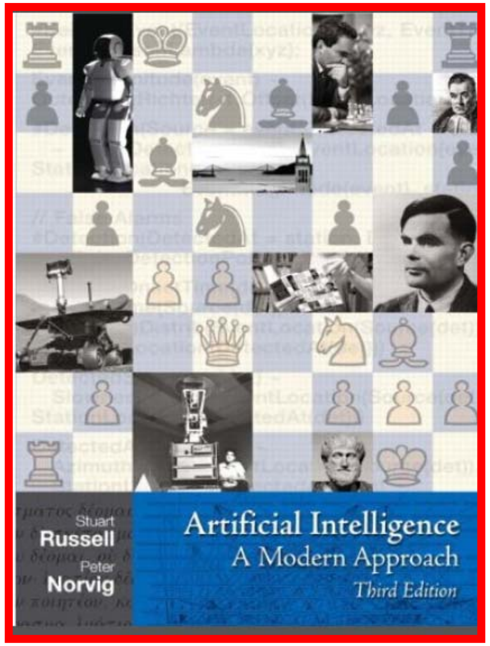


(CHAPTER-9)

FIRST-ORDER LOGIC FOR INFERENCE

Yanmei Zheng

Outline

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

Barber Paradox

- The barber shave all those men, and only those men, who do not shave themselves.
- There is no barber in town.
- *B(x):* x is a barber.
- *H(x,y):* x shave y.
- FOL sentences:

$$(\forall x)(B(x) \Rightarrow (\forall y)(\neg H(y,y) \Rightarrow H(x,y)))$$
$$(\forall x)(B(x) \Rightarrow (\forall y)(H(y,y) \Rightarrow \neg H(x,y)))$$
$$\neg(\exists x)B(x)$$

- **How to prove it using resolution like in PL?**

Inference in First-Order Logic

- **Need to add new logic rules above those in Propositional Logic**
  - **Universal Elimination**

$$\forall x \;\; Likes(x, Semisonic) \Rightarrow Likes(Liz, Semisonic)$$

  - **Existential Elimination**

$$\exists x \;\; Likes(x, Semisonic) \Rightarrow Likes(Person1, Semisonic)$$

(Person1 does not exist elsewhere in KB)

  - **Existential Introduction**

$$Likes(Glenn, Semisonic) \Rightarrow \exists x \;\; Likes(x, Semisonic)$$

Example of inference rules

- "It is illegal for students to copy music."
- "Joe is a student."
- "Every student copies music."
- Is Joe a criminal?

Knowledge Base:

$$\forall x, y \;\; Student(x) \wedge Music(y) \wedge Copies(x, y) \qquad (1)$$
$$\Rightarrow Criminal(x)$$

$$Student(Joe) \qquad (2)$$

$$\forall x \;\; \exists y \;\; Student(x) \wedge Music(y) \wedge Copies(x, y) \qquad (3)$$

Example cont...

$$\text{From}: \forall x \;\; \exists y \;\; Student(x) \wedge Music(y) \wedge Copies(x, y)$$

Universal Elimination

$$\exists y \;\; Student(Joe) \wedge Music(y) \wedge Copies(Joe, y)$$

Existential Elimination

$$Student(Joe) \wedge Music(SomeSong) \wedge Copies(Joe, SomeSong)$$

Modus Ponens

$$Criminal(Joe)$$

How could we build an inference engine?

- Software system to try all inferences to test for *Criminal(Joe)*
  - A very common behavior is to do:
    - And-Introduction
    - Universal Elimination
    - Modus Ponens

Example of this set of inferences

- **Bob is a buffalo**

  Buffalo(Bob) **(1)**

- ***Pat is a pig***

  Pig(Pat) **(2)**

- **Buffaloes outrun pigs**

  $\forall x,y\ Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x,y)$ **(3)**

- **Bob outruns Pat**

  Faster(Bob,Pat)

  $Buffalo(Bob) \wedge Pig(Pat)$ **(1) & (2)** **(4)**

  $Buffalo(Bob) \wedge Pig(Pat) \Rightarrow Faster(Bob,Pat))$ **UE,{x/Bob,y/Pat}** **(5)**

  Faster(Bob,Pat) **(4) &(5)**

- **Generalized Modus Ponens does this in one shot**

Substitution

- $P \lor R$, $Q \lor \neg R$ can infer $P \lor Q$ using resolution in PL.
- $P \lor R(a)$, $Q \lor \neg R(a)$ can infer $P \lor Q$
- Can $P \lor R(x)$, $Q \lor \neg R(y)$ infer $P \lor Q$?
- First, we need substitution

Substitution {9.1.1}

- **FOL is Similar to PL**
  - **Important differences**
    - Quantifiers
    - Variables
- **Important concept: substitution**
  - **Subst(θ, α), θ is like {x/Michael, y/Bob}**
  - **replace variables with terms**

$$(\forall x)(Man(x) \Rightarrow Mortal(x))$$

$$\text{Subst}(\{x/Michael\},(\forall x)(Man(x) \Rightarrow Mortal(x))$$

$$Man(Michael) \Rightarrow Mortal(Michael)$$

Substitution

- A substitution σ in a sentence binds variables to particular values
- Examples:

$$p = Student(x)$$
$$\sigma = \{x / Cheryl\}$$
$$p\sigma = Student(Cheryl)$$

$$q = Student(x) \wedge Lives(y)$$
$$\sigma = \{x / Christopher, y / Goodhue\}$$
$$q\sigma = Student(Christopher) \wedge Lives(Goodhue)$$

Inference rules for quantifiers{9.1.1}

- **Universal Instantiation**
- **Everyone at AI class is smart.**
- $(\forall x)\ (AtAI(x) \Rightarrow Smart(x))$
- **after substitution {x/A}, {x/B}, {x/brother(C)} becomes**
- $AtAI(A) \Rightarrow Smart(A)$
- $AtAI(B) \Rightarrow Smart(B)$
- $AtAI(brother(A)) \Rightarrow Smart(brother(A))$
- **We've replaced the variable with all possible ground terms (terms without variables)**

Universal instantiation (UI)

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

$$\frac{\forall \boldsymbol{v}\ \alpha}{\mathrm{Subst}(\{v/g\}, \alpha)}$$

for any variable *v* and ground term *g*

- **E.g.,** $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ **yields:**

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

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

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

Inference rules for quantifiers{9.1.1}

- **Existential Instantiation**
- **Example: Someone at AI is sleeping in class.**
- $(\exists x)(AtAI(x) \wedge Sleep(x))$
  - **Let's call it *a***
  - **becomes: (AtAI(a)$\wedge$Sleep(a))**

- You can replace the variable with ***a constant symbol* that does not appear elsewhere** in the knowledge base
- The constant symbol is a *Skolem* constant

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:**

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

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

Inference rules for quantifiers{9.1.1}

Existential instantiation

- **consider: Everyone has a mother.**
- $\forall y \; \exists x \; Mother(x,y)$
- **how to instantiate x?**
- $\forall y \; Mother(M_{12},y)$ **correct?**
- M_{12} **a *constant symbol* that does not appear elsewhere in the knowledge base**

Inference rules for quantifiers{9.1.1}

- **Existential instantiation**
  - **consider: Everyone has a mother.**
  - $\forall y \exists x\ Mother(x,y)$
  - **how to instantiate x?**
  - $\forall y\ Mother(m(y),y)$ **correct?**
- **m(y) is a Skolem function**

Inference rules for quantifiers{9.1.1}

- **Instantiate**

$(\forall x_1)\ldots(\forall x_{k-1})(\exists x_k)(Qx_{k+1})\ldots(Qx_n)M(x_1,\ldots,x_k,\ldots,x_n)$

- **Replace x_k with a Skolem function** $f(x_1,\ldots,x_{k-1})$
- $(\forall x_1)\ldots(\forall x_{k-1})(Qx_{k+1})\ldots(Qx_n)M(x_1,\ldots,f(x_1,\ldots,x_{k-1}),\ldots,x_n)$

Substitute more than once for $\forall$?
Substitute more than once for $\exists$?

inference rules for quantifiers{9.1.1}

Only perform substitution once for existential quantifier

- *$\exists x$ Kill (x, Victim)*
- **There exists a *x* who killed *Victim*.**
  - **Someone killed the victim**
  - **Maybe more than one person killed the victim**
  - **Existential quantifier says at least one person was killer**
- **Replacement is**
  - *Kill (Murderer, Victim)*

Reduction to propositional inference

- **Suppose the KB contains just the following:**
  - $\forall x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$
  - *King(John)*
  - *Greedy(John)*
  - *Brother(Richard,John)*
- **Instantiating the universal sentence in all possible ways, we have:**
  - $King(John) \land Greedy(John) \Rightarrow Evil(John)$
  - $King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$
  - *King(John)*
  - *Greedy(John)*
  - *Brother(Richard,John)*
- **The new KB is propositionalized: proposition symbols are**
  - *King(John), Greedy(John), Evil(John),* **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)))*

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

E.g., from:

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

$King(John)$

$Greedy(John)$

- 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.

A first-order inference rule

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

- **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**

- **Represent the above inference process as a single inference rule—Generalized Modus Ponens (GMP)**

Dr.Yanmei Zheng

Generalized Modus Ponens

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{subst(q,\vartheta)}$$ **where subst(p_i', θ)= subst(p_i, θ) for all *i***

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

p_1' is *King(John)*	p_1 is *King(x)*
p_2' is *Greedy(y)*	p_2 is *Greedy(x)*
	q is *Evil(x)*

Substitution

θ is *{x/John,y/John}*

qθ is *Evil(John)* **or SUBST(θ,q) is** *Evil(John)*

Dr.Yanmei Zheng

Generalized Modus Ponens

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

where $p_i'\sigma = p_i\sigma$ for all i

$$\begin{aligned}
\text{E.g. } p_1' =\; & \text{Faster(Bob,Pat)} \\
p_2' =\; & \text{Faster(Pat,Steve)} \\
p_1 \wedge p_2 \;\Rightarrow\; q \;=\; & Faster(x,y) \wedge Faster(y,z) \;\Rightarrow\; Faster(x,z) \\
\sigma =\; & \{x/Bob, y/Pat, z/Steve\} \\
q\sigma =\; & Faster(Bob, Steve)
\end{aligned}$$

Generalized Modus Ponens

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

GMP is called a lifted version of Modus Ponens

Substitution

- $P \lor R$, $Q \lor \neg R$ can infer $P \lor Q$ using resolution in PL.
- $P \lor R(a)$, $Q \lor \neg R(a)$ can infer $P \lor Q$
- Can $P \lor R(x)$, $Q \lor \neg R(y)$ infer $P \lor Q$?
- First, we need substitution
- What is the substitution for this problem?

Substitution

- $P \vee R$, $Q \vee \neg R$ can infer $P \vee Q$ using resolution in PL.
- $P \vee R(a)$, $Q \vee \neg R(a)$ can infer $P \vee Q$
- Can $P \vee R(x)$, $Q \vee \neg R(y)$ infer $P \vee Q$?
- First, we need substitution
- What is the substitution for this problem?
- We need a unifier θ where SUBST(θ,R(x))=SUBST(θ,R(y))
- A unifier is a substitution

Unifier {9.2.2}

- Unification is the process of finding substitutions
- *UNIFY* takes two sentences and returns a unifier if one exists
- UNIFY(p,q)=θ where SUBST(θ,p)=SUBST(θ,q)

Unifier {9.2.2}

- We can get the inference immediately if we can find a substitution.
- UNIFY(p,q)=θ where SUBST(θ,p)=SUBST(θ,q)

p	**q**	θ
P(x)	*P(a)*	*{x/a}*
P(f(x), y, g(y))	*P(f(x), x, g(x))*	*{x/y}*
P(f(x), z, z)	*P(f(x), g(a, y),g(a, y))*	*{z/g(a, y)}*

Unifier {9.2.2}

- Lifted inference rules require finding substitutions that make different logical expressions look identical. This process is called unification.
- Unify(α,β) = θ if subst(α, θ)= subst(β,θ)
- substitution σ unifies sentences p and q if $p\sigma = q\sigma$.

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

Unifier {9.2.2}

p	q	θ
Knows(John,x)	*Knows(John,Jane)*	*{x/Jane}*
Knows(John,x)	*Knows(y,Phil)*	*{x/Phil,y/John}*
Knows(John,x)	*Knows(y,Mother(y))*	*{y/John, x/Mother(John)}*
Knows(John,x)	*Knows(x,OJ)*	*{fail}*

- **Use unification in drawing inferences: unify premises of rule with known facts, then apply to conclusion**
  - **If we know** *q*, **and** *Knows(John,x)* → *Likes(John,x)*
  - **Conclude**
    - *Likes(John, Jane)*
    - *Likes(John, Phil)*
    - *Likes(John, Mother(John))*

Unifier {9.2.2}

Multiple unifiers are possible

- **To unify** *Knows(John,x)* **and** *Knows(y,z)*,
- **θ = {y/John, x/z }** **or** **θ = {y/John, x/John, z/John}**
- *Knows (John, z)* **or** *Knows (John, 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 }*

Unifier {9.2.2}

- **Consider the sentence**
- *UNIFY(Knows(John, x),Knows(x, Elizabeth))*
- **=_______________________**

Unifier {9.2.2}

- **Consider the sentence**
- *UNIFY(Knows(John, x),Knows(x, Elizabeth))*
- = **Fail**
  - **This fails because *x* cannot take on two values**
  - **But “Everyone knows Elizabeth” and it should not fail**
- **How?**

Unifier {9.2.2}

- **Consider the sentence**
- *UNIFY(Knows(John, x),Knows(x, Elizabeth))*
- = **Fail**
  - **This fails because *x* cannot take on two values**
  - **But “Everyone knows Elizabeth” and it should not fail**
- **Must standardize apart one of the two sentences to eliminate reuse of variable**
- *UNIFY (Knows(John,x), Knows(x1,Elizabeth))*

Unification

$$
\begin{array}{l}
P[z, f(w), B] \\
P[x, f(A), B] \\
P[g(z), f(A), B] \\
P[C, f(A), B]
\end{array}
\Leftarrow P[x, f(y), B]
\qquad
\begin{array}{l}
s_1 = \{z/x, w/y\} \\
s_2 = \{y/A\} \\
s_3 = \{x/g(z), y/A\} \\
s_4 = \{x/C, y/A\}
\end{array}
$$

Algorithm for finding MGU

```
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 θ = 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
```

In a compound expression such as *F(A,B),*
the *Op* field picks out the function symbol *F*
and the *ARGS* field picks out the argument *list(A,B)*

Algorithm (contd)

function UNIFY-VAR(var, x, θ) **returns** a substitution
  inputs: var, a variable
    x, any expression
    θ, the substitution built up so far

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

OCCUR-CHECK: when matching a variable against a complex term, one must check whether the variable itself occurs inside the term

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 Colonel West is a criminal

Example knowledge base contd.

- ... 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,M1)* **and** *Missile(M1)*
- … 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

- Start with the data (facts) and draw conclusions

- When a new fact ***p*** is added to the KB:
  - For each rule such that ***p*** unifies with a premise
    - if the other premises are known
      - add the conclusion to the KB and continue chaining

Forward chaining algorithm

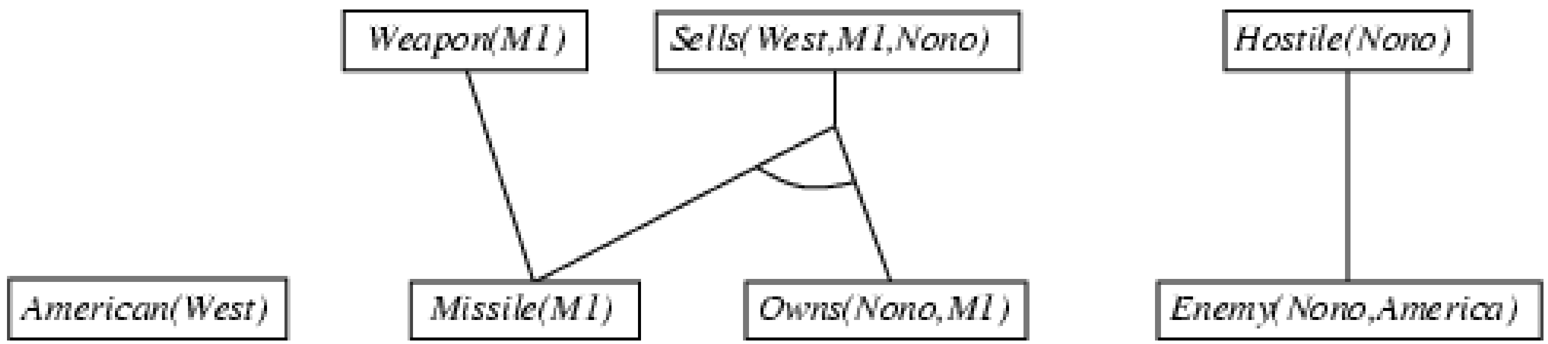
function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
$new \leftarrow \{\}$
for each sentence r **in** KB **do**
$(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
for each θ such that $(p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta$
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*

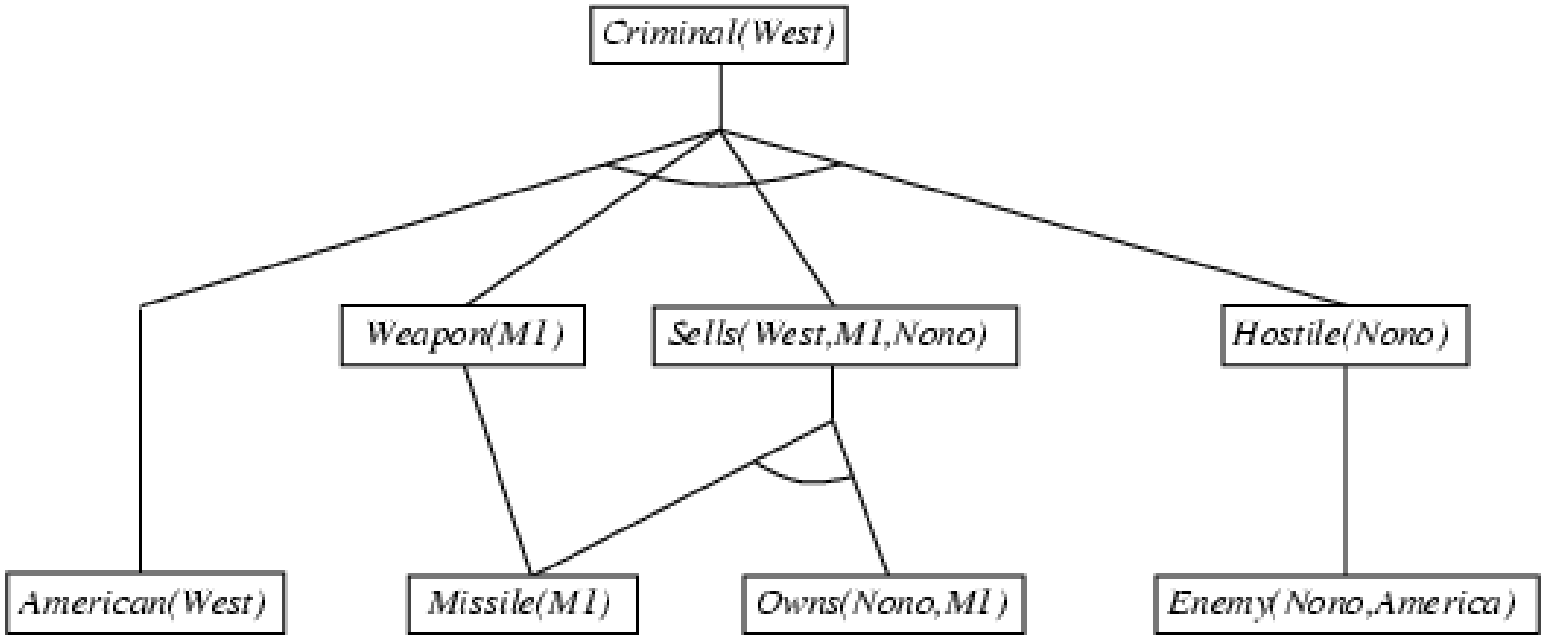
Forward chaining proof

$\boxed{American(West)}$ $\boxed{Missile(M1)}$ $\boxed{Owns(Nono,M1)}$ $\boxed{Enemy(Nono,America)}$

Forward chaining proof



Forward chaining proof



FOL-FC-ASK

Sound and complete for first-order definite clauses

Efficiency of forward chaining

Matching rules against Known facts

- We can remind ourselves that most rules in real-world knowledge bases are small and simple, conjunct ordering
- We can consider subclasses of rules for which matching is efficient, most constrained variable
- We can work hard to eliminate redundant rule matching attempts in the forward chaining algorithm, which is the subject of the next section

Efficiency of forward chaining

- The "inner loop" of the algorithm involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in the knowledge base(pattern matching) and can be very expensive.
- The algorithm rechecks every rule on every iteration to see whether its premises are satisfied, even if very few additions are made to the knowledge base on each iteration.
- The algorithm might generate many facts that are irrelevant to the goal.

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

Avoid irrelevant conclusions

- Backward chaining
- Restrict forward chaining to a selected subset of rules.
- In the deductive database, rewrite the rule set, using information from the goal, so that only relevant variable bindings-those belonging to a so-called magic set-are considered during forward inference.

Backward Chaining

- Start with the query, and try to find facts to support it
- When a query *q* is asked:
  - If a matching fact *q'* is known, return unifier
  - For each rule whose consequent *q'* matches *q*
  - Attempt to prove each premise of the rule by backward chaining
- Prolog does backward chaining

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

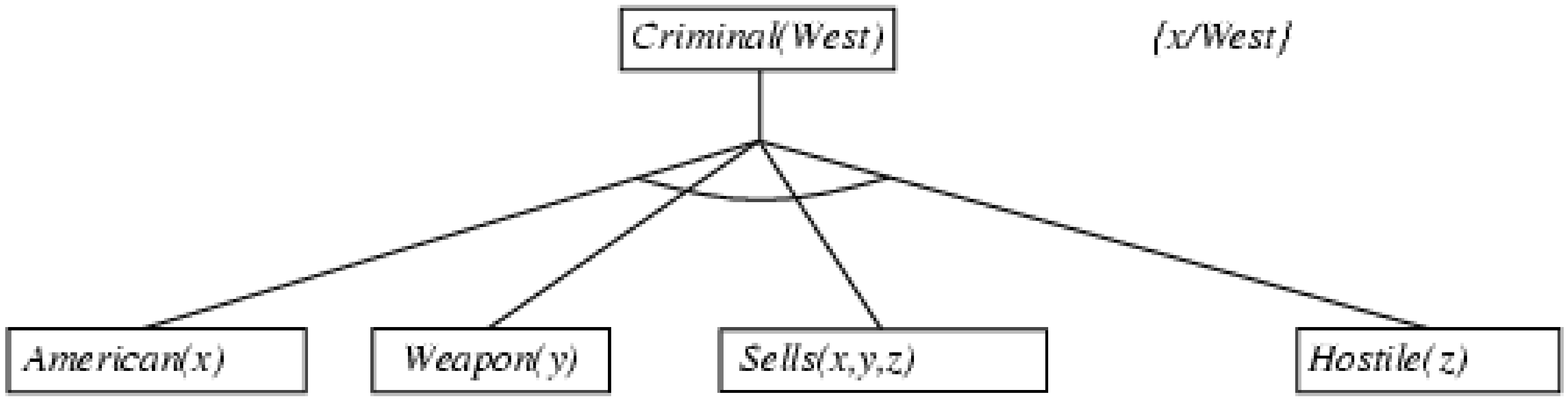
  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow \mathrm{SUBST}(\theta, \mathrm{FIRST}(goals))$
  for each r **in** KB where $\mathrm{STANDARDIZE\text{-}APART}(r) = (\,p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow \mathrm{UNIFY}(q, q')$ succeeds
   $ans \leftarrow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

SUBST(COMPOSE(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))

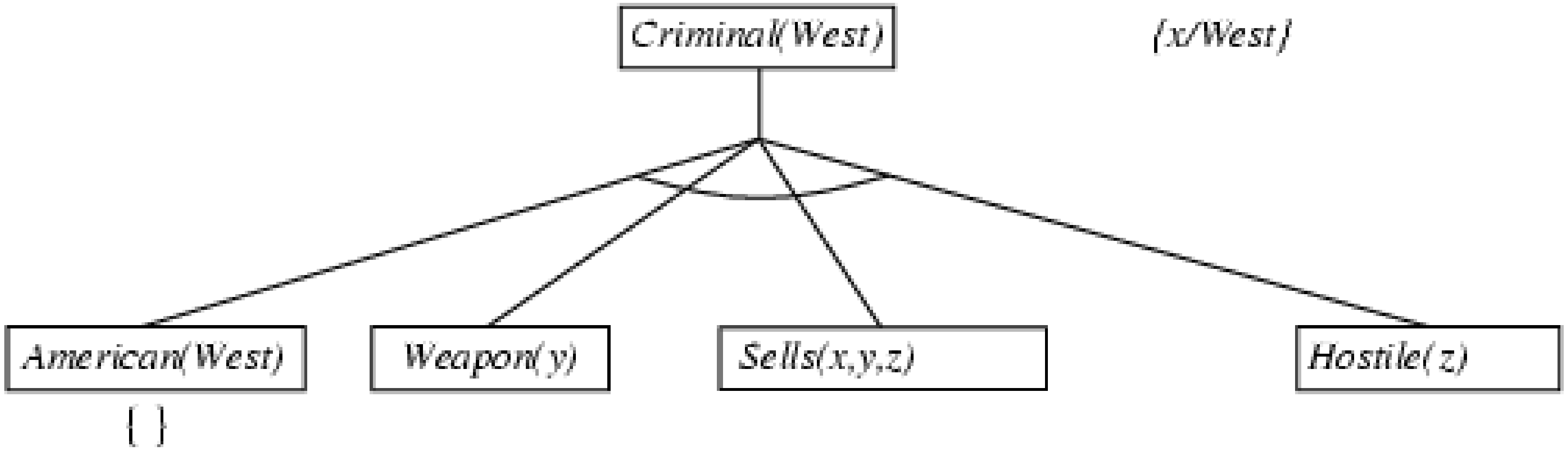
Backward chaining example

$\boxed{Criminal(West)}$

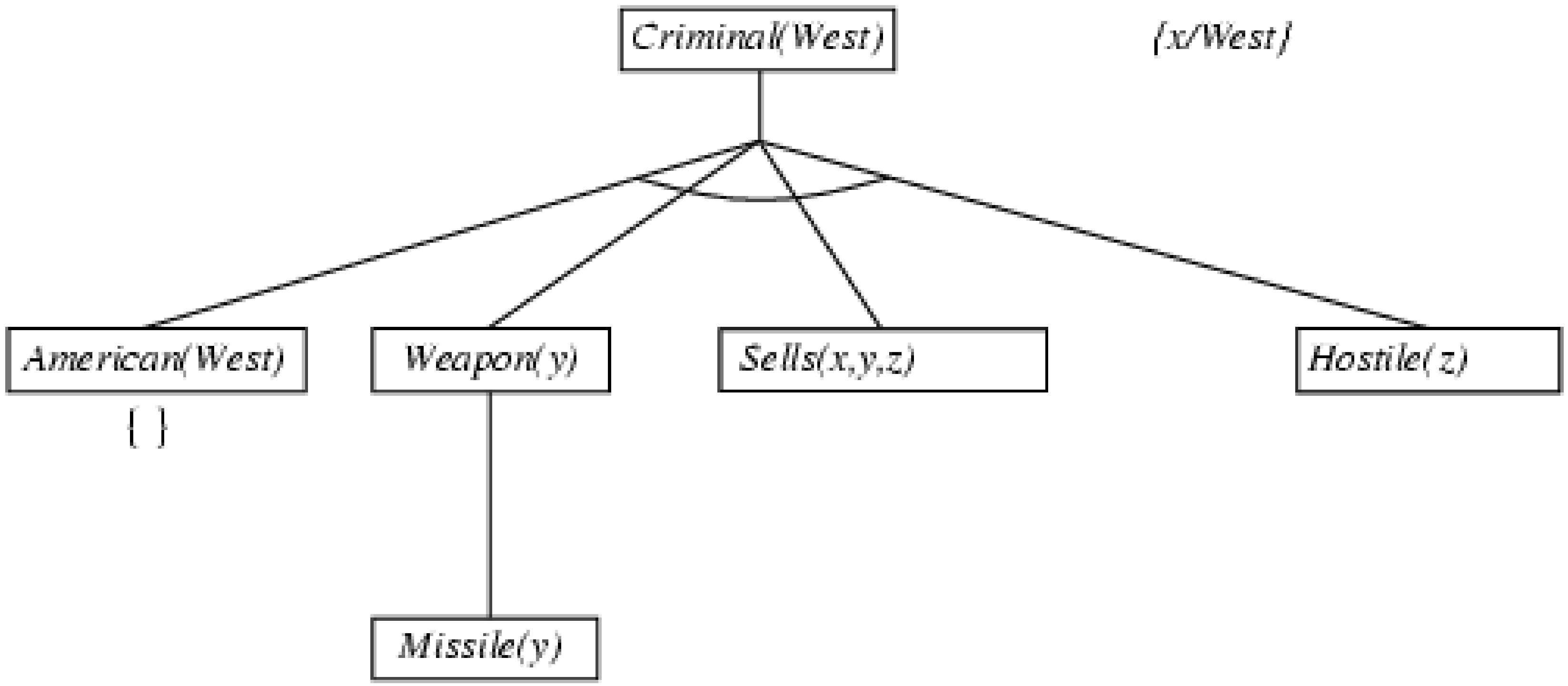
Backward chaining example



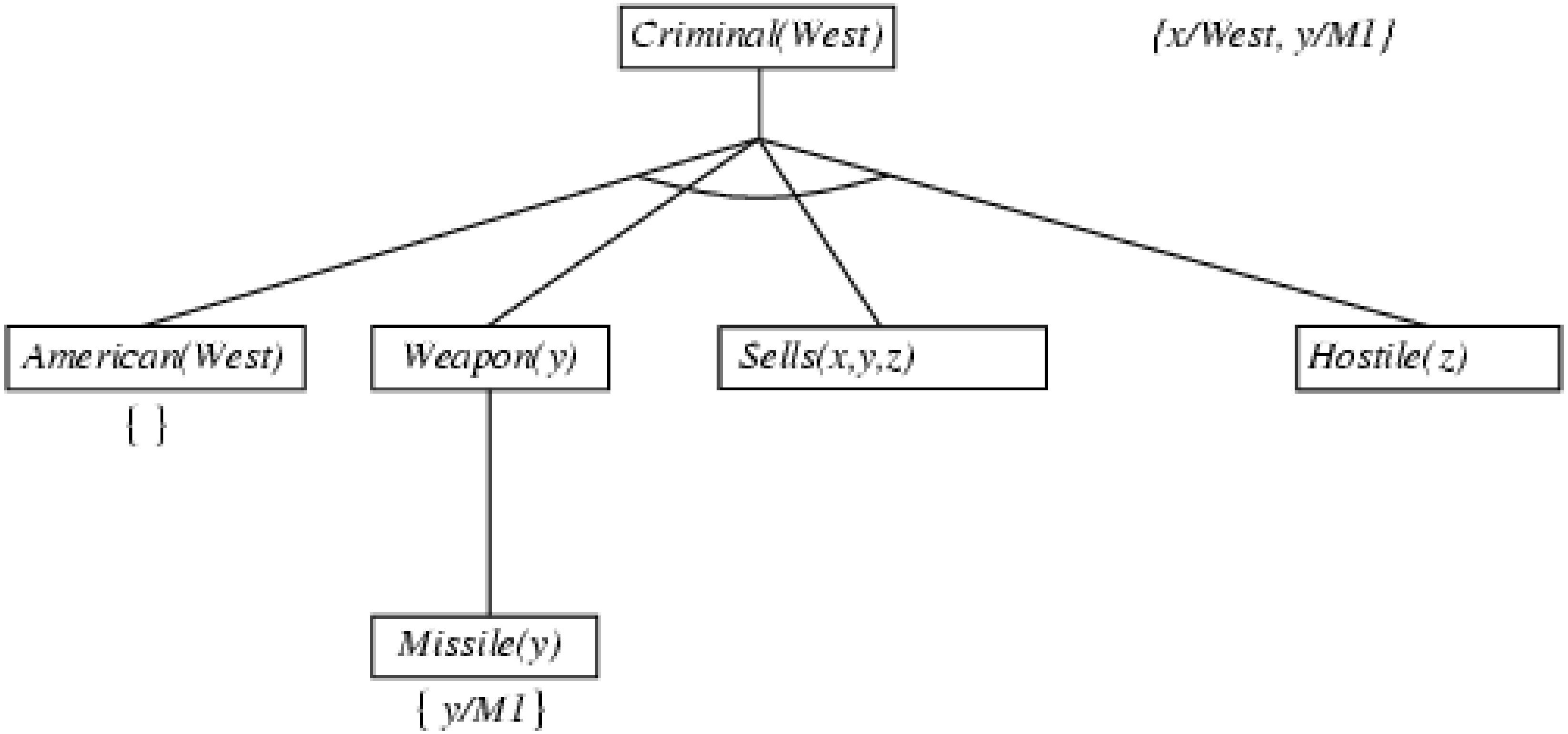
Backward chaining example



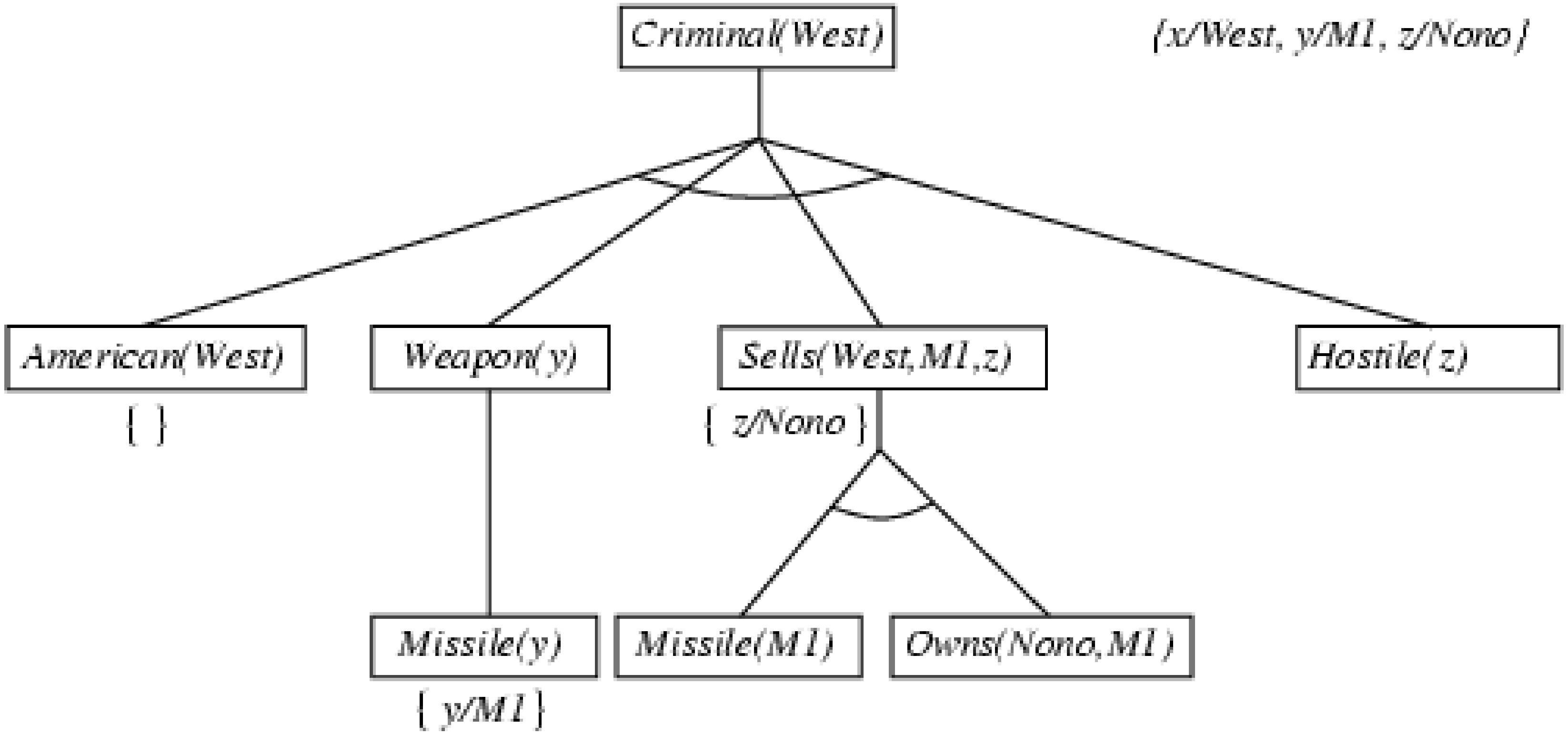
Backward chaining example



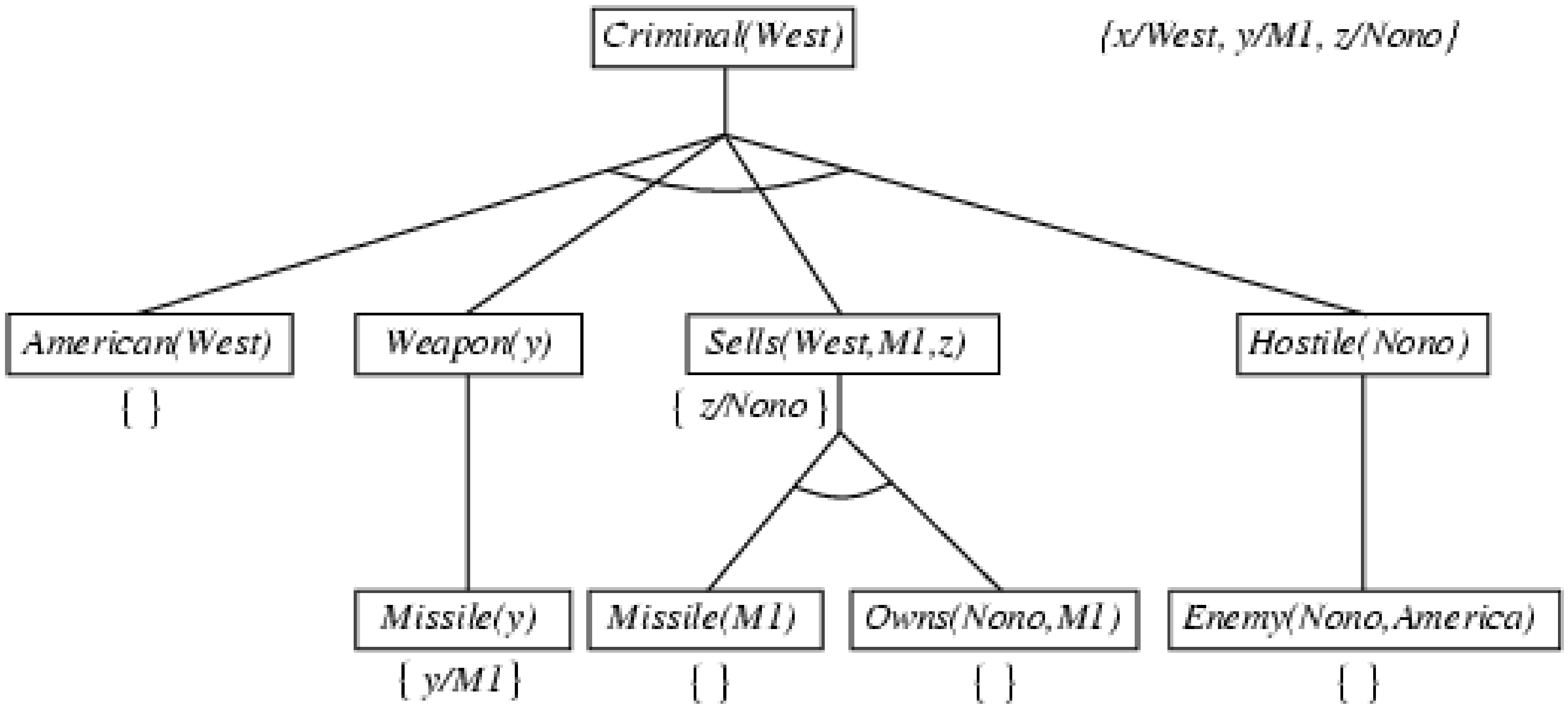
Backward chaining example



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
  - ⇒ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - ⇒ fix using caching of previous results (extra space)
- Widely used for logic programming

Convert to CNF {9.5.1}

- Essentially same as propositional logic
  - Eliminate implications
  - Move negation inwards
  - Standardize variables
  - Skolemize
  - Drop universal quantifiers
  - Distribute $\vee$ over $\wedge$
- Split Conjunctions into clauses
- Try:

$$(\forall x)\{P(x) \Rightarrow [(\forall y)[P(y) \Rightarrow P(f(x,y))] \wedge \neg(\forall y)[Q(x,y) \Rightarrow P(y)]]\}$$

Convert to CNF {9.5.1}

$$(\forall x)\{P(x) \Rightarrow [(\forall y)[P(y) \Rightarrow P(f(x,y))] \wedge \neg(\forall y)[Q(x,y) \Rightarrow P(y)]]\}$$

$$(\forall x)\{\neg P(x) \vee [(\forall y)[\neg P(y) \vee P(f(x,y))] \wedge \neg(\forall y)[\neg Q(x,y) \vee P(y)]]\}$$

$$(\forall x)\{\neg P(x) \vee [(\forall y)[\neg P(y) \vee P(f(x,y))] \wedge (\exists y)[Q(x,y) \wedge \neg P(y)]]\}$$

$$(\forall x)\{\neg P(x) \vee [(\forall y)[\neg P(y) \vee P(f(x,y))] \wedge (\exists z)[Q(x,z) \wedge \neg P(z)]]\}$$

$$(\forall x)\{\neg P(x) \vee [(\forall y)[\neg P(y) \vee P(f(x,y))] \wedge [Q(x,g(x)) \wedge \neg P(g(x))]]\}$$

$$\neg P(x) \vee [[\neg P(y) \vee P(f(x,y))] \wedge [Q(x,g(x)) \wedge \neg P(g(x))]]$$

$$[\neg P(x) \vee \neg P(y) \vee P(f(x,y))] \wedge [\neg P(x) \vee Q(x,g(x))] \wedge [\neg P(x) \vee \neg P(g(x))]$$

CNF

SPLIT CNF into clauses:

$$\neg P(x) \vee \neg P(y) \vee P(f(x,y))$$

$$\neg P(x) \vee Q(x,g(x))$$

$$\neg P(x) \vee \neg P(g(x))$$

Dr.Yanmei Zheng

Resolution: FOL version {9.5.2}

- In propositional logic:
- **Resolution** inference rule (for CNF):

$$\frac{l_1 \vee \ldots \vee l_k, \qquad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where l_i and m_j are complementary literals.

- Resolution Refutation
  - Apply resolution steps to CNF(KB $\wedge$ $\neg\alpha$) and infer contradiction (empty clause)

Resolution: brief summary

- **Full first-order version:**

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_2 \vee \cdots \vee l_k \vee m_2 \vee \cdots \vee m_n)\theta}$$

  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 $\wedge \neg\alpha$);** and infer contradiction (empty clause)

Conversion to CNF

- **Everyone who loves all animals is loved by someone:**

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

- **Eliminate biconditionals and implications**

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

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

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

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

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

Conversion to CNF contd.

- **Standardize variables: each quantifier should use a different one**

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

- **Skolemize: a more general form of existential instantiation.**

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

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

- **Drop universal quantifiers:**

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

- **Distribute $\vee$ over $\wedge$:**

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

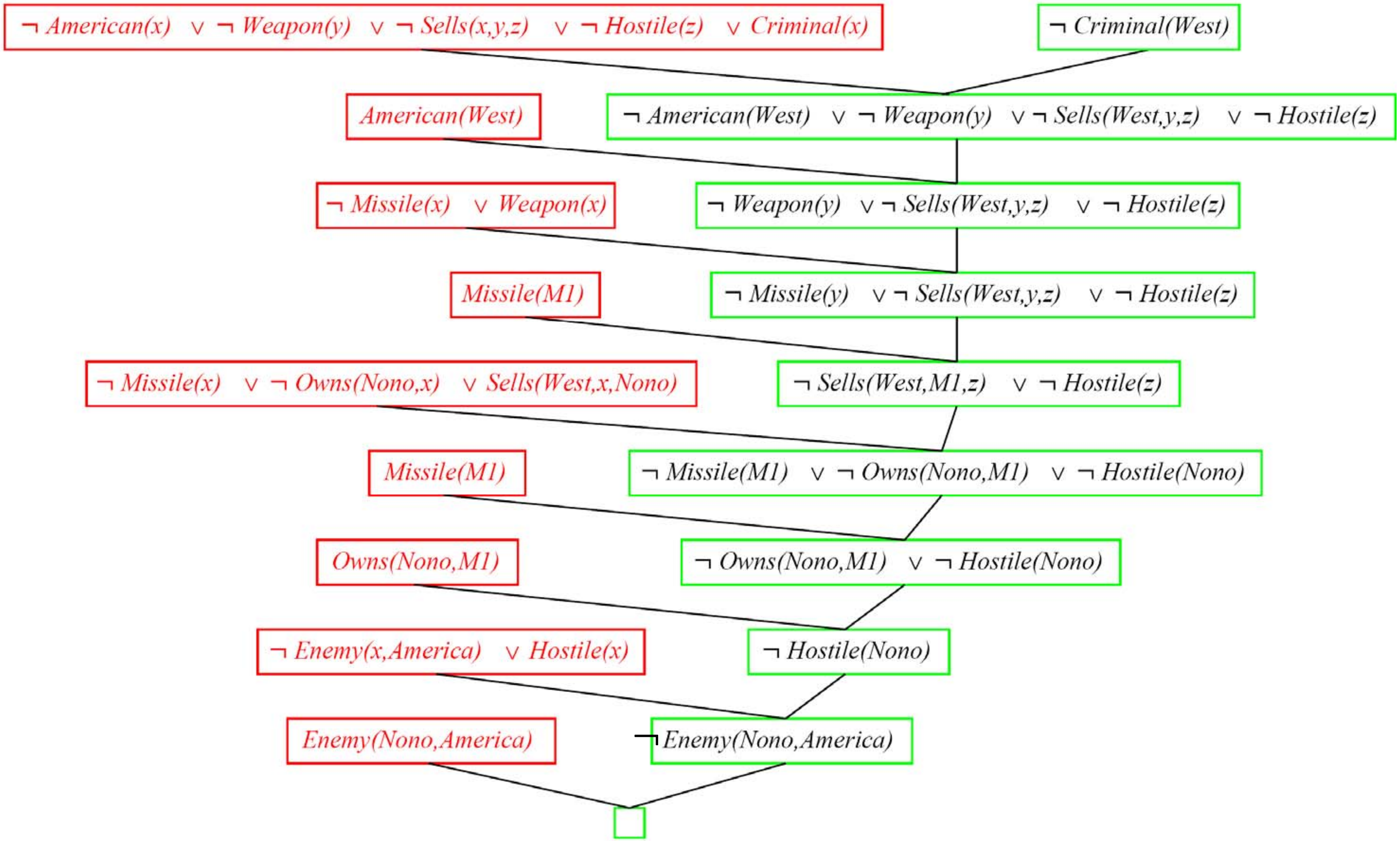
Example knowledge base contd.

- ... 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,M1) and Missile(M1)*
- … 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)*

Resolution Steps {9.5.3}

1. Convert problem into FOL KB
2. Convert FOL statements in KB to CNF
3. Add negation of query (in CNF) in KB
4. Use resolution to infer new clauses from KB
5. Produce a contradiction that proves our query

Resolution proof: definite clauses



Thank you

End of Chapter 9