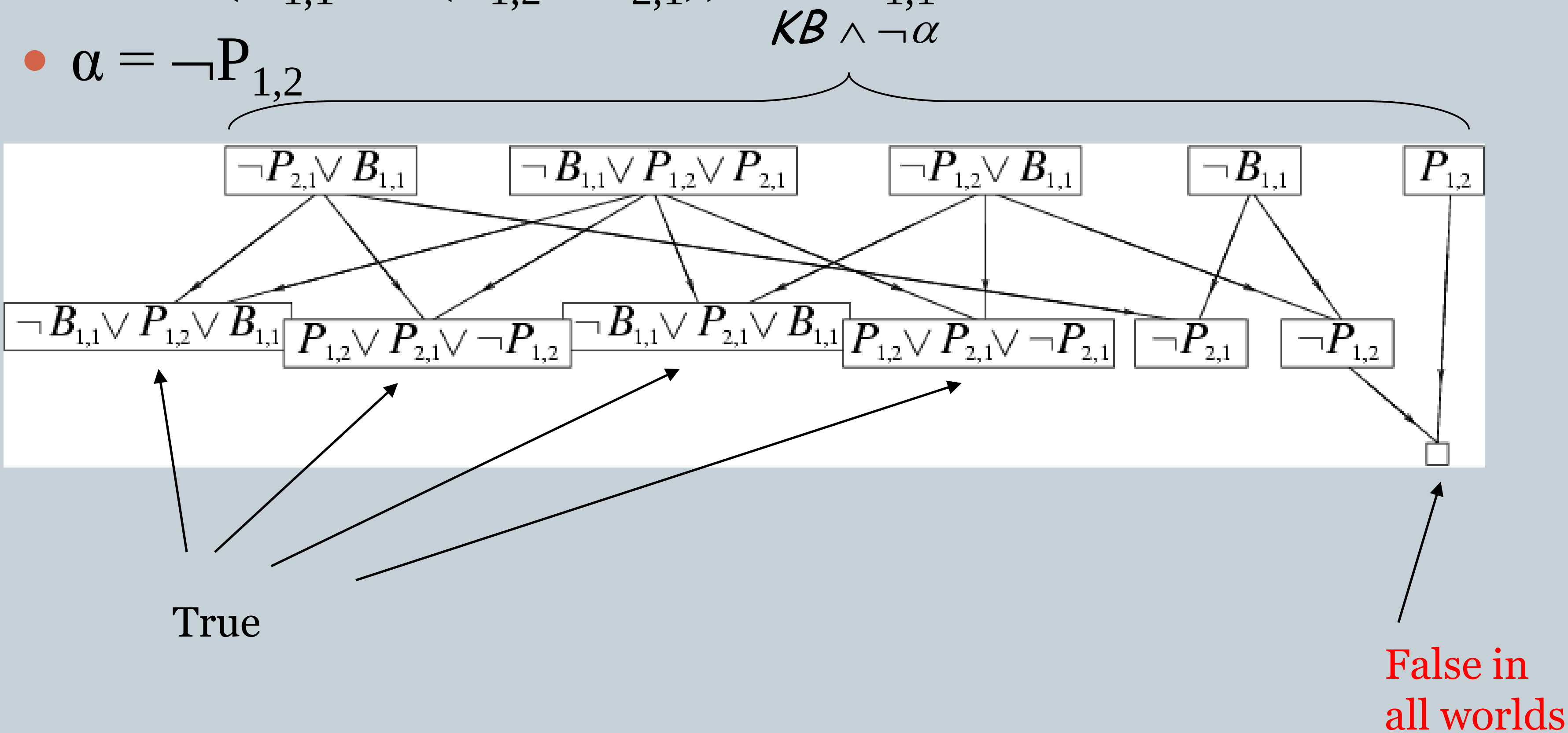
1. We find $P \wedge \neg P$ which is unsatisfiable, i.e. we can entail the query.

2. We find no contradiction: there is a model that satisfies the Sentence (non-trivial) and hence we cannot entail the query.

$KB \wedge \neg\alpha$

Resolution example

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$



Resolution in FOL

- Full first-order version:

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

where `Unify`(ℓ_i, $\neg m_j$) = θ.

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

- For example,

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

with θ = {x/Ken}

- Apply resolution steps to CNF(KB ∧ ¬α); complete for FOL

Converting FOL sentences to CNF

Original sentence:

Anyone who likes all animals is loved by someone:

$\forall x\ [\forall y\ Animal(y) \Rightarrow Likes(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

1. Eliminate biconditionals and implications

$\forall x\ [\neg\forall y\ \neg Animal(y) \vee Likess(x,y)] \vee [\exists y\ Loves(y,x)]$

2. Move $\neg$ inwards:

Recall: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\exists x\ p \equiv \forall x\ \neg p$

$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Likes(x,y))] \vee [\exists y\ Loves(y,x)]$

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

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

Either there is some animal that x doesn't like if that is not the case then someone loves x

Conversion to CNF contd.

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

   $\forall$x [$\exists$y *Animal*(*y*) $\wedge$ $\neg$*Likes*(*x*,*y*)] $\vee$ [$\exists$z *Loves*(*z*,*x*)]

4. Skolemize:

   $\forall$x [*Animal*(*A*) $\wedge$ $\neg$*Likes*(*x*,*A*)] $\vee$ *Loves*(*B*,*x*)

   Everybody fails to love a particular animal A or is loved by a particular person B

   Animal(cat)
   Likes(,arry, cat)
   Loves(john, marry)
   Likes(cathy, cat)
   Loves(Tom, cathy)

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*)

   (reason: animal y could be a different animal for each x.)

Conversion to CNF contd.

5. Drop universal quantifiers:

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

(all remaining variables assumed to be universally quantified)

6. Distribute $\vee$ over $\wedge$:
7.

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

Original sentence is now in CNF form – can apply same ideas to all sentences in KB to convert into CNF

Also need to include negated query Then use resolution to attempt to derive the empty clause which show that the query is entailed by the KB

Recall: Example Knowledge Base in FOL

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile":

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

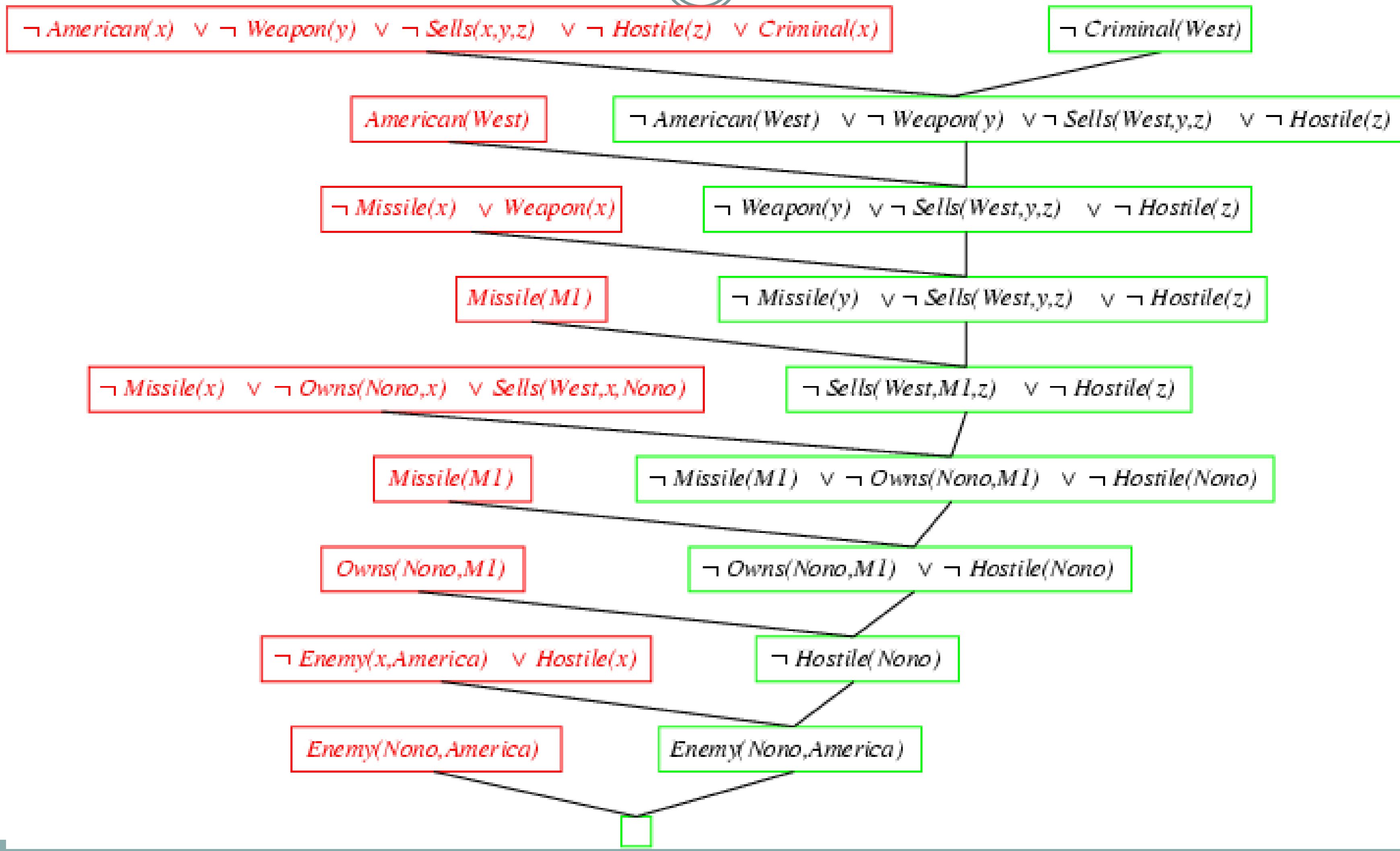
The country Nono, an enemy of America …

Enemy(Nono,America)

Can be converted to CNF

Query: Criminal(West)?

Resolution proof



Second Example

KB:

Everyone who loves all animals is loved by someone Anyone who kills animals is loved by no-one Jack loves all animals. Either Curiosity or Jack killed the cat, who is named Tuna

Query: *Did Curiousity kill the cat?*

Inference Procedure:

Express sentences in FOL

Convert to CNF form and negated query

A. $\forall x\,[\forall y\ \text{Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y\ \text{Loves}(y, x)]$

B. $\forall x\,[\exists y\ \text{Animal}(y) \wedge \text{Kills}(x,y)] \Rightarrow [\forall z\ \neg\text{Loves}(z,x)]$

C. $\forall x\ \text{Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x)$

D. $\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})$

E. $\text{Cat}(\text{Tuna})$

F. $\forall x\ \text{Cat}(x) \Rightarrow \text{Animal}(x)$

¬G. $\neg\text{Kills}(\text{Curiosity}, \text{Tuna})$

A1. $\text{Animal}(F(x)) \lor \text{Loves}(G(x), x)$

A2. $\neg\text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x)$

B. $\neg\text{Animal}(y) \lor \neg\text{Kills}(x,y) \lor \neg\text{Loves}(z,x)]$

C. $\neg\text{Animal}(x) \lor \text{Loves}(\text{Jack}, x)$

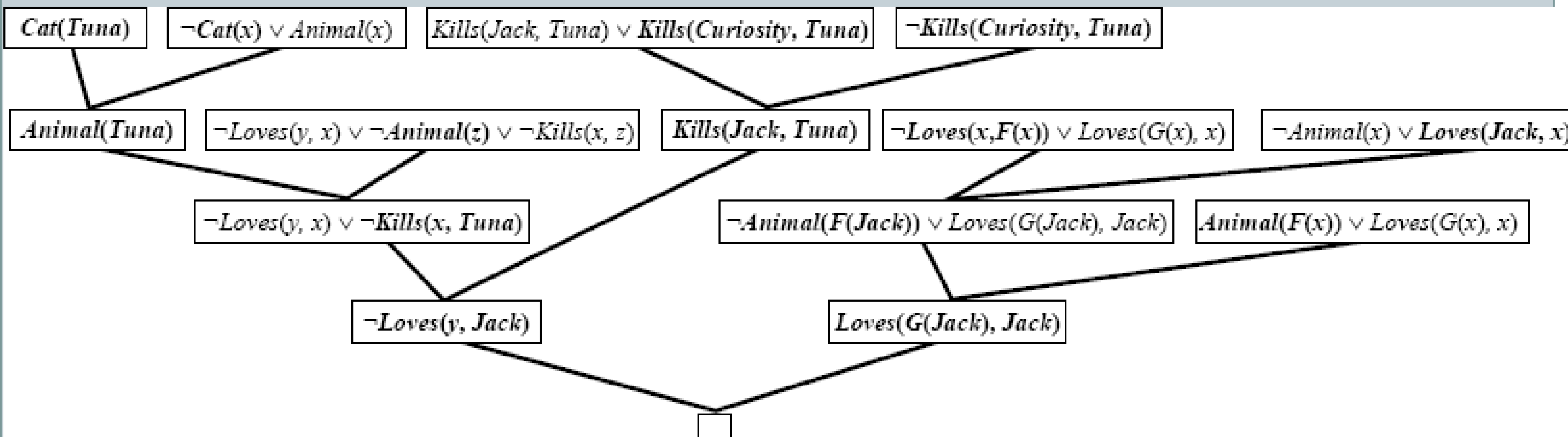
D. $\text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna})$

E. $\text{Cat}(\text{Tuna})$

F. $\neg\text{Cat}(x) \lor \text{Animal}(x)$

¬G. $\neg\text{Kills}(\text{Curiosity}, \text{Tuna})$

Resolution-based Inference



Summary

- Inference in FOL
  - Simple approach: reduce all sentences to PL and apply propositional inference techniques
  - Generally inefficient
- FOL inference techniques
  - Unification
  - Generalized Modus Ponens
    - Forward-chaining: complete with definite clauses
  - Resolution-based inference
    - Refutation-complete
- Read Chapter 9
  - Many other aspects of FOL inference we did not discuss in class
- Homework 4 due on Tuesday