Knowledge Representation & Reasoning COMP9016

Dr Ruairí O’Reilly
ruairi.oreilly@cit.ie

Inference in First Order Logic

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Outline

>> Reducing first-order inference to propositional inference

>> Unification

>> Generalized Modus Ponens

>> Forward and backward chaining

>> Logic programming

>> 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., $\forall x \;\; King(x) \wedge Greedy(x) \;\Rightarrow\; Evil(x)$ 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 (EI)

>> For any sentence a, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:

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

>> E.g., $\exists x \quad Crown(x) \wedge OnHead(x, John)$ yields

$$Crown(C_1) \wedge OnHead(C_1, John)$$

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

Another example: from $\exists x \;\; d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided e is a new constant symbol

Existential instantiation contd.

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

>> EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but is satisfiable iff the old KB was satisfiable

>> ∃ x Kill(x, Victim) – added initially

>> Kill (Murderer , Victim) – Once added above is irrelevant.

Reduction to propositional inference

>> Suppose the KB contains just 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: proposition symbols are

$King(John)$, $Greedy(John)$, $Evil(John)$, $King(Richard)$ etc.

Reduction contd.

Claim: a ground sentence* is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

Idea: propositionalize KB and query, apply resolution, return result

Problem: with function symbols, there are infinitely many ground terms,

e.g., $Father(Father(Father(John)))$

Theorem: Herbrand (1930). If a sentence a is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB

Idea: For $n = 0$ to ∞ do

create a propositional KB by instantiating with depth-n terms see if a is entailed by this KB

Problem: works if a is entailed, loops if a is not entailed

>> Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences.
E.g., from

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

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

With function symbols, it gets much worse!

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

$\text{Unify}(\alpha, \beta) = \theta$ 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., $Knows(x_{17}, OJ)$

$UNIFY(Knows(John, x), Knows(x17, OJ)) = \{x/OJ, x17/John\}$.

Generalized Modus Ponens (GMP)

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

where $p_i'\theta = p_i\theta$ 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

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 definite clause 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. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$

3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

One further complication

UNIFY should return a substitution that makes the two arguments look the same. But there could be more than one such unifier.

UNIFY(Knows(John, x), Knows(y, z))
could return the following unifiers:
i) {y/John, x/z} or
ii) {y/John, x/John, z/John}.

The first unifier gives
Knows(John, z)

The second unifier gives
Knows(John, John)

The second result could be obtained from the first by an additional substitution {z/John};

It turns out that, for every unifiable pair of expressions, there is a single **most general unifier** (or MGU) that is unique up to renaming and substitution of variables

```
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
```

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

   if {var/val} ∈ θ then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {var/x} to θ
```

Figure 9.1 The unification algorithm. The algorithm works by comparing the structures of the inputs, element by element. The substitution θ that is the argument to UNIFY is built up along the way and is used to make sure that later comparisons are consistent with bindings that were established earlier. 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).

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

$Owns(Nono, M_1)$ and $Missile(M_1)$

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

$$\forall x \quad Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

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 algorithm

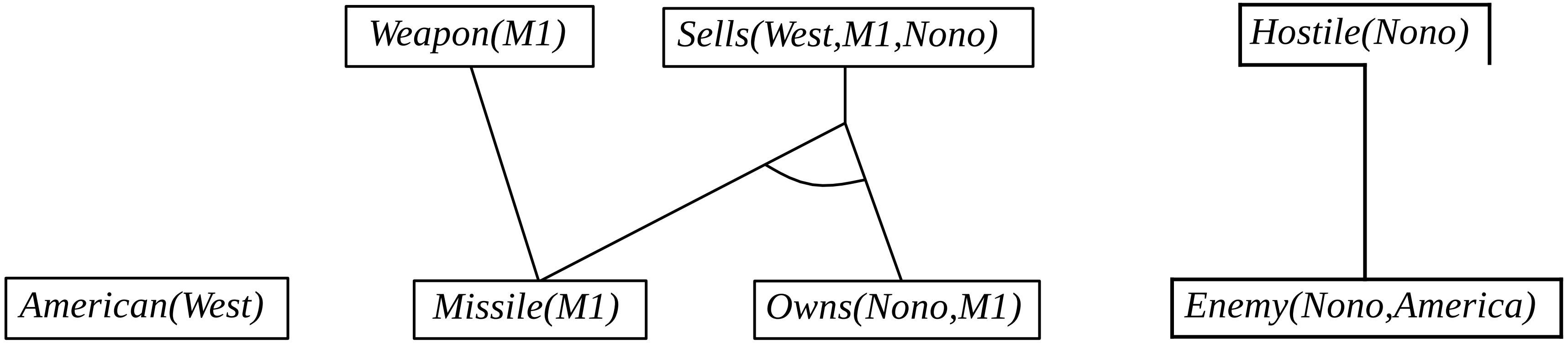
```
function FOL-FC-Ask(KB, α) returns a substitution or false

  repeat until new is empty
      new ← { }
      for each sentence r in KB do
          ( p1 ∧ . . . ∧ pn  ⇒  q) ← Standardize-Apart(r)
          for each θ such that (p1 ∧ . . . ∧ pn)θ = (p'1 ∧ . . . ∧ p'n)θ
                          for some p'1, . . . , p'n in KB
              q' ← Subst(θ, q)
              if q' is not a renaming of a sentence already in KB or new then do
                  add q' to new
                  φ ← Unify(q', α)
                  if φ is not fail then return φ
      add new to KB
  return false
```

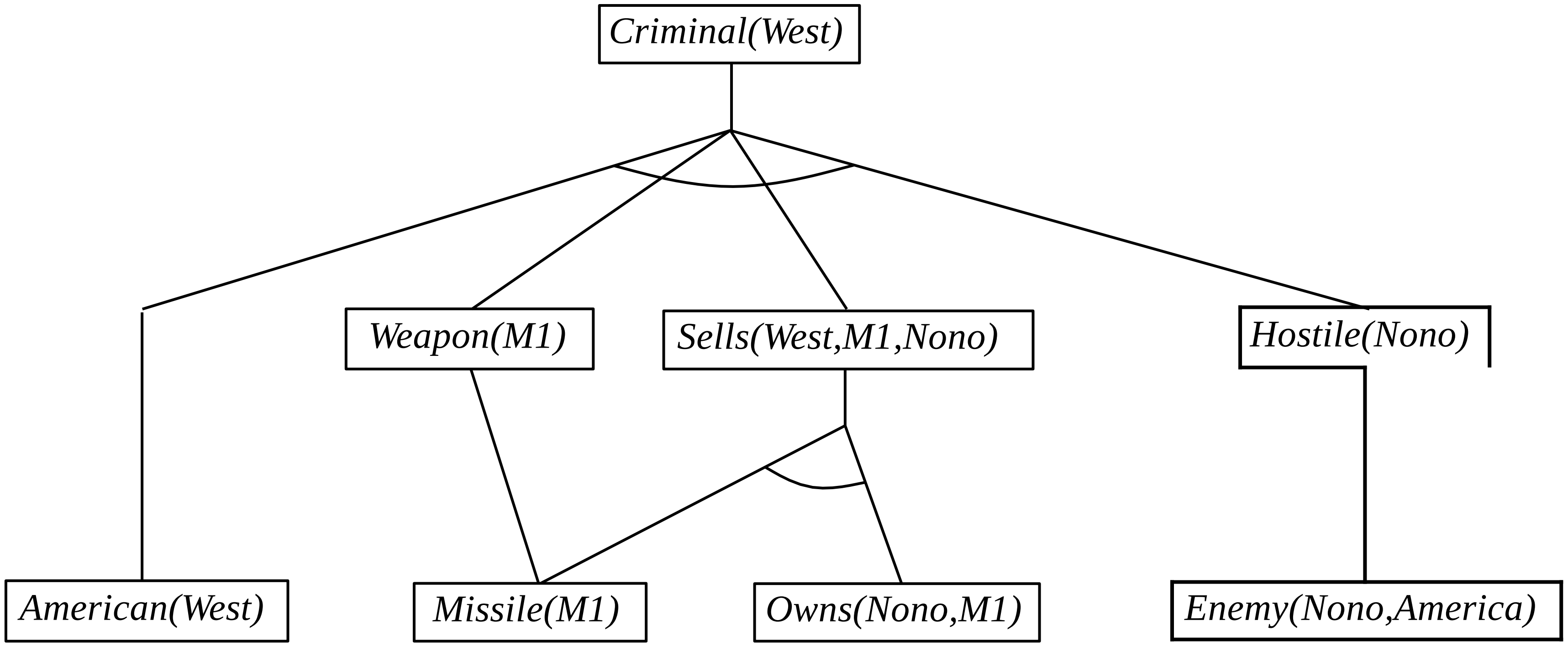
Forward chaining proof

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
(proof similar to propositional proof)

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

May not terminate in general if a is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

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

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 NP-hard

Forward chaining is widely used in deductive databases

Backward chaining algorithm

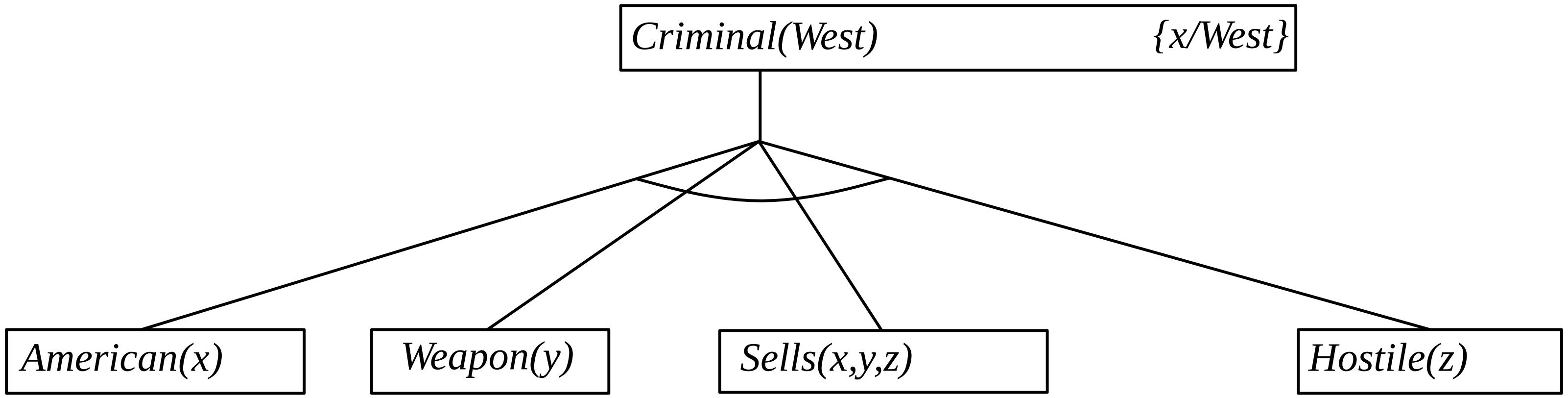
```
function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions
    inputs: KB, a knowledge base
                goals, a list of conjuncts forming a query (θ already applied)
                θ, the current substitution, initially the empty substitution { }
    local variables: answers, a set of substitutions, initially empty

    if goals is empty then return {θ}
    q' ← Subst(θ, First(goals))
    for each sentence r in KB
                where Standardize-Apart(r) = ( p1 ∧ . . . ∧ pn ⇒ q)
                and θ' ← Unify(q, q') succeeds
          new_goals ← [p1, . . . , pn|Rest(goals)]
          answers ← FOL-BC-Ask(KB, new_goals, Compose(θ', θ)) ∪ answers
    return answers
```

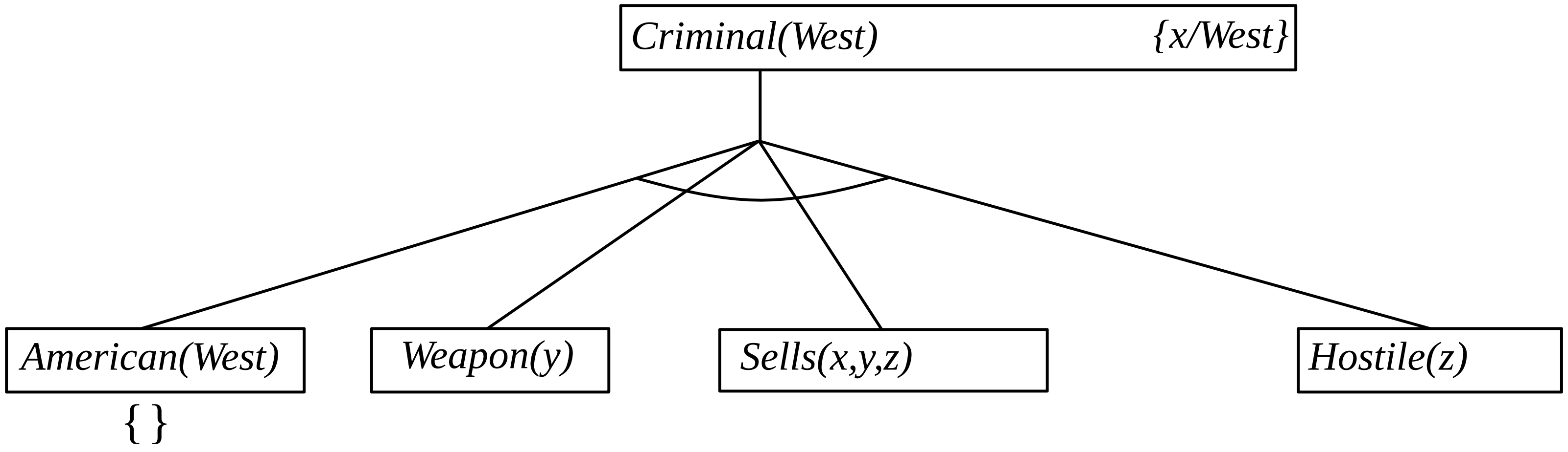
Backward chaining example

$\boxed{Criminal(West)}$

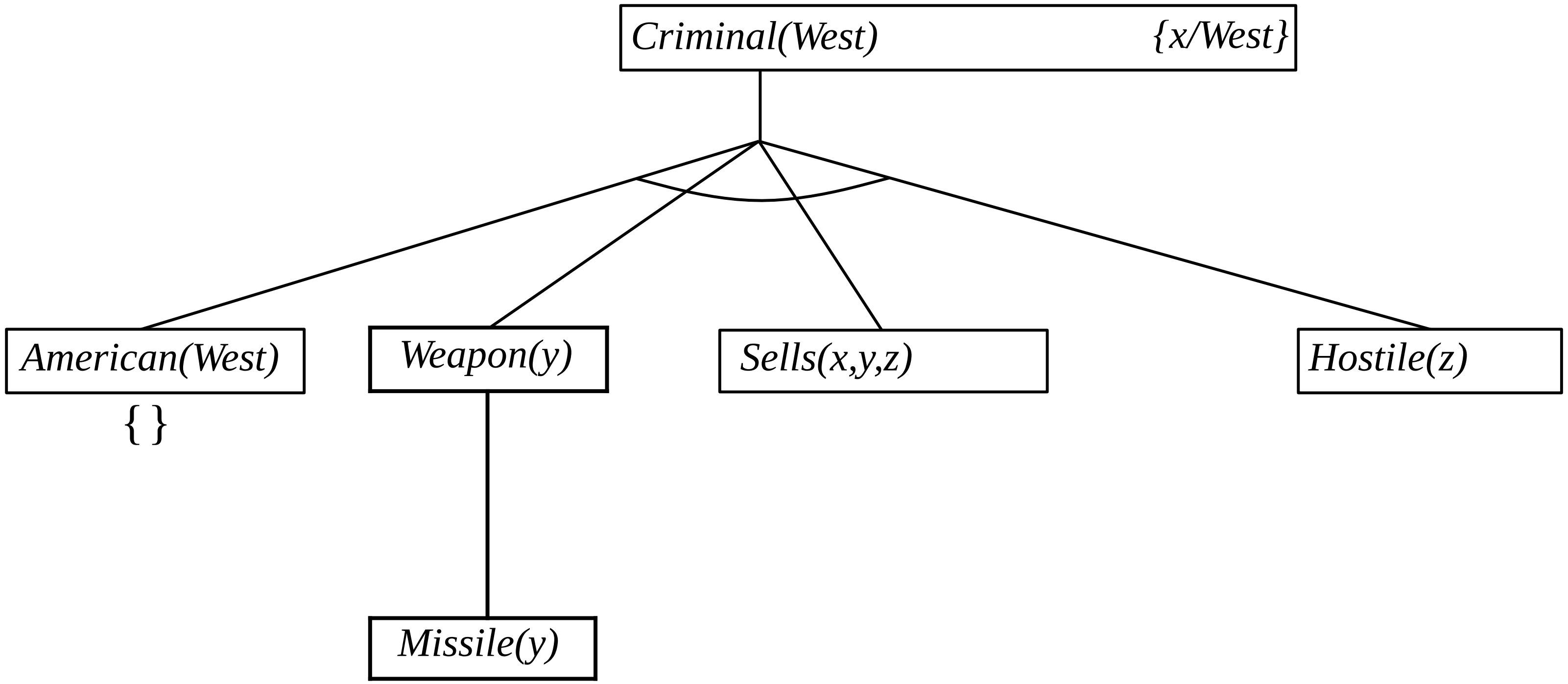
Backward chaining example



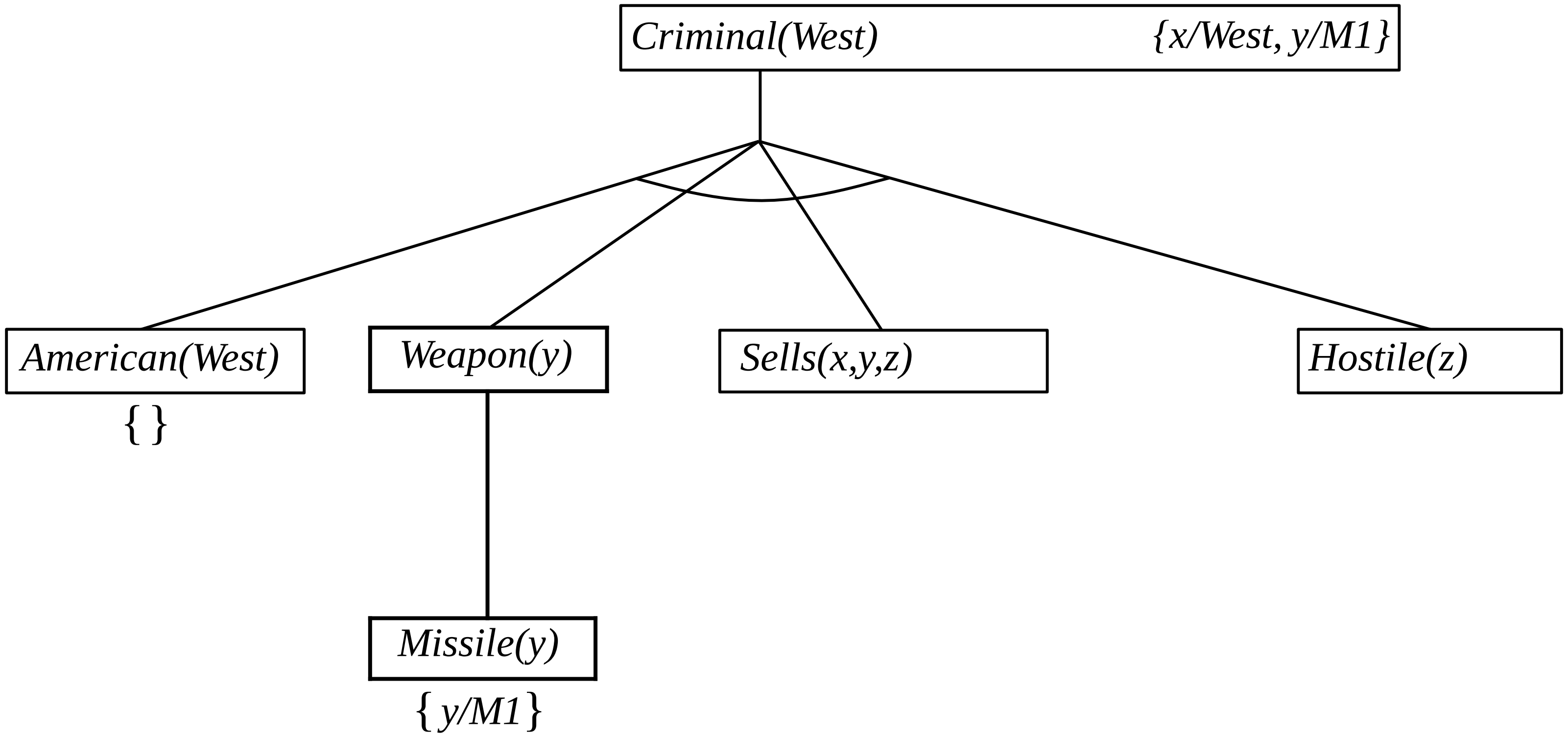
Backward chaining example



Backward chaining example

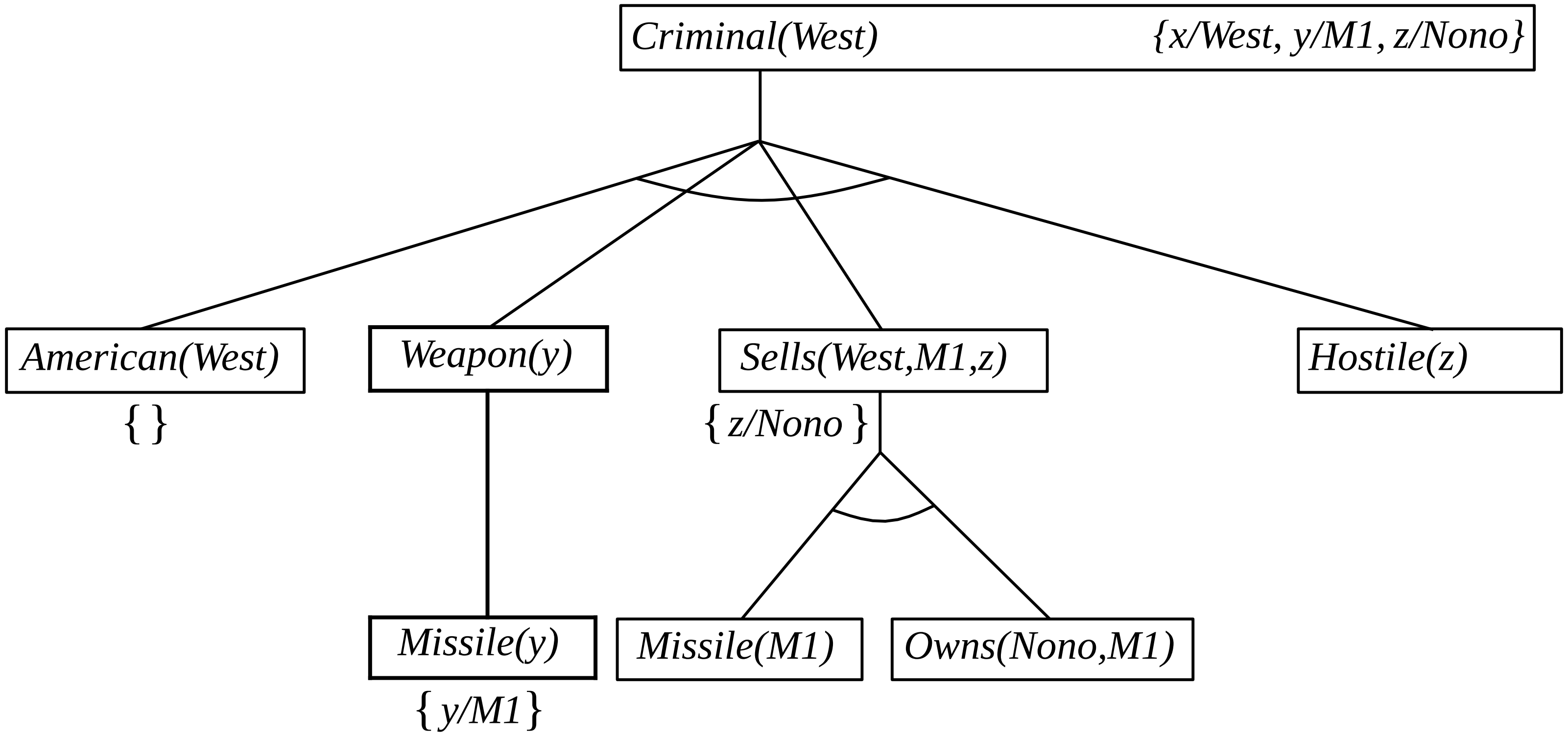


Backward chaining example

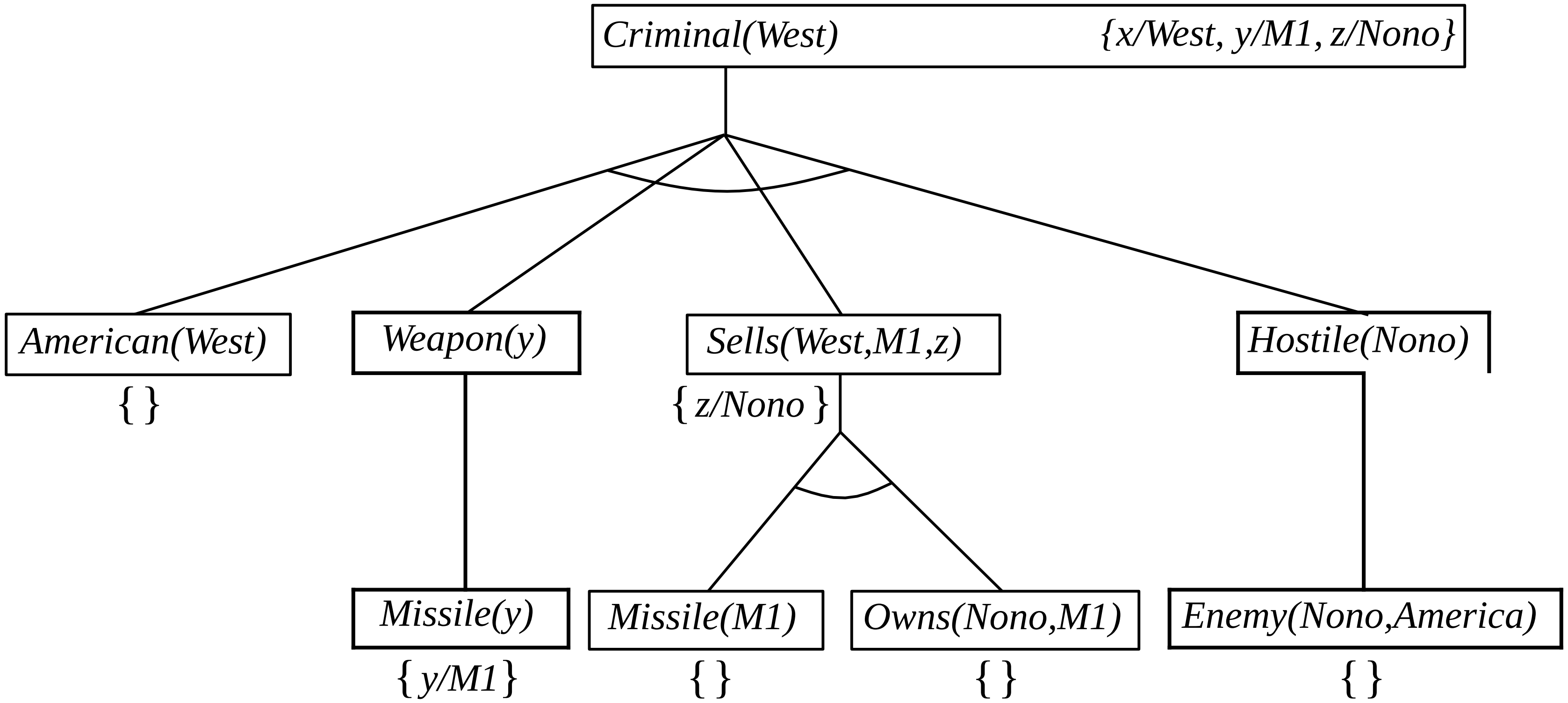


CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Backward chaining example



Backward chaining example



CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

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 (without improvements!) for logic programming

Logic programming

Sound bite: computation as inference on logical KBs

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 instance as facts	Encode problem instance as data
6. Ask queries	Apply program to data
7. Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

Conjunctive normal form for first-order logic

\>> For example,

$\forall x\ American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal\ (x)$

becomes, in CNF,

$\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x, y, z) \lor \neg Hostile(z) \lor Criminal\ (x).$

\>> Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.

Resolution: brief summary

Full first-order version:

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\mathrm{Unify}(l_i, \neg m_j) = \theta$.

For example,

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

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

Apply resolution steps to $CNF(KB \wedge \neg a)$; complete for FOL

For Example

[Animal (F(x)) ∨ Loves(G(x), x)] and [¬Loves(u, v) ∨ ¬Kills(u, v)]

unifier θ={u/G(x), v/x} to produce

[Animal (F(x)) ∨ ¬Kills(G(x), x)] .

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

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

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

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Conversion to CNF contd.

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

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

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

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

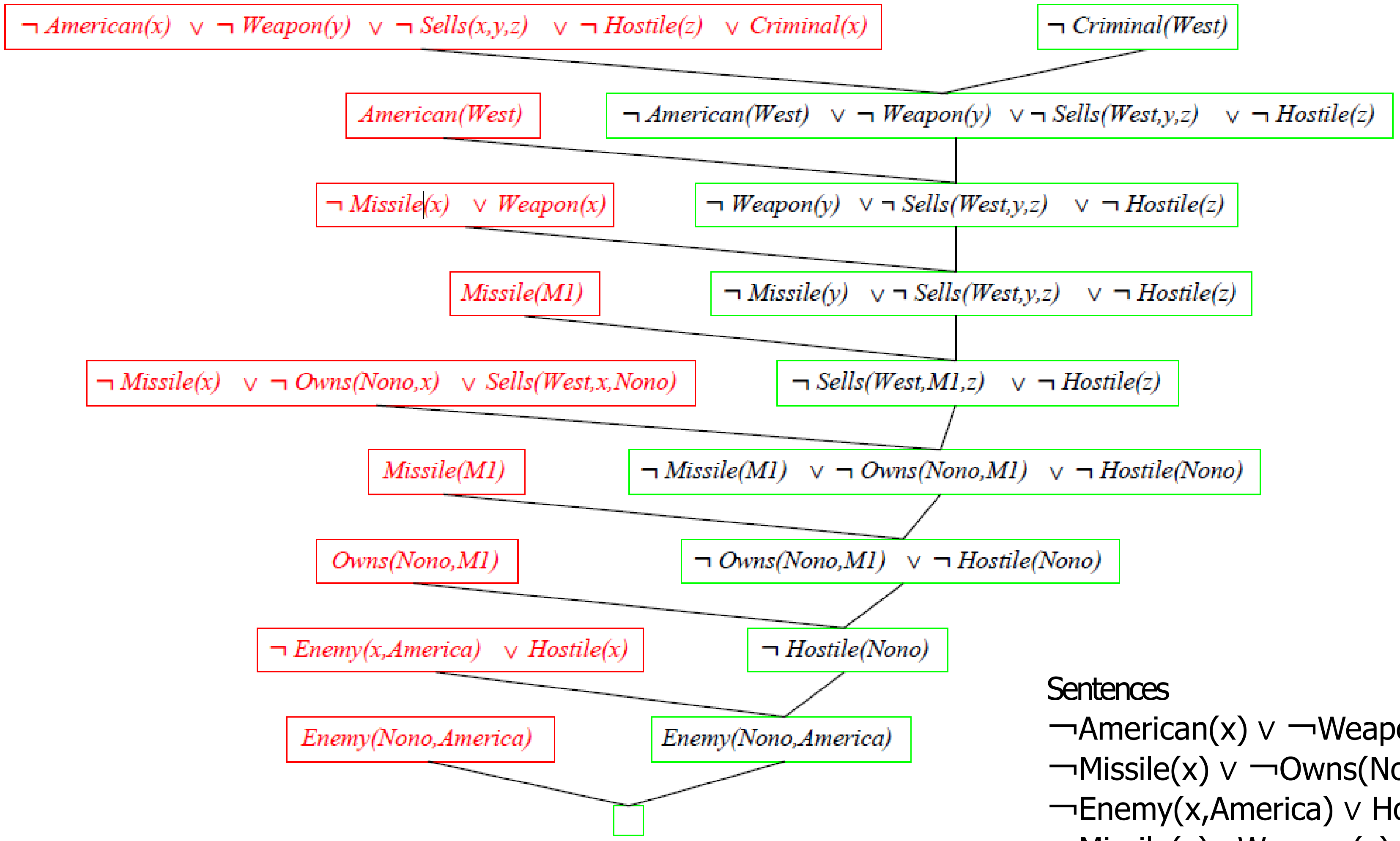
5. Drop universal quantifiers:

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

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

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

Resolution proof: definite clauses



Sentences

¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x, y, z) ∨ ¬Hostile(z) ∨ Criminal (x)
¬Missile(x) ∨ ¬Owns(Nono, x) ∨ Sells(West, x, Nono)
¬Enemy(x,America) ∨ Hostile(x)
¬Missile(x) ∨Weapon(x)
Owns(Nono,M1) Missile(M1)
American(West) Enemy(Nono,America) .

Summary

>>

References

[1] Russell, S. and Norvig, P., 2002. Artificial intelligence: a modern approach Logical Agents, Chapter 9.

[2] – Based on Lecture slides, chapter09.pdf, by Russell, S.