Introduction to Artificial Intelligence

Chapter 3: Knowledge Representation and Reasoning
(4) Inference with FOL

Nguyễn Hải Minh, Ph.D
nhminh@fit.hcmus.edu.vn

Outline

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

Reducing first-order inference to propositional inference

First-order inference can be done by converting the knowledge base to propositional logic and using propositional inference

Universal Instantiation (UI)

- The rule of **Universal Instantiation**:
  - We can infer any sentence obtained by substituting a **ground term** (*a term without variables*) for the variable.

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

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

- The rule of **Existential Instantiation**:
  - for any sentence α, variable *v*, and constant symbol *k* that *does not appear elsewhere* in KB

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$

- 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:
  - $\forall x\ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  - King(John)
  - Greedy(John)
  - Brother(Richard,John)
- Instantiating the universal sentence in all possible ways, we have:
  - $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
  - $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
  - 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 to propositional inference

- 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 to propositional inference

- Theorem: Herbrand (1930)
  - If an original FOL KB |- α, then α 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 non-entailed sentence.)

Problems with propositionalization

- Propositionalization seems to generate lots of *irrelevant* sentences.
- E.g., from:
  - $\forall x\ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  - King(John)
  - $\forall y\ \text{Greedy}(y)$
  - Brother(Richard,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).

→ *The substitution θ = {x/John, y/John} works*

Generalized Modus Ponens (GMP)

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

where $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$ 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)
- $\text{SUBST}(\theta, q)$ is Evil(John)

❑All variables assumed universally quantified

→ *GMP is a lifted version of Modus Ponens*

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: UNIFY(p, q)

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

Problem is due to use of same variable x in both sentences

Standardizing apart eliminates overlap of variables
Knows(z, Steve)

Unification

- UNIFY(Knows(John, x), Knows(y, z)) = θ,
  1. θ = {y/John, x/z }
  2. 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}

p	q	θ
P(x)	P(A)	{x/A}
P(f(x), y, g(x))	P(f(x), x, g(x))	{y/x} or {x/y}
P(f(x), y, g(y))	P(f(x), z, g(x))	{y/x, z/x}
P(x, B, B)	P(A, y, z)	{x/A, y/B, z/B}
P(g(f(v)), g(u))	P(x, x)	{x/g(f(v)), u/f(v)}
P(x, f(x))	P(x, x)	*fail*

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 expression
    y, a variable, constant, list, or compound expression
    θ, the substitution built up so far (optional, defaults to empty)

  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(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
  else return failure

The unification algorithm

function UNIFY-VAR(var, x, θ) **returns** a substitution

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

Exercise

- Find the MGU when we UNIFY(p, q)

p	q	MGU?
A(B, C)	A(x, y)	
A(x, F(D, x))	A(E, F(D, y))	
A(x, y)	A(F(C, y), z)	
P(A, x, F(G(y))	P(y, F(z), F(z))	
P(x, F(y))	P(z, G(w))	

Forward Chaining

First-order Horn Clauses

- Other name: First-order definite clauses
  - Disjunctions of literals of which exactly one is positive

  → *Can be atomic or implication*
- Example:
  - King(x) ∧ Greedy(x) => Evil(x)
  - King(John)
  - Greedy(y)

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:

R1: American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

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

R2: Owns(Nono,M_1); *R3: Missile(M_1)*

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

R4: Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

R5: Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile“:

R6: Enemy(x,America) ⇒ Hostile(x)

West, who is American …

R7: American(West)

The country Nono, an enemy of America …

R8: Enemy(Nono,America)

Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*
  inputs: KB, the knowledge base, a set of first-order definite clauses
    α, the query, an atomic sentence
  local variables: *new*, the new sentences inferred on each iteration

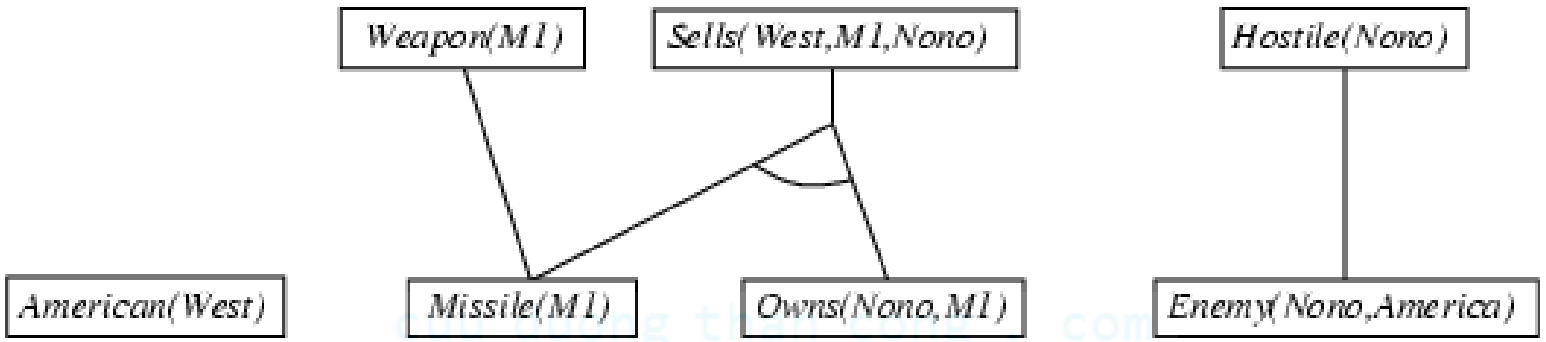
  repeat until *new* is empty
    new $\leftarrow \{\}$
    for each *rule* **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or *new* **then**
          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 Example: The crime KB

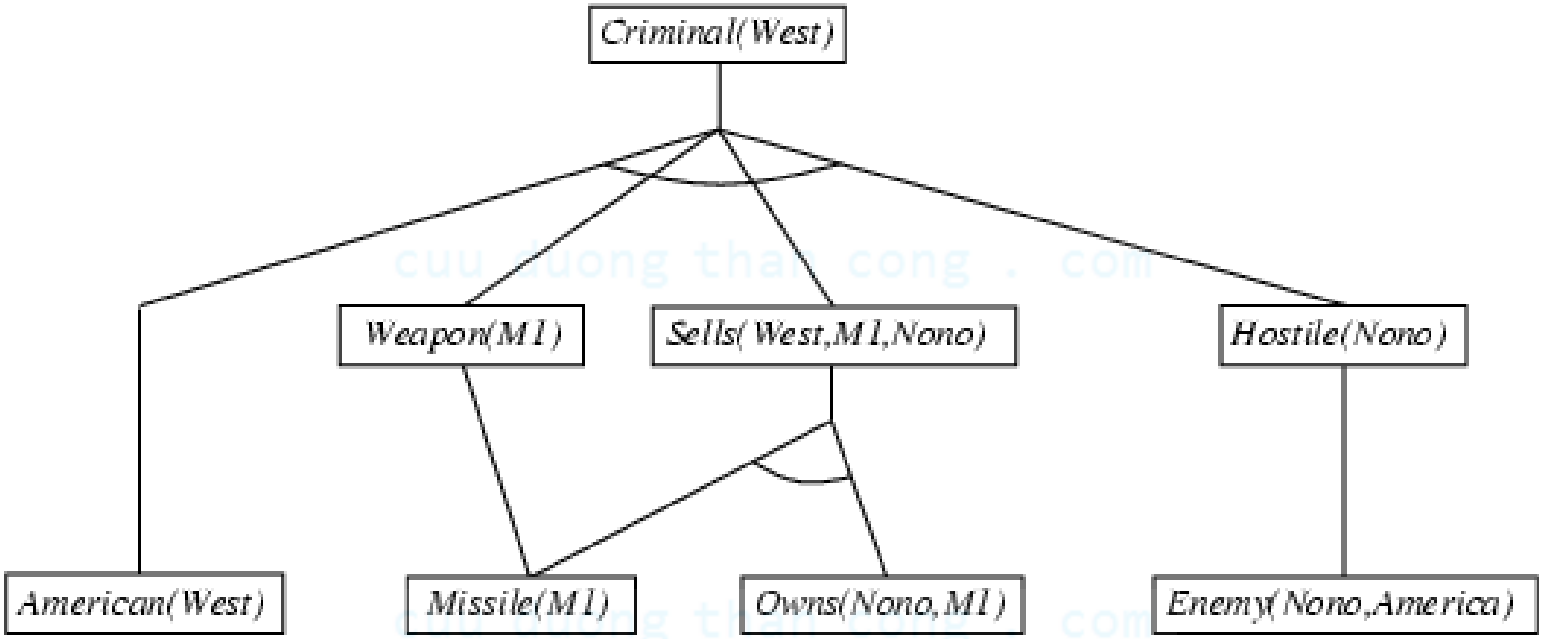
- Iteration 1:
  - R4 satisfied with {x/M1} → R9: Sells(West, M1, Nono) is added to KB
  - R5 satisfied with {x/M1} → R10: Weapon(M1) is added
  - R6 satisfied with {x/Nono} → R11: Hostile(Nono) is added
- Iteration 2:
  - R1 is satisfied with {x/West, y/M1, z/Nono} → Criminal(West) is added.

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
- 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

Backward Chaining

Backward chaining algorithm

function FOL-BC-ASK(KB, $query$) **returns** a generator of substitutions
  return FOL-BC-OR(KB, $query$, $\{\}$)

---

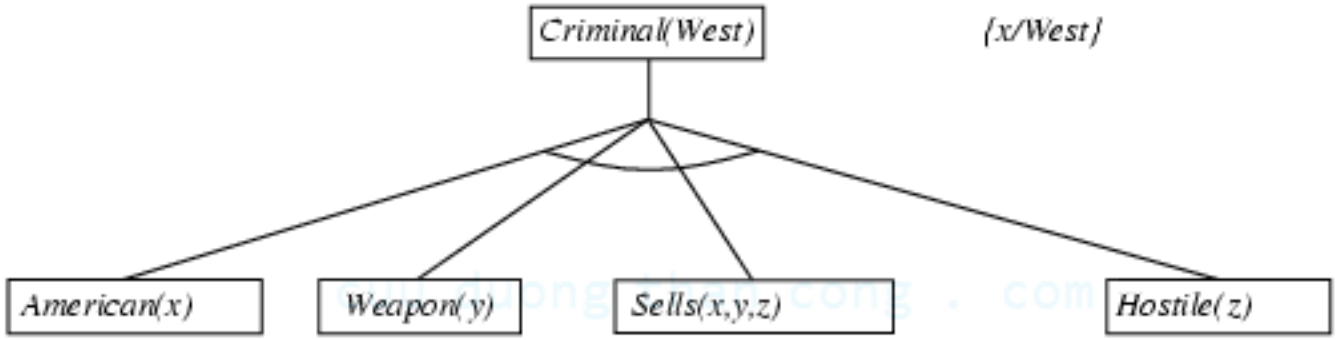
generator FOL-BC-OR(KB, $goal$, θ) **yields** a substitution
  for each rule ($lhs \Rightarrow rhs$) in FETCH-RULES-FOR-GOAL(KB, $goal$) **do**
    (lhs, rhs) $\leftarrow$ STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ' **in** FOL-BC-AND(KB, lhs, UNIFY(rhs, $goal$, θ)) **do**
      yield θ'

---

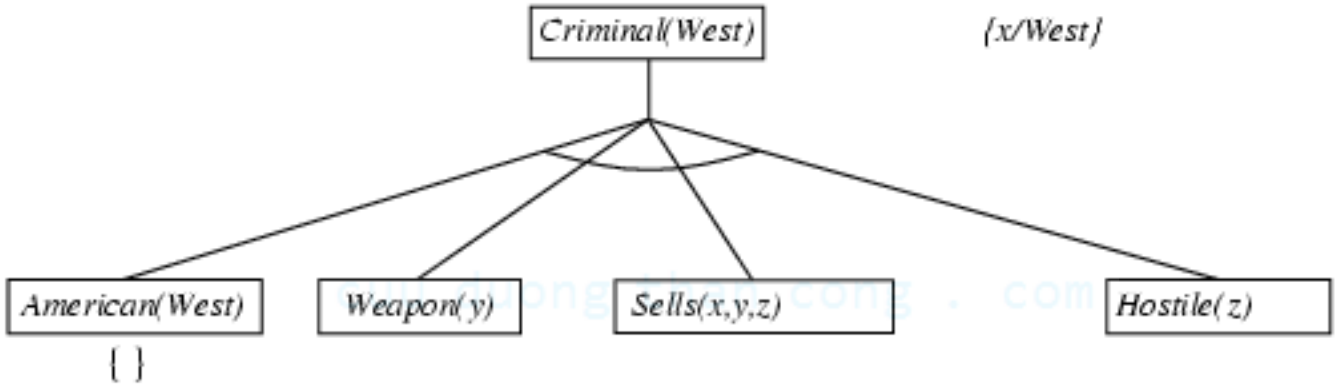
generator FOL-BC-AND(KB, $goals$, θ) **yields** a substitution
  if $\theta = failure$ **then return**
  else if LENGTH($goals$) = 0 **then yield** θ
  else do
    $first$, $rest$ $\leftarrow$ FIRST($goals$), REST($goals$)
    for each θ' **in** FOL-BC-OR(KB, SUBST(θ, $first$), θ) **do**
      for each θ'' **in** FOL-BC-AND(KB, $rest$, θ') **do**
        yield θ''

Backward chaining example

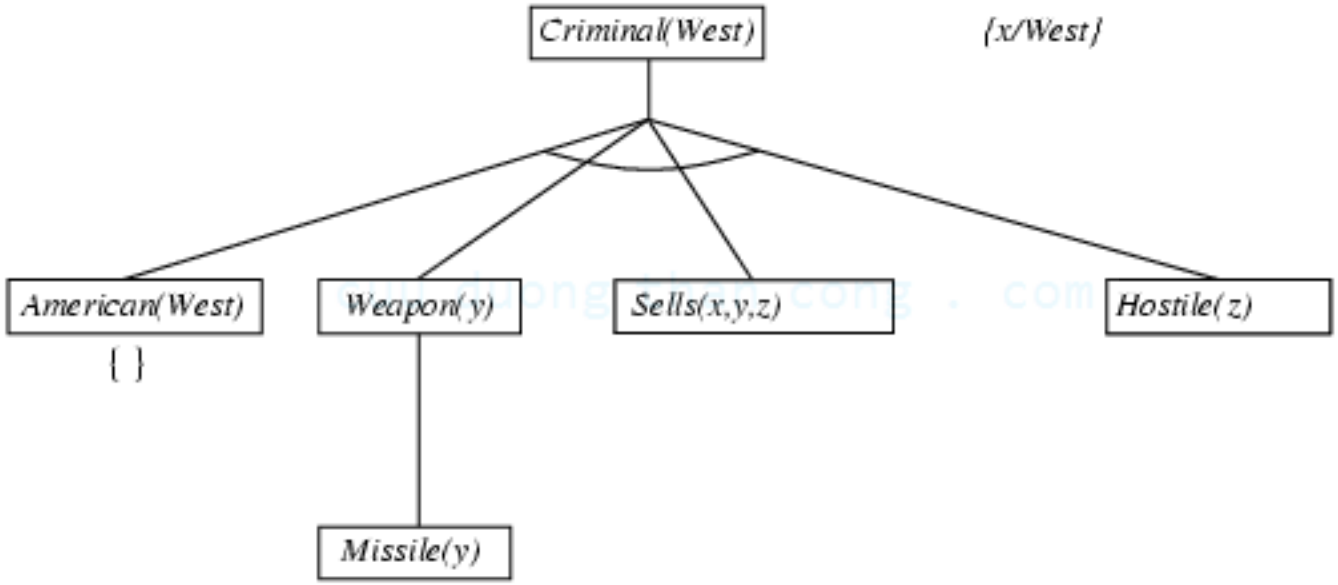
Backward chaining example



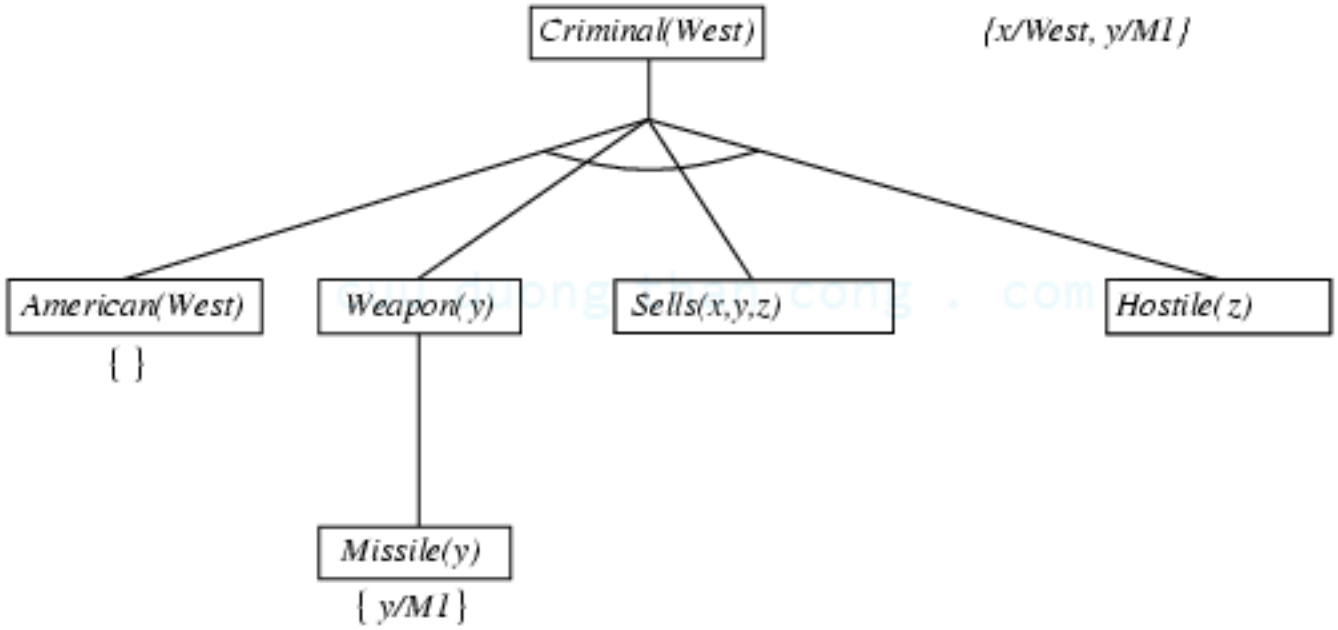
Backward chaining example



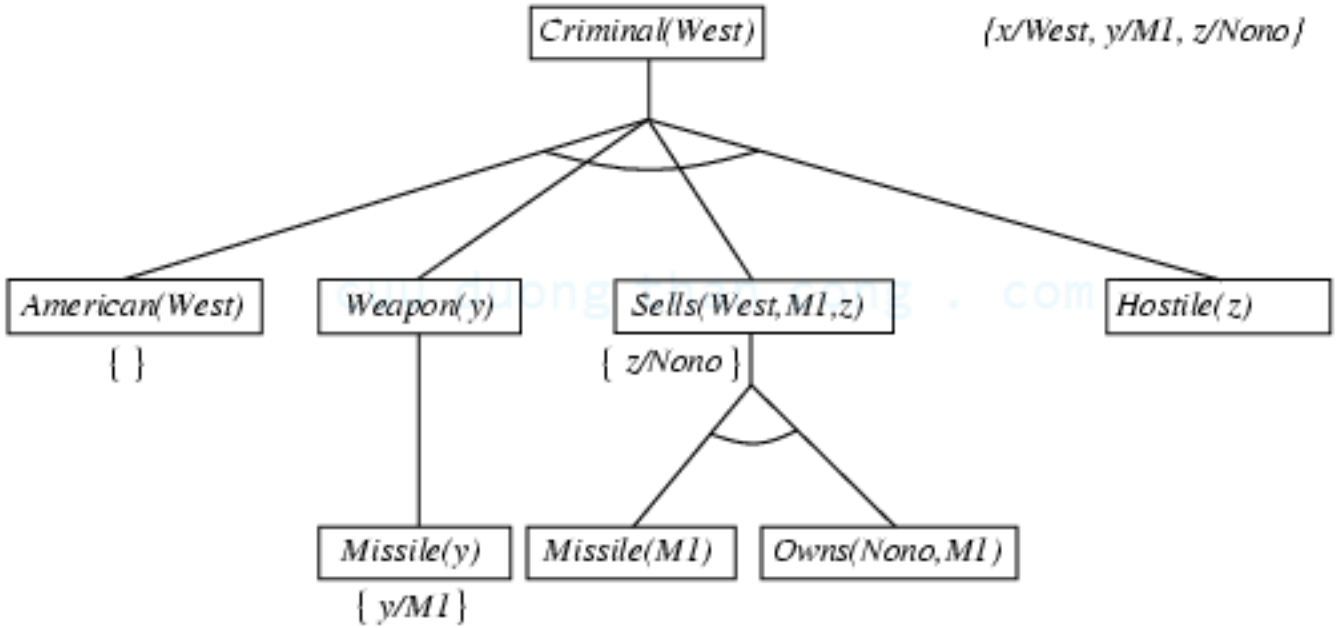
Backward chaining example



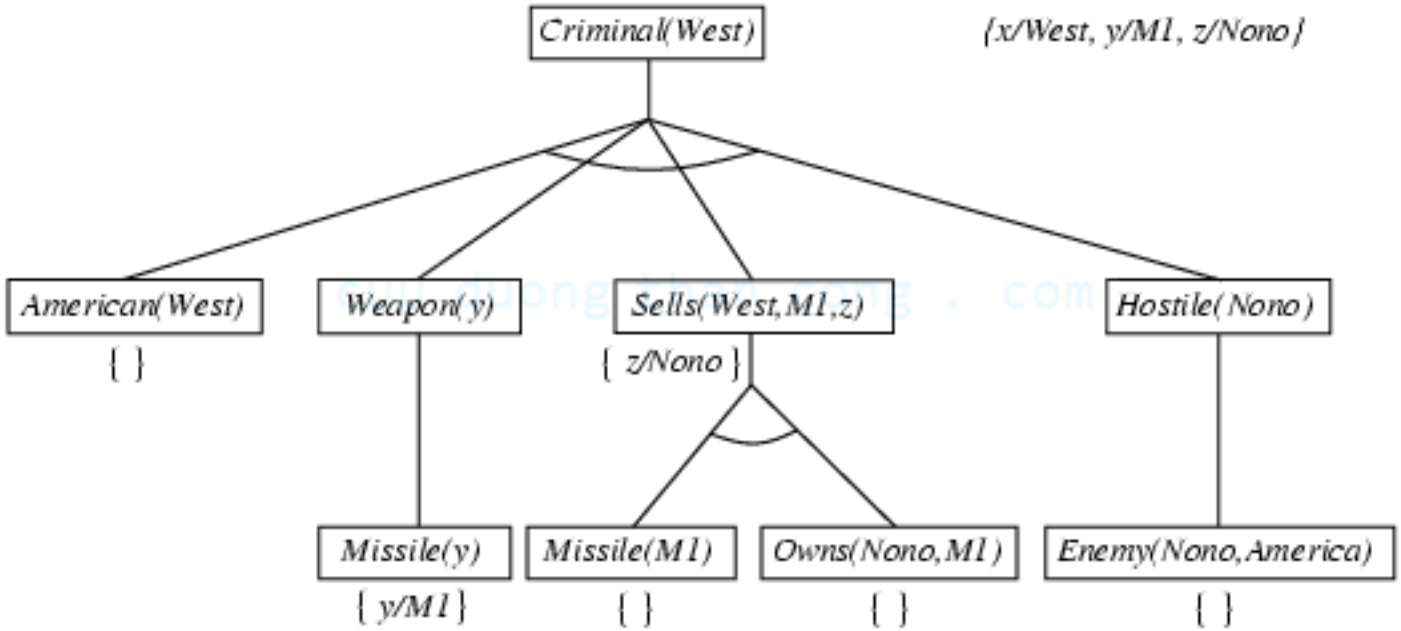
Backward chaining example



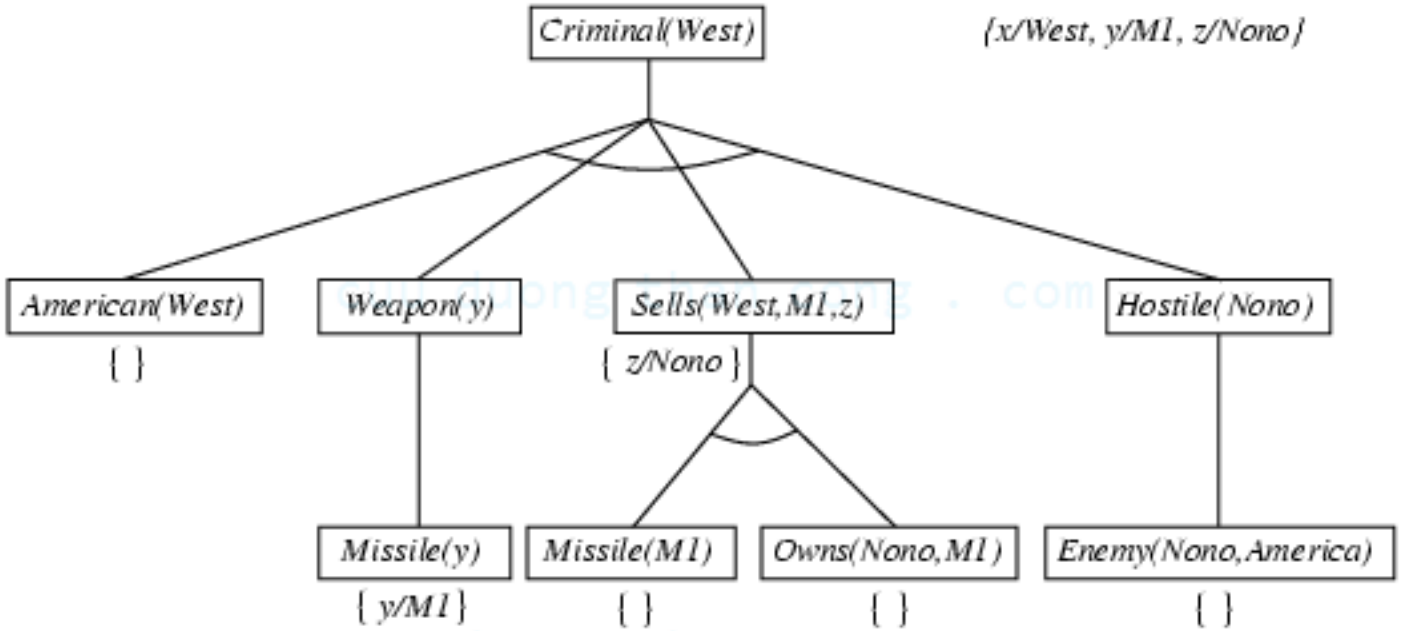
Backward chaining example



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

→ 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

Resolution

Resolution: brief summary

- 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

Conversion to CNF

Everyone who loves all animals is loved by someone:

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

1. Eliminate implications

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

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

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

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

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

Conversion to CNF contd.

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

$$\forall x\,[\exists y\,Animal(y) \land \neg Loves(x,y)] \lor [\exists z\,Loves(z,x)]$$

4. Skolemize: removing existential quantifiers by elimination. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x\,[Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(z),x)$$

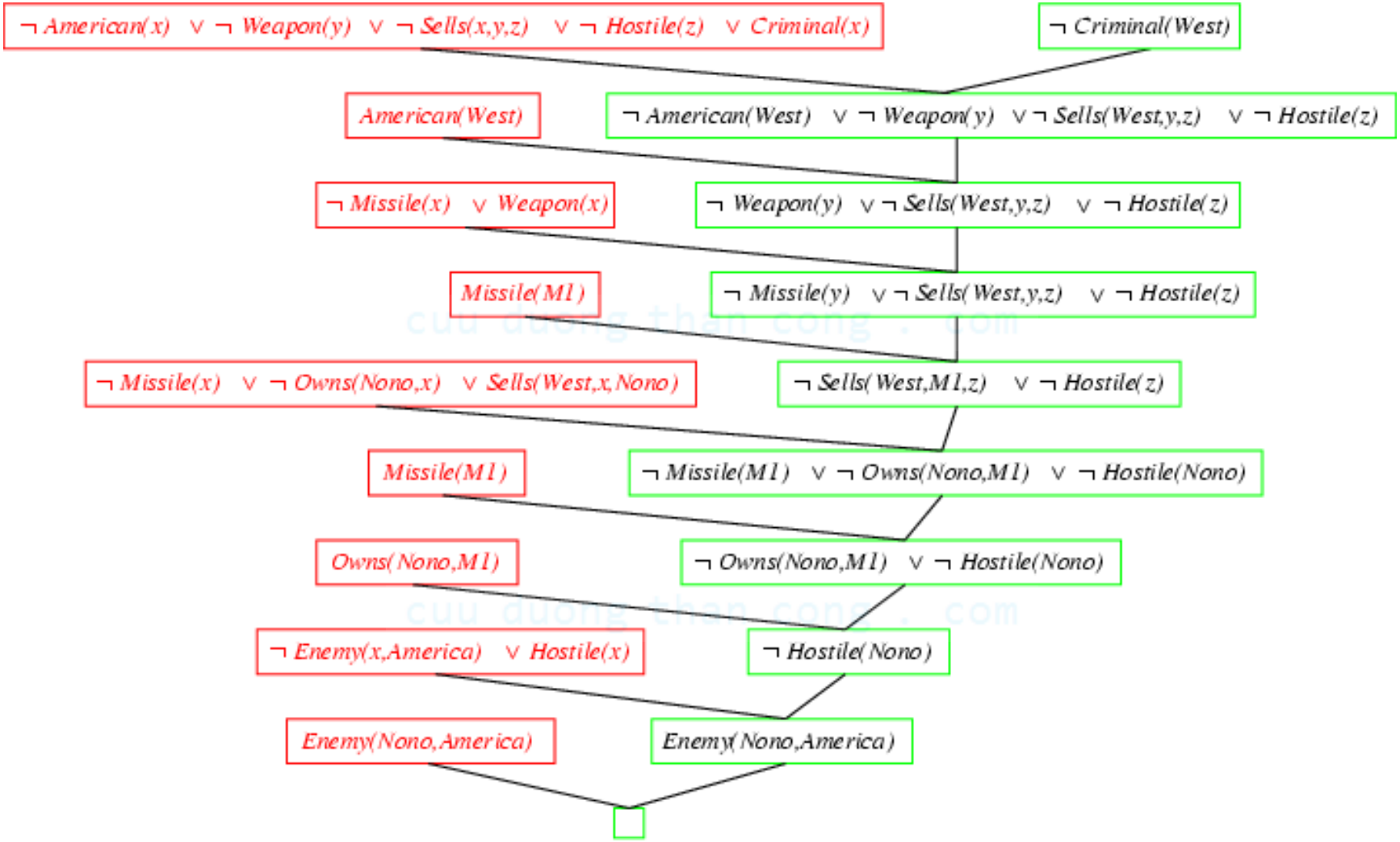
5. Drop universal quantifiers:

$$[Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(z),x)$$

6. Distribute $\lor$ over $\land$:

$$[Animal(F(x)) \lor Loves(G(z),x)] \land [\neg Loves(x,F(x)) \lor Loves(G(z),x)]$$

Resolution proof



Conclusion

- Logical inference in FOL:
  - Inference Rules (UI & Existantial Instantiation) is used to propositionalize the inference problem. It is typically slow.
  - The use of Unification to identify appropriate substitutions for variables eliminates the instantiation step in first-order proofs → more efficient.
  - Forward/Backward chaining apply Generalized Modus Ponens and unification to sets of definite clause.
    - FW chaining is complete for Datalog and runs in polynomial time
    - BW chaining suffers from redundant inferences and infinite loops
  - Generalized Resolution inference rule provides a complete proof system for FOL, using KB in CNF

Next week

- Chapter 4: Learning
  - Learning Decision Trees
  - Artificial Neural Network

cuu duong than cong . com