First-Order Logic

Tian-Li Yu

Taiwan Evolutionary Intelligence Laboratory (TEIL)
Department of Electrical Engineering
National Taiwan University
tianliyu@ntu.edu.tw

Readings: AIMA 8.2∼8.3; 9.1∼9.5.

Outline

找出替代法使得
左右兩邊變數替
換完後長得一樣

First-Order Logic (FOL)

和PL不一樣的地方

Objects: People, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, wumpus ...

Relations: red, round, bogus, prime ..., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between ...

沒有一對多、多對一的限制
relation 也可是function

Functions: father of, best friend, one more than, end of ...

- "One plus one equals two"

  Objects: one, one plus one, two; Relation: equals; Function: plus.
- "Squares neighboring the wumpus are smelly"

  Objects: wumpus, squares; Relation: smelly; Functions: neighbor of.
- One may, of course, use relations to express functions.

Syntax of FOL

Constants (PL原本就有)	A, 2, *NTU*, *John*, ...
Variables	a, b, x, y, ...
Functions	*Mother*, *LeftLeg*, ...
Predicates	*After*, *Loves*, ... (描述relation)
Connectives	$\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$.
Equality	$=$, ($\neq$ for $\neg =$).
Quantifiers	$\forall$, $\exists$.

- Check Figure 8.3 in AIMA for more details.

Quantifiers

- Typically, $\Rightarrow$ is the main connective with $\forall$.
- Typically, $\wedge$ is the main connective with $\exists$.

- Be careful
  - $\forall x\ In(x, NTU) \Rightarrow Smart(x)$: Everyone in NTU is smart.
  - $\forall x\ In(x, NTU) \wedge Smart(x)$: Everyone is in NTU and everyone is smart.
  - $\exists x\ In(x, NTU) \Rightarrow Smart(x)$: This is TRUE when no one is in NTU!
  - $\exists x\ In(x, NTU) \wedge Smart(x)$: Someone in NTU is smart.

Properties of Quantifiers

基本

- $\forall x \forall y$ is equivalent to $\forall y \forall x$.
- $\exists x \exists y$ is equivalent to $\exists y \exists x$.
- $\exists x \forall y$ is NOT equivalent to $\forall y \exists x$.
  - $\exists x \forall y\ Loves(x, y)$
    There exists someone who loves everyone.
  - $\forall y \exists x\ Loves(x, y)$
    Everyone is loved by at least one person.
- Quantifier duality
  - $\forall x\ Likes(x, IceCream)$ is equivalent to $\neg \exists x\ \neg Likes(x, IceCream)$.
  - $\exists x\ Likes(x, Studying)$ is equivalent to $\neg \forall x\ \neg Likes(x, Studying)$.

Expressing with FOL

- John has two brothers, Mark and David.

  $Brother(John, Mark) \land Brother(John, David)$?

- It only says Mark and David are John's brothers. We need to make sure John has no other brothers.

  $Brother(John, Mark) \land Brother(John, David) \land$
  $(\forall x\ Brother(John, x) \Rightarrow (x = Mark \lor x = David))$?

- John might have only one brother with two names....orz

  $Brother(John, Mark) \land Brother(John, David) \land$
  $(\forall x\ Brother(John, x) \Rightarrow (x = Mark \lor x = David)) \land (Mark \neq David)$

Back to the Wumpus World, Again

文字

- Percept([Stench, Breeze, Glitter, None, None], 5), where 5 is the step number.
- Actions: Turn(Right), Turn(Left), Forward, Shoot, Grab, Climb.
- Interaction with KB: AskVars($\exists a\ BestAction(a, 5)$)

  Returns a substitution (binding list) $\{a/\textsc{Grab}\}$.
- Define raw percept data:

  $\forall t, s, g, m, c\ \textsc{Percept}([s, \textsc{Breeze}, g, m, c], t) \Rightarrow Breeze(t)$.

  $\forall t, s, b, m, c\ \textsc{Percept}([s, b, \textsc{Glitter}, m, c], t) \Rightarrow Glitter(t)$.
- Simple reflex best action:

  $\forall t\ Glitter(t) \Rightarrow BestAction(\textsc{Grab}, t)$.

Back to the Wumpus World, Again

- Define adjacency:

  $\forall x, y, a, b\ Adjacent([x, y], [a, b]) \Leftrightarrow$

  $(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$.

- Location predicator, x is at square s at time t:

  $\forall t\ At(\textsc{Wumpus}, [2, 2], t)$. 只有一隻wumpus

  $\forall x, s_1, s_2, t\ At(x, s_1, t) \wedge At(x, s_2, t) \Rightarrow s_1 = s_2$.

- Define property for squares:

  原本定義在時間點上 –> 定義在方格上

  $\forall s, t\ At(\textsc{Agent}, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$.

  $\forall s, t\ At(\textsc{Pit}, s, t) \Rightarrow Pit(s)$.

  $\forall s, t\ At(\textsc{Wumpus}, s, t) \Rightarrow Wumpus(s)$.

- Rules of the wumpus world can be defined.

  $\forall s\ Breezy(s) \Leftrightarrow \exists r\ Adjacent(r, s) \wedge Pit(r)$.

  $\forall t\ HaveArrow(t + 1) \Leftrightarrow (HaveArrow(t) \wedge \neg Action(Shoot, t))$.

Instantiation

重點兒

- $\text{SUBST}(\theta, \alpha)$: apply the substitution θ to the sentence α.

Universal Instantiation(UI)

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

可以做很多次

where g is a ground term.

Existence Instantiation(EI)

只能做一次

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

where k was not in the KB.

- $\forall King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields X 可以隨便換無限多次

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

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

- $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields

  $Crown(C_1) \wedge OnHead(C_1, John)$, where C_1 is a Skolem constant.

  但這個只能換一次，且這個 symbol 不能在你的 knowledge base裡面

ex.
可以說x殺了John
但不能說Mark 殺了john
最後想要得到x=Mark

Instantiation

- UI can be applied several times to add new sentences; the new KB is logically equivalent to the old.

搞不好只存在一個，apply第二次會出事～～

- EI can be applied only once to replace the existential sentence.
- No longer need ~~$\exists x\ Kill(x, Victim)$~~ once we have $Kill(Murderer, Victim)$.
- Strictly speaking, the new KB is **not** logically equivalent to the old.
- However, the new KB is satisfiable **iff** the old was satisfiable.
- We call them inferentially equivalent.

EI 替換後兩個邏輯不等價，但在推論上等價
用兩種導出來的東西會相同

推論上等價
邏輯上不等價

Reduction to Propositional Inference

- Suppose the KB contains only the following:

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

  $King(John)$

  $Greedy(John)$

  $Brother(Richard, John)$

- 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.

Reduction to Propositional Inference

重要

- A ground sentence is entailed by new KB **iff** entailed by original KB.
- Every FOL KB can be propositionalized so as to preserve entailment.
- Propositionalize KB and query, apply resolution, return result.
- Problem: with function symbols, there are infinitely many ground terms, e.g., $Father(Father(Father(John)))$ FOL 只要有function: ground term 就會∞ cannot transform to a finite PL
- **Theorem:** Herbrand (1930). If a sentence is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB.
- Idea: For $d = 0$ **to** ∞ **do**

  create a propositional KB by instantiating with depth-d terms

  see if α is entailed by this KB.

  如果和John有關係：
  必在finite裡 （e.g. John的朋友的朋友的朋友...
  如果和John沒關係：
  怎麼證明和他沒關係？如果再多一層？
- Problem: works if α is entailed, loops if α is not entailed.
- Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable. 如果是 entail 的話，我可以在有限時間內 verify 你是 yes，是 no 的話沒辦法在有限的時間內證明你是 no

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences and can be inefficient.
  E.g., from the KB,

$$\forall x\ \overset{\text{P1}}{King}(x) \wedge \overset{\text{P2}}{Greedy}(x) \Rightarrow \overset{\text{q}}{Evil}(x)$$
$$King(John)\ \text{P1'}$$
$$\forall y\ Greedy(y)\ \text{P2'}$$
$$Brother(Richard, John)$$

可是y可以填很多人！但這些事實他們都跟誰是evil無關

找到一個subst 他們一樣 x/john, y/john

  It seems obvious that query $Evil(x)$ yields $x = John$, but propositionalization produces lots of irrelevant facts such as $Greedy(Richard)$.

- With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations

- With function symbols, it gets much worse!

Generalized Modus Ponens (GMP)

GMP

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

前提（替換）

結論

where $\forall i \ \ \textsc{Subst}(\theta, p_i') = \textsc{Subst}(\theta, p_i)$

p_1' is $King(John)$
p_2' is $Greedy(y)$
θ is $\{x/John, y/John\}$
$\textsc{Subst}(\theta, q)$ is $Evil(John)$

p_1 is $King(x)$
p_2 is $Greedy(x)$
q is $Evil(x)$

眾多 V or 裡 只有一個+

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

Soundness of GMP

- Need to show that

$$p_1', \ldots, p_n', \quad (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models \textsc{Subst}(\theta, q)$$

provided that $\forall i \ \textsc{Subst}(\theta, p_i') = \textsc{Subst}(\theta, p_i)$

- Lemma: For any definite clause p, we have $p \models \textsc{Subst}(\theta, p)$ by UI.

Proof.

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models \textsc{Subst}(\theta, p_1 \wedge \ldots \wedge p_n \Rightarrow q) = \textsc{Subst}(\theta, p_1) \wedge \ldots \wedge \textsc{Subst}(\theta, p_n) \Rightarrow \textsc{Subst}(\theta, q)$ 分配律XD
2. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models \textsc{Subst}(\theta, p_1') \wedge \ldots \wedge \textsc{Subst}(\theta, p_n')$
3. From 1 and 2, $\textsc{Subst}(\theta, q)$ follows by ordinary Modus Ponens.

□

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

- $\textsc{Unify}$ takes two sentences and returns a unifier for them:
  兩邊看起來沒什麼不一樣

  $\textsc{Unify}(p, q) = \theta$ where $\textsc{Subst}(\theta, p) = \textsc{Subst}(\theta, q)$.

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Bill)$	$\{x/Bill, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, Eliza)$	$fail$ $(why?)$

x: dumy variable
John認識所有人 和 所有人認識Eliza 不衝突

我們還不知道
John 是否等於 Eliza

Unification

- Standardizing apart eliminates overlap of variables

  $\text{UNIFY}(Knows(John, x), Knows(x_{17}, Eliza)) = \{x/Eliza, x_{17}/John\}$

- UNIFY returns the most general unifier (MGU) if there are several.

  E.g., $\text{UNIFY}(Knows(John, x), Knows(y, z))=$
  - $\{y/John, x/z\}$ (MGU)
  - $\{y/John, x/John, z/John\}$

- To retrieve MGU, the algorithm recursively explore the expressions simultaneously.

  Need to perform occur check so that $S(x)$ doesn't unify with $S(S(x))$.

First-Order Definite Clauses

- 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.
- For efficient inference, we use first-order definite clauses:
  - Exactly one positive literal.
  - May include variables (universally quantified).

Knowledge Base Using First-Order Definite Clauses

變數都是for all

- "... it is a crime for an American to sell weapons to hostile nations":
  $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
- "Nono ... has some missiles", i.e., $\exists x\ Owns(Nono, x) \land Missile(x)$:
  $Owns(Nono, M_1)$ and $Missile(M_1)$ 把存在換成 constant，讓變數都是 for all
- "... all of its missiles were sold to it by Colonel West":
  $\forall x\ Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- 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

FOL-FC-Ask(KB, α)

```
1  repeat until new is empty
2      new = {}
3      for each rule in KB
4          (p1 ∧ ... ∧ pn ⇒ q) = Standardize-Variables(rule)
5          for each θ such that Subst(θ, p1 ∧ ... ∧ pn) ==
                                Subst(θ, p1' ∧ ... ∧ pn') for some p1', ..., pn' in KB
6              q' = Subst(θ, q)
7              if q' does not unify with some sentence already in KB or new
8                  add q' to new
9                  φ = Unify(q', α)
10                 if φ is not fail
11                     return φ
12     add new to KB
13 return FALSE
```

和先前不一樣：
無法count多少個

Forward Chaining Proof

rules

Facts

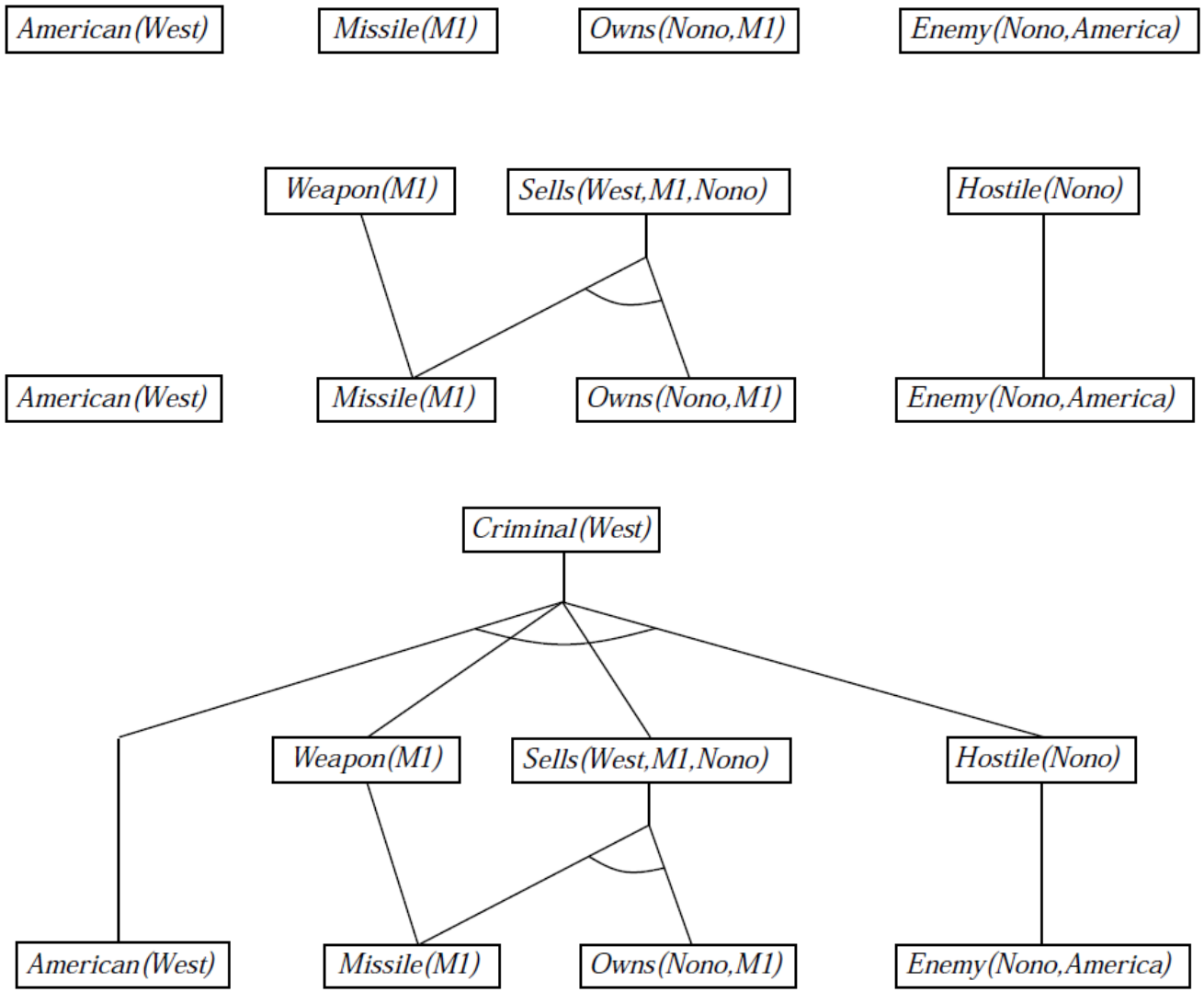
1. $Owns(Nono, M_1)$
2. $Missile(M_1)$
3. $American(West)$
4. $Enemy(Nono, America)$

Implications

5. $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
6. $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
7. $Missile(x) \Rightarrow Weapon(x)$
8. $Enemy(x, America) \Rightarrow Hostile(x)$

- 1^{st} iteration, R_5 has unsatisfied premises.
  R_6 is satisfied with $\{x/M_1\}$, $Sells(West, M_1, Nono)$ is added.
  R_7 is satisfied with $\{x/M_1\}$, $Weapon(M_1)$ is added.
  R_8 is satisfied with $\{x/Nono\}$, $Hostile(Nono)$ is added.
- 2^{nd} iteration, R_5 is satisfied with $\{x/West, y/M_1, z/Nono\}$, $Criminal(West)$ is added.

Forward Chaining Proof



Properties of Forward Chaining

- Sound and complete for first-order definite clauses.
  Sound because of generalized Modus Ponens.
  Complete proof similar to propositional proof by introducing the concept of fixed point.

- Datalog = first-order definite clauses + no functions (e.g., crime KB)
  FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ distinct ground facts.

- May not terminate in general if α is not entailed.

- This is unavoidable: entailment with definite clauses is semi-decidable.

Efficiency of Forward Chaining

- Simple observation: 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 literal.

- Matching itself can be expensive.

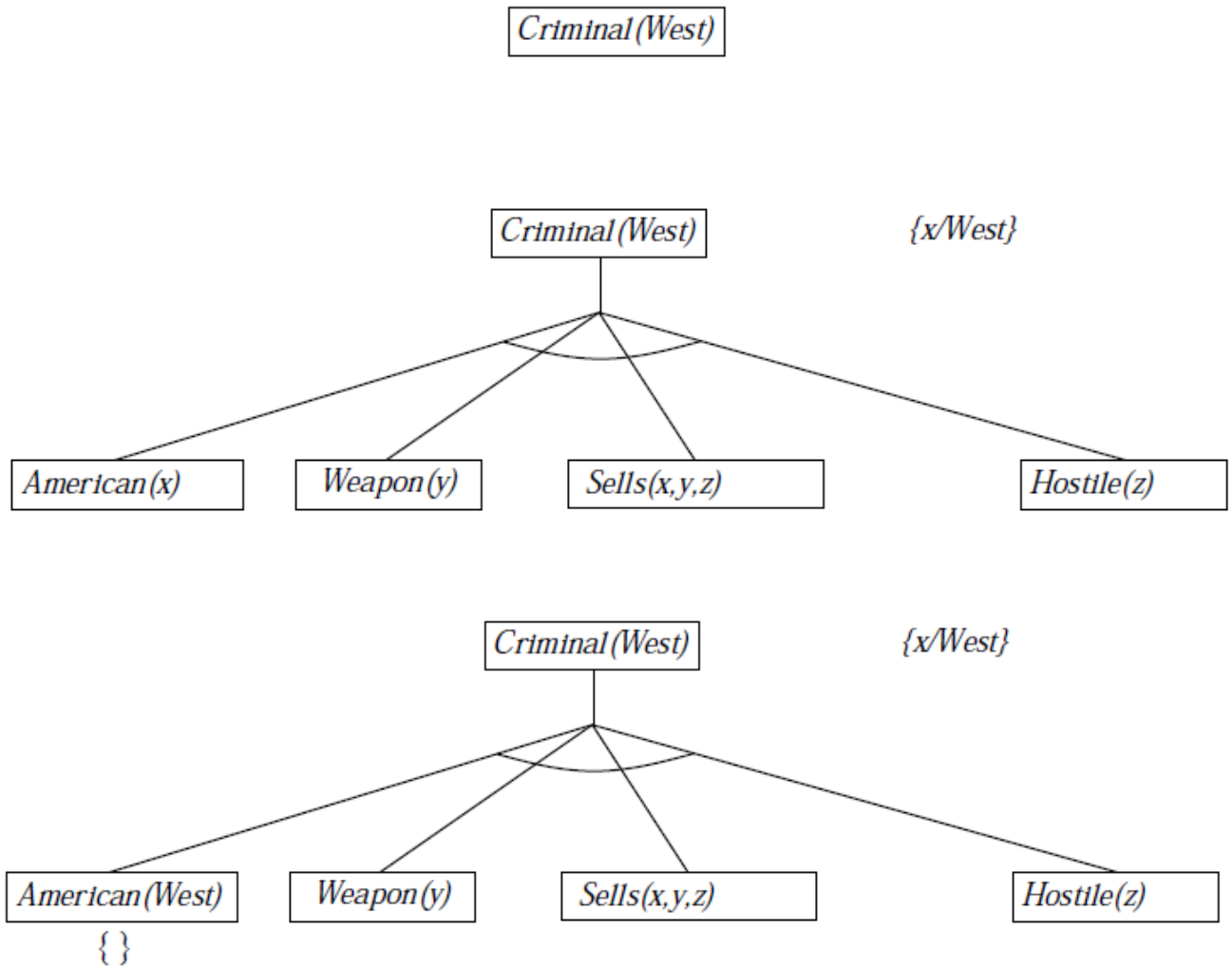
hash table

- Database indexing allows $O(1)$ retrieval of known facts
  e.g., query $Missile(x)$ retrieves $Missile(M_1)$

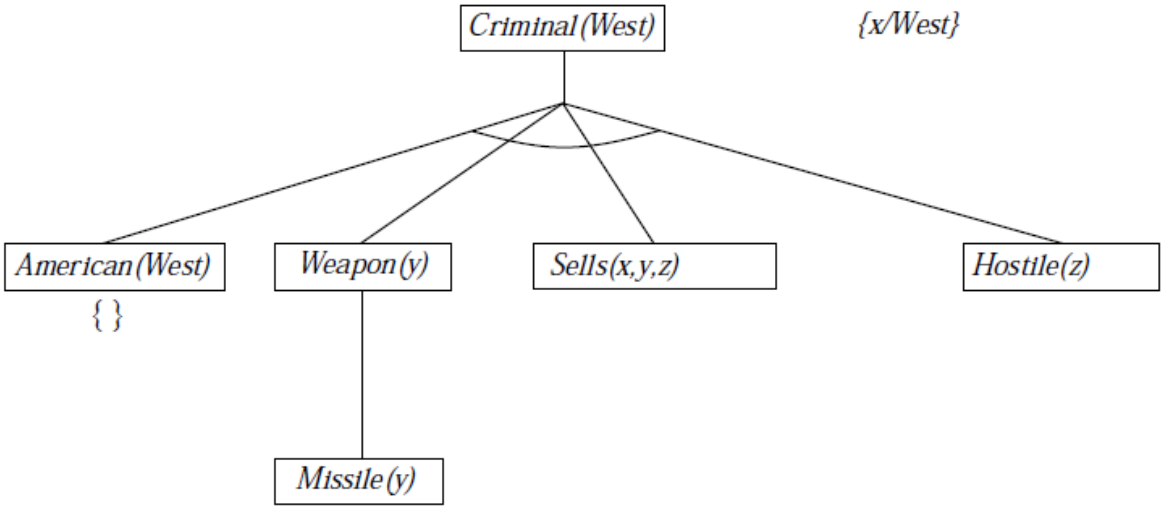
- Matching conjunctive premises against known facts is $\mathcal{NP}$-hard.
  先問誰(order的問題)比較好？
  e.g., $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
  Conjunction ordering problem: find an ordering to minimize the cost.
  Nevertheless, good heuristics are available.
  可能 Missile(M1) 是 true 但 Owns(Nono, M1)是 false，如何有效找 x 是 NP-hard

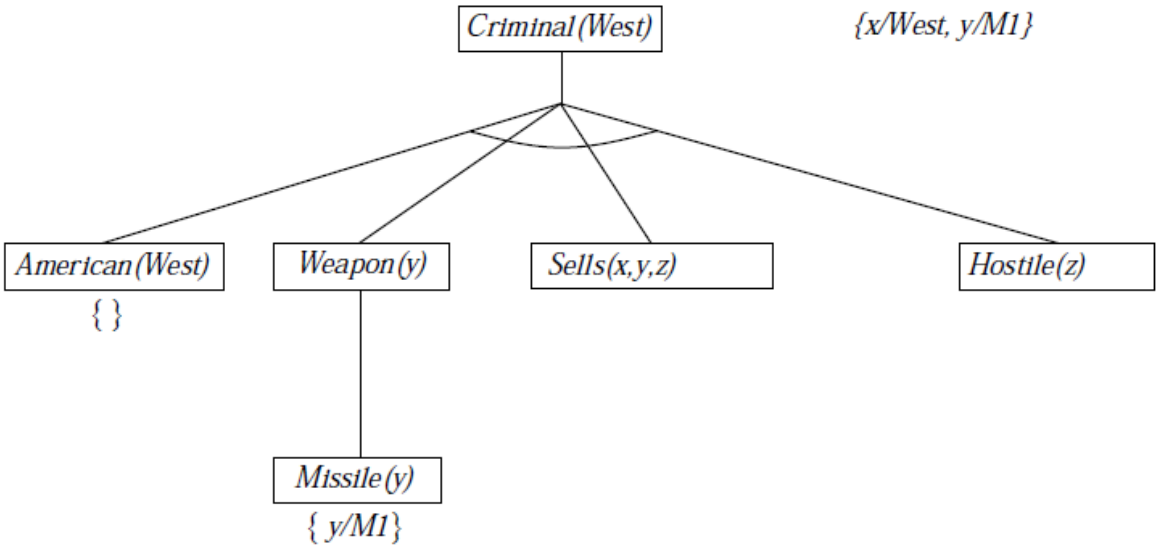
- Forward chaining is widely used in deductive databases

Backward Chaining Example

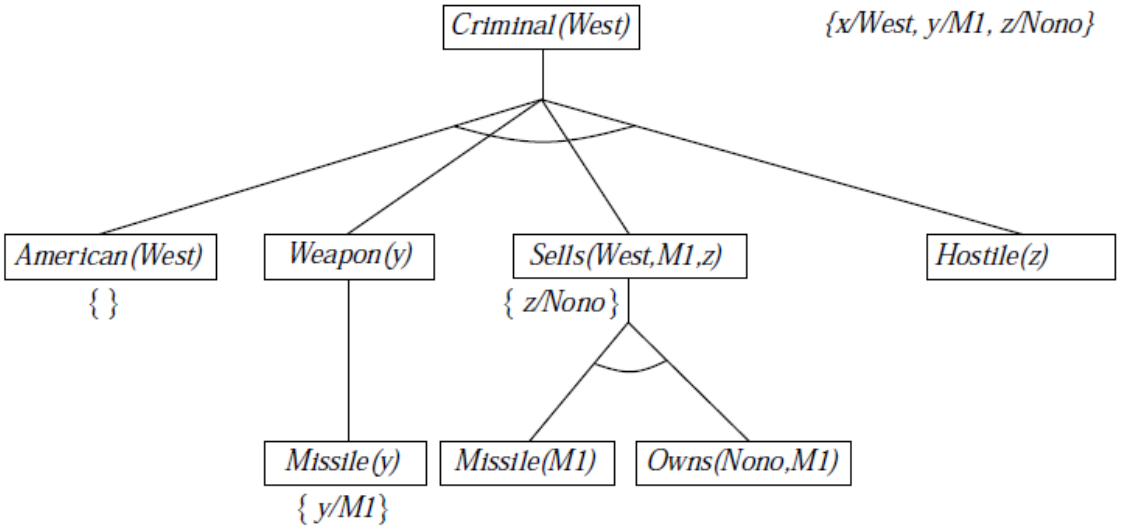


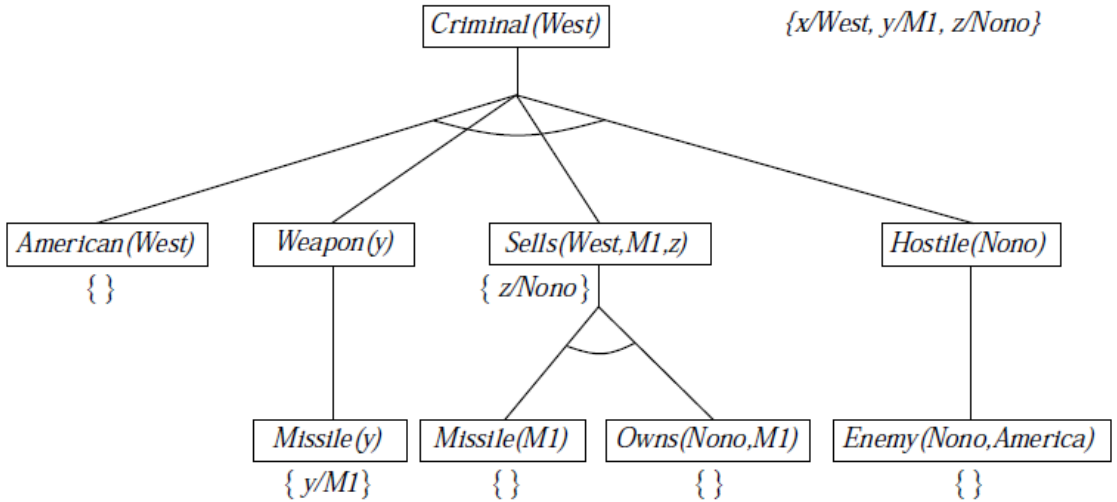
Backward Chaining Example





Backward Chaining Example





Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof.

Depth-first 因為memory不夠，但問題：一路在左支往下衝不回頭

- AND-OR search: AND for all premises; OR since the goal query can be proved by any rules.

- 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 (without improvements!) for logic programming

Logic Programming

- *Algorithm* = *Logic* + *Control*

	Logic Programming	Ordinary Programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem as facts	Encode problem as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Prolog Systems

不會要你寫，但是你要看的懂，要知道他 inference 結果是什麼，或是會不會卡住
要去 ceiba 跑跑看程式，一定會考

- Lowercases for constants; uppercases for variables (opposite of the textbook).
- Program = set of definite clauses.
  $A \wedge B \Rightarrow C$ in Prolog is `C :- A, B.`
  `criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`
- `[E|L]` is a list whose first element is `E` and rest is `L`.

Prolog Examples

backward training

- Depth-first search from a start state `X`:

```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

- Appending two lists (`X` and `Y`) to produce a third (`Z`):

```
append([],Y,Y).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).
```

(X 連 A 再連 Y 就是 Z 連 A)　(前提：X 連 Y 得到 Z)

query: `append(X,Y,[1,2]) ?`

answers: `X=[] Y=[1,2];`
`X=[1] Y=[2];`
`X=[1,2] Y=[]`

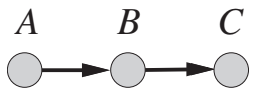
Prolog Systems

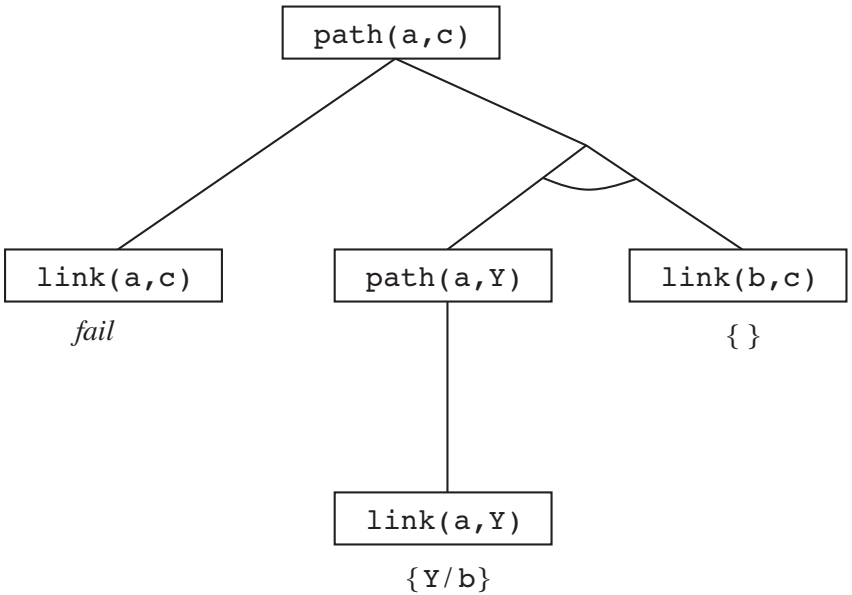
- Unification without the occur check, may results in unsound inferences. But almost never a problem in practice.
- Depth-first, left-to-right backward chaining search with no checks for infinite recursion.
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`; no arbitrary equation solving, e.g., `5 is X+Y` fails.
- Database semantics instead of first-order semantics.

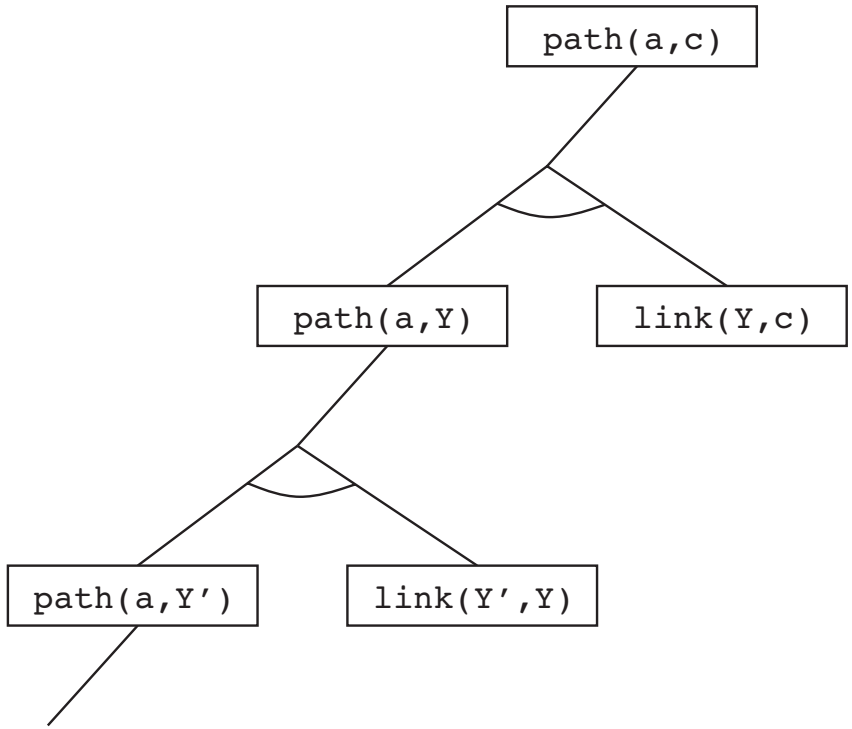
Redundant Inference and Infinite Loops in Prolog

```
path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y), link(Y,Z).
```

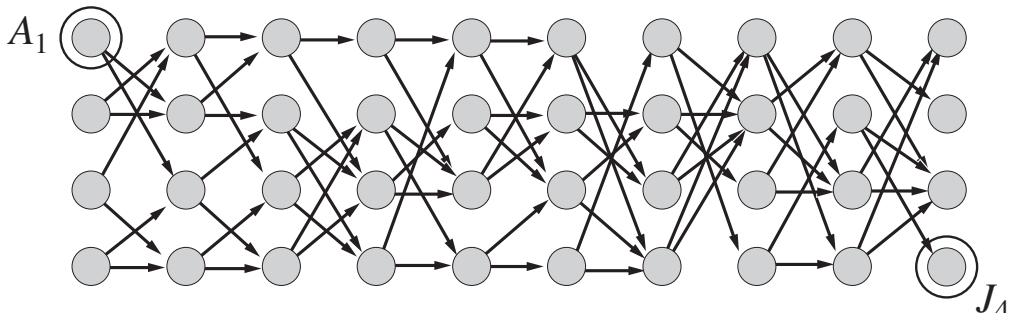
Query: `path(a,c) ?`







Redundant Inference and Infinite Loops in Prolog



- Backward chaining (Prolog) takes 877 inferences.
- Forward chaining (similar to dynamic programming) takes only 62 inferences.
- To make backward chaining more efficient, memoization can be adopted, but extra memory is needed.

Database Semantics of Prolog

- Unique-names assumption — different names refer to distinct objects.
- Closed-world assumption — anything not known to be true is false.
- Domain closure — only those mentioned exist in the domain.

Prolog assertions:

```
Course(CS,101), Course(CS,102), Course(CS,106), Course(EE,101).
```

FOL:

at most 4 courses:

$$Course(d, n) \Leftrightarrow (d = CS \wedge n = 101) \vee (d = CS \wedge n = 102) \vee (d = CS \wedge n = 106) \vee (d = EE \wedge n = 101).$$

at least 4 courses:

$$x = y \Leftrightarrow (x = CS \wedge y = CS) \vee (x = EE \wedge y = EE) \vee (x = 101 \wedge y = 101) \vee (x = 102 \wedge y = 102) \vee (x = 106 \wedge n = 106).$$

Resolution

- Full first-order version:

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

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

- For example,

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

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

- Apply resolution steps to $\mathit{CNF}(\mathit{KB} \wedge \neg\alpha)$; 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 biconditionals and 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

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:** a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

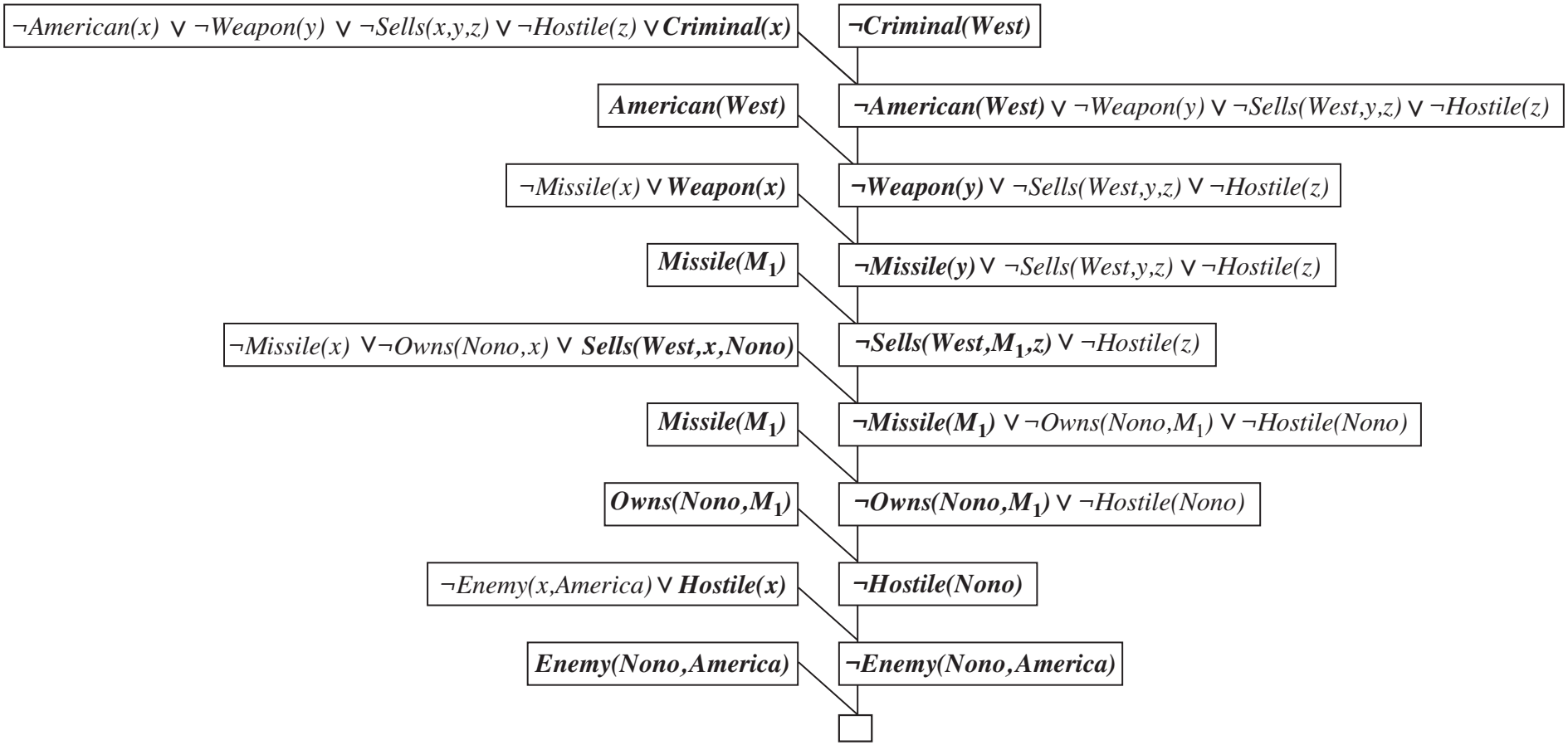
5. Drop universal quantifiers:

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

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

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

Resolution Proof: Definite Clauses



Completeness of Resolution

- Resolution is refutation-complete. If a set of sentences is unsatisfiable, resolution always derives a contradiction.
- It cannot generate all logical consequences.
- It can find all answers of a given question, $Q(x)$, by proving that $KB \wedge \neg Q(x)$ is unsatisfiable.
- Check out AIMA for the (brief) proof:

> *if S is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to S will yield a contradiction.*

Summary

- For small domains, we can use UI and EI to propositionalize the problem.
  很多次　1次
  推論上等價、邏輯上不等價
- Unification identifies proper substitutions, more efficient than instantiation.
- Forward- and backward-chaining uses the generalized Modus Ponens on a sets of definite clauses.
- GMP is complete for definite clauses, where the entailment is semi-decidable; for Datalog KB (function-less definite clauses), entailment can be decided in $\mathcal{P}$-time (with forward-chaining).
- Backward chaining is used in logic programming systems; inferences are fast but may be unsound or incomplete.
- Resolution is sound and (refutation-)complete for FOL, using CNF KB.