Artificial Intelligence
CS 6364

Professor Dan Moldovan

Section 6

Inference in First Order Logic

Outline

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

Inference in FOL

- Inference rules for quantifiers
  - Universal Instantiation
  - Existential Instantiation
- Reduction to propositional inference
- Unification and lifting

A brief history of reasoning

450B.C.	Stoics	propositional logic, inference (maybe)
322B.C.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

Universal Instantiation

- This rule says that we can infer any sentence obtained by substituting a ground term (a term without variables) for a variable
- Substitution or binding list is a set of variable/term pairs
- SUBST(θ, α) denotes the result of applying the substitution θ to sentence α
  - Example: α: $\forall x,y \;\; P(x) \Rightarrow Q(x,y) \wedge R(y)$

    $\theta = x/A$

    SUBST(θ, α) = ($\forall y \; P(A) \Rightarrow Q(A,y) \wedge R(y)$)
- **Universal Instantiation**:

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

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

$\forall x$ Young(x)$\wedge$Beautiful(x) $\Rightarrow$ Attractive(x)

SUBST(x/Robert) :
Young(Robert) $\wedge$Beautiful(Robert) $\Rightarrow$ Attractive(Robert)

Universal Instantiation (UI)

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

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

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))

$\vdots$

Existential instantiation (EI)

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

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

- E.g.,
  ∃x Crown(x) ∧ OnHead(x,John) yields:

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

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

Existential & Universal Instantiation

- Universal Instantiation (UI) can be applied several times to **add** new sentences to the KB
  - The new KB' is logically equivalent to KB (the old one)

- Existential Instantiation (EI) can be applied once to **replace** the existential sentence
  - The new KB' is not logically equivalent to the old one, but it is satisfiable iff the old KB was satisfiable.

Logical equivalence: two sentences α and β are logically equivalent *if they are true in the same set of models.* $\alpha \equiv \beta$ if $\alpha \models \beta$ and $\beta \models \alpha$

Validity: a sentence is valid if it is true in all models. Valid sentences are also known as *tautologies.*

Satisfiability: a sentence is satisfiable if it is true in some model.

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 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: 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 continued

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

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 nonentailed sentence.)

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 pn^k instantiations.

Generalized Modus Ponens

For sentences p_i , p_i' and q where there is a substitution θ such that

SUBST (θ, p_i') = SUBST (θ, p_i), for all i,

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

p_1' is *King (John)*
p_2' is *Greedy (y)*
θ is {*x/John, y/John*}
SUBST (θ,q) is *Evil (John)*

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

Generalized Modus Ponens

Generalized Modus Ponens is a sound inference rule

Proof:

$p \models \text{SUBST}(\theta, p)$ holds and from $p_1', p_2', \ldots, p_n'$ we infer

$\text{SUBST}(\theta, p_1') \wedge \ldots \wedge \text{SUBST}(\theta, p_n')$

and from the implications $p_1 \wedge \ldots \wedge p_n \Rightarrow q$ infer

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

θ in Generalized Modus Ponens is defined so that $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$, for all i; $\text{SUBST}(\theta, q)$ follows by Modus Ponens.

Generalized Modus Ponens is a **lifted** version of 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

UNIFY(α,β) $= \theta$ if $\alpha\theta = \beta\theta$ *or SUBST (θ , α) = SUBST (θ , β)*

α	β	θ
Knows(John,x)	*Knows(John,Jane)*	
Knows(John,x)	*Knows(y,OJ)*	
Knows(John,x)	*Knows(y,Mother(y))*	
Knows(John,x)	*Knows(x,OJ)*	

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

UNIFY(α,β) $= \theta$ if $\alpha\theta = \beta\theta$ *or SUBST (θ , α) = SUBST (θ , β)*

α	β	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)*

Unification Example

- *Suppose we have a rule*

  Knows(John,x) ⇒ Hates(John,x)

  "John hates everyone he knows"

?? We want to use this rule with the Modus Ponens inference rule to find whom he hates (x=?)

how?

We need to find those sentences in the KB that Unify with Knows(John,x) and then apply the unifier to Hates(John,x)

Working the Unification

- Let the KB contain:
  - Knows(John, Jane)
  - Knows(y, Mother(y))
  - Knows(y, Leonid)
  - Knows(x, Elisabeth)

- Unifying the antecedent of the rule Knows(John,x) $\Rightarrow$ Hates(John,x) against each of the sentences in the KB:

UNIFY(Knows(John,x),Knows(John,Jane))={x/Jane}
UNIFY(Knows(John,x),Knows(y,Leonid))=
{x/Leonid, y/John}
UNIFY(Knows(John,x),Knows(y,Mother(y)))=
{y/John, x/Mother(John)}
UNIFY(Knows(John,x),Knows(x,Elisabeth))=fail

Most General Unifier

- 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 }*

Most General Unifier (MGU)

- MGU is the substitution that makes the least commitment about the binding of the variables
- For example

UNIFY(Knows(John,x), Knows(y,z)) = {y/John, x/z}
or {y/John, x/z, w/Treda}
or {y/John, x/John, z/John}
or ...

MGU = {y/John, x/z}

Forward Chaining

<u>The idea:</u> Start with the atomic sentences in the KB and apply Modus Ponens in the forward direction, adding new atomic sentences until no further inferences can be Made.

Situation: 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.

- What do we want to prove?
  $\Rightarrow$ West is a criminal

FOL representation

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

P.1 $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Hostile}(z) \wedge \text{Sells}(x,y,z) \Rightarrow \text{Criminal}(x)$

"... Nono...has some missiles":

P.2 $\text{Owns}(\text{Nono},M_1)$
$\text{Missile}(M_1)$

"All of its missiles were sold by Colonel West":

P.3 $\text{Missile}(x) \wedge \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono})$

Common sense knowledge

We need also to know that missiles are weapons:

P.4 $\forall x$ Missile(x) $\Rightarrow$ Weapons(x)

An enemy of America counts as "hostile"

P.5 $\forall x$ Enemy(x,America) $\Rightarrow$ Hostile(x)

"...West who is American..."

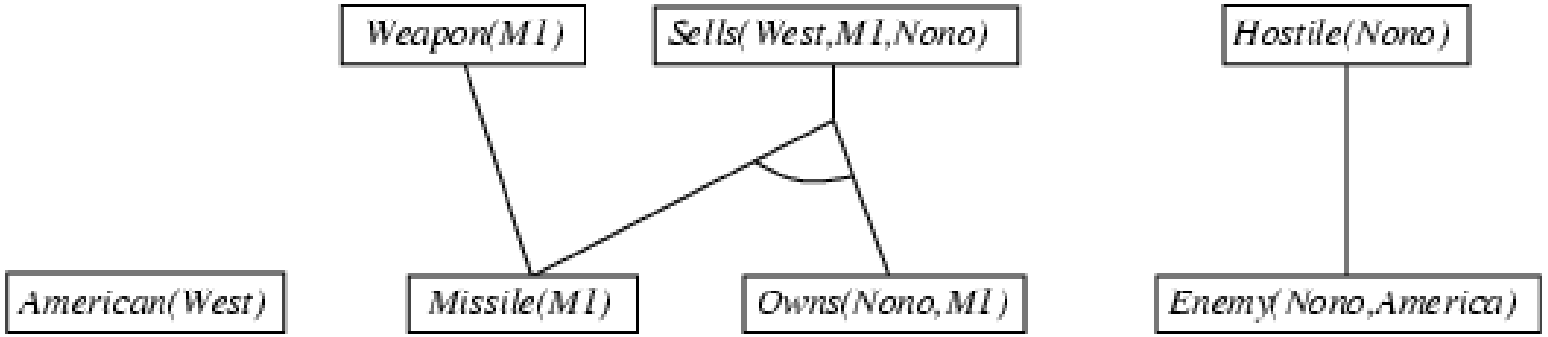
P.6 American(West)

"... Nono, an enemy of America..."

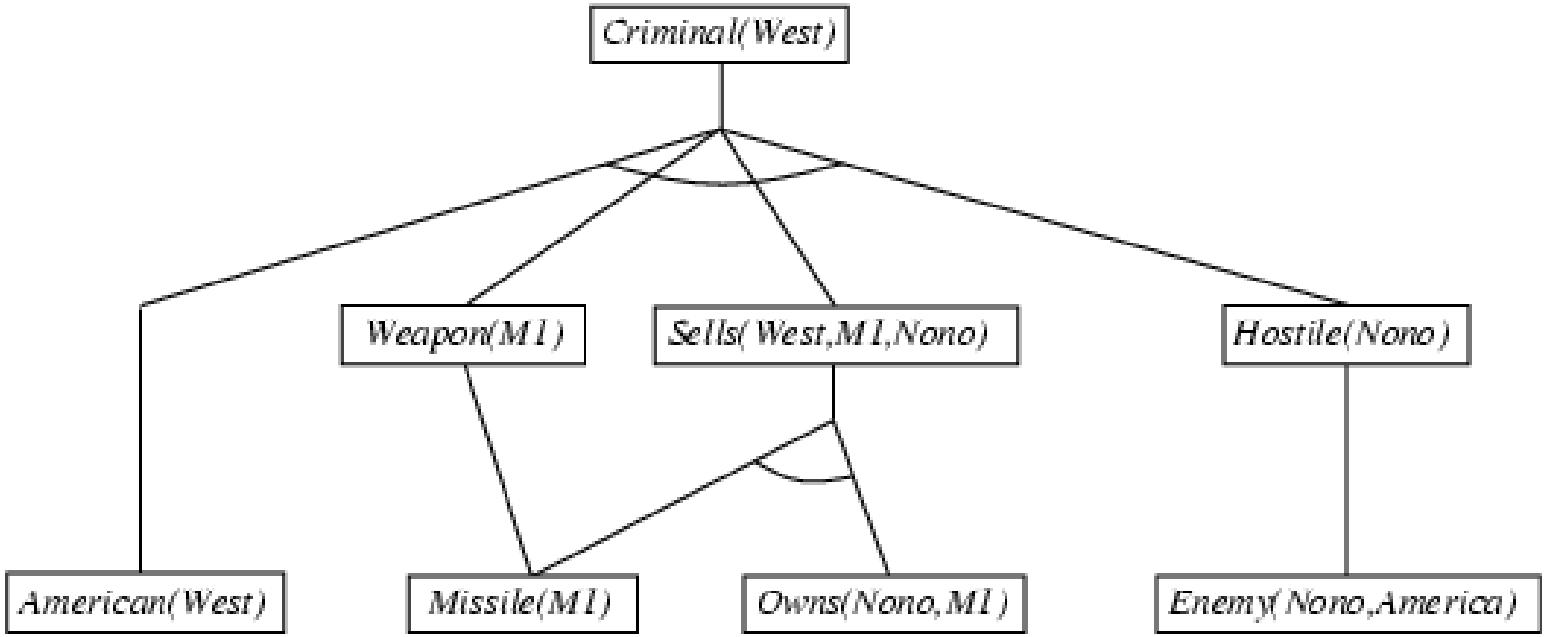
P.7 Enemy(Nono, America)

Forward chaining proof

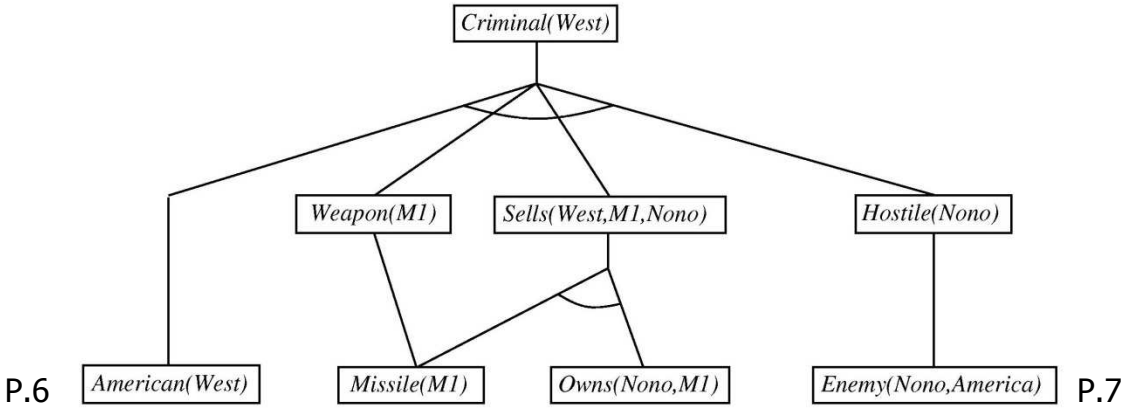
Forward chaining proof



Forward chaining proof



The proof Tree



First iteration: the implication sentences

← Has unsatisfied Premises in first iteration On the second iteration It is satisfied with x/West Y/M1, z/Nono

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

P.1 American(x) $\wedge$ Weapon(y) $\wedge$ Hostile(z) $\wedge$ Sells(x,y,z) $\Rightarrow$ Criminal(x)

"All of its missiles were sold by Colonel West":

P.3 Missile(x) $\wedge$ Owns(Nono,x) $\Rightarrow$ Sells(West,x,Nono)

We need also to know that missiles are weapons:

P.4 $\forall$x Missile(x) $\Rightarrow$ Weapons(x)

An enemy of America counts as "hostile"

P.5 $\forall$x Enemy(x,America) $\Rightarrow$ Hostile(x)

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*

$\Rightarrow$ match each rule whose premise contains a newly added positive literal

Matching itself can be expensive:

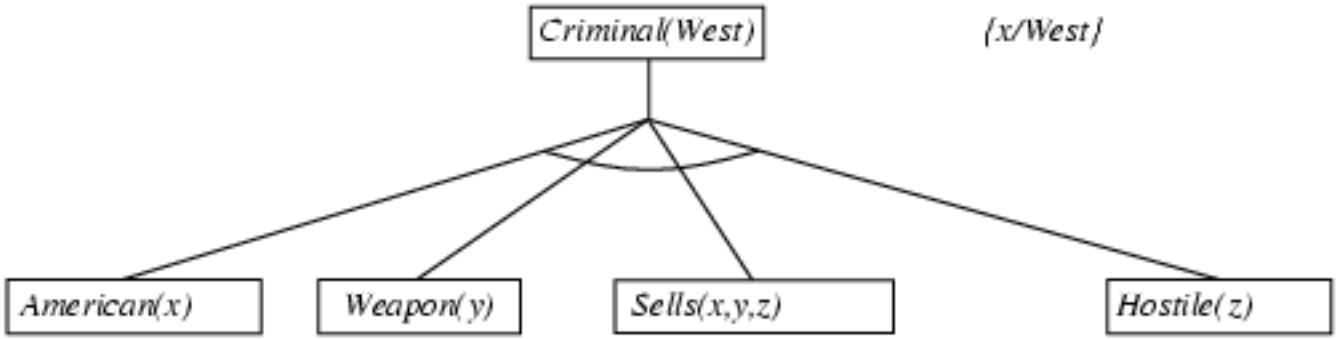
Database indexing allows O(1) retrieval of known facts

- e.g., query *Missile(x)* retrieves *Missile(M_1)*

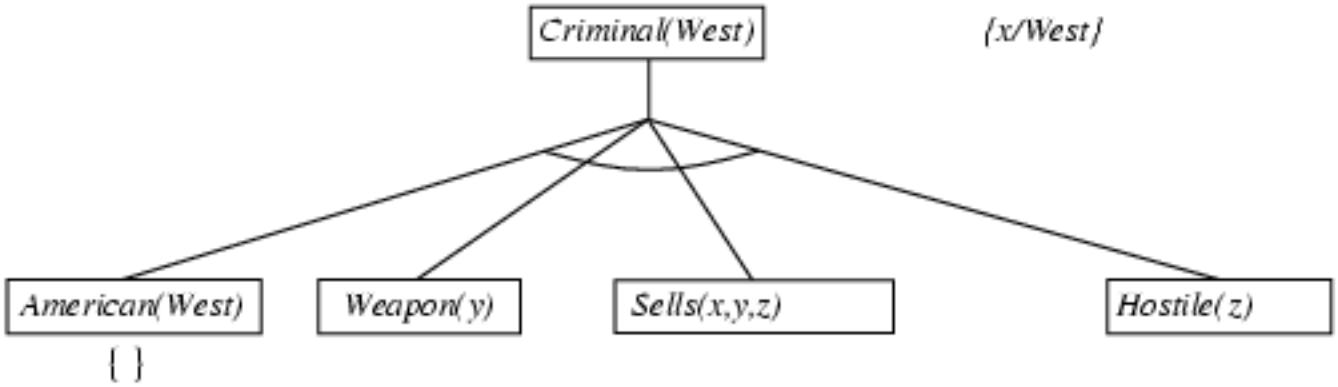
Forward chaining is widely used in deductive databases

Backward chaining example

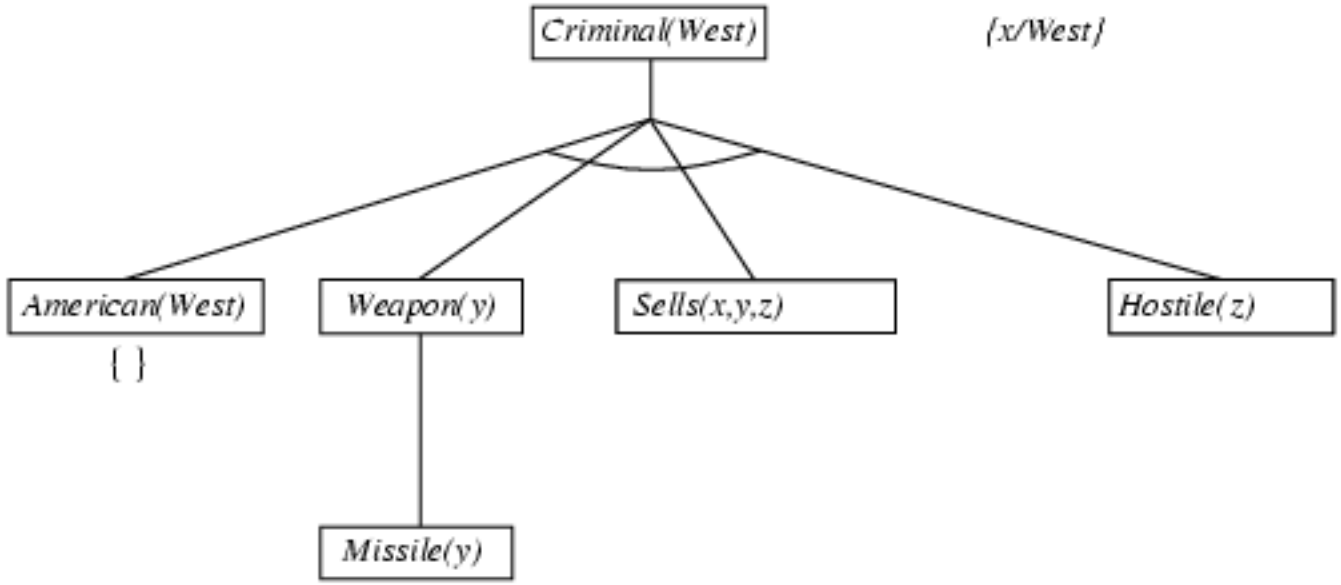
Backward chaining example



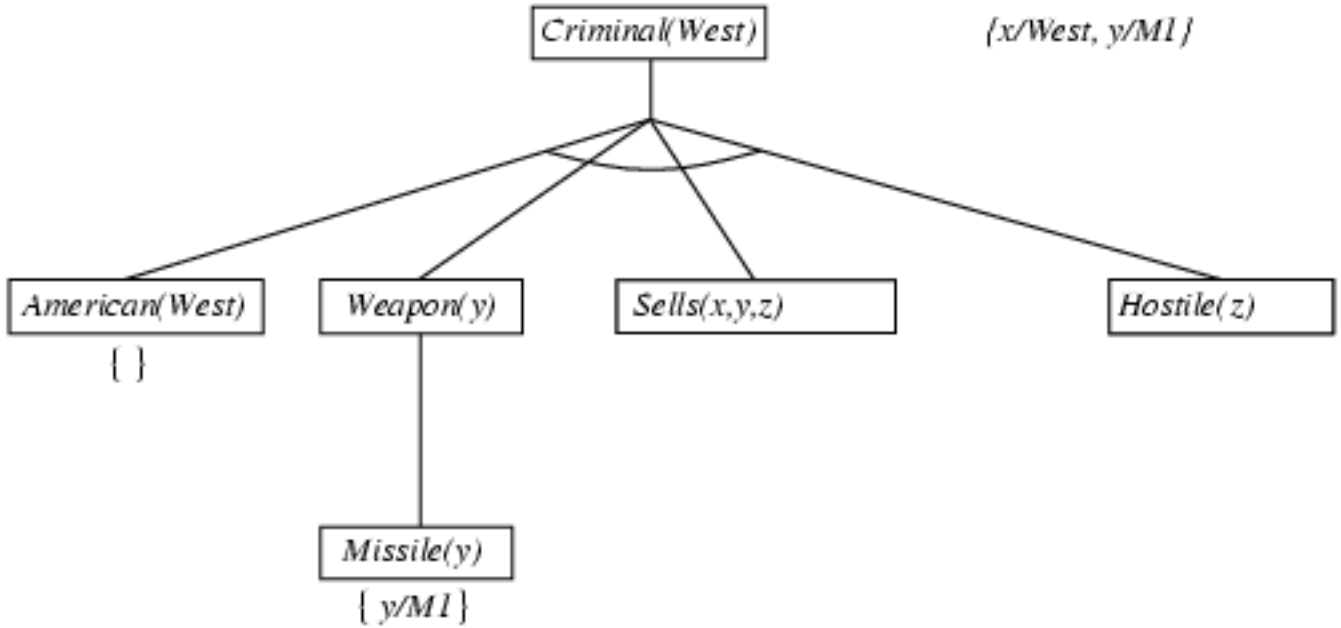
Backward chaining example



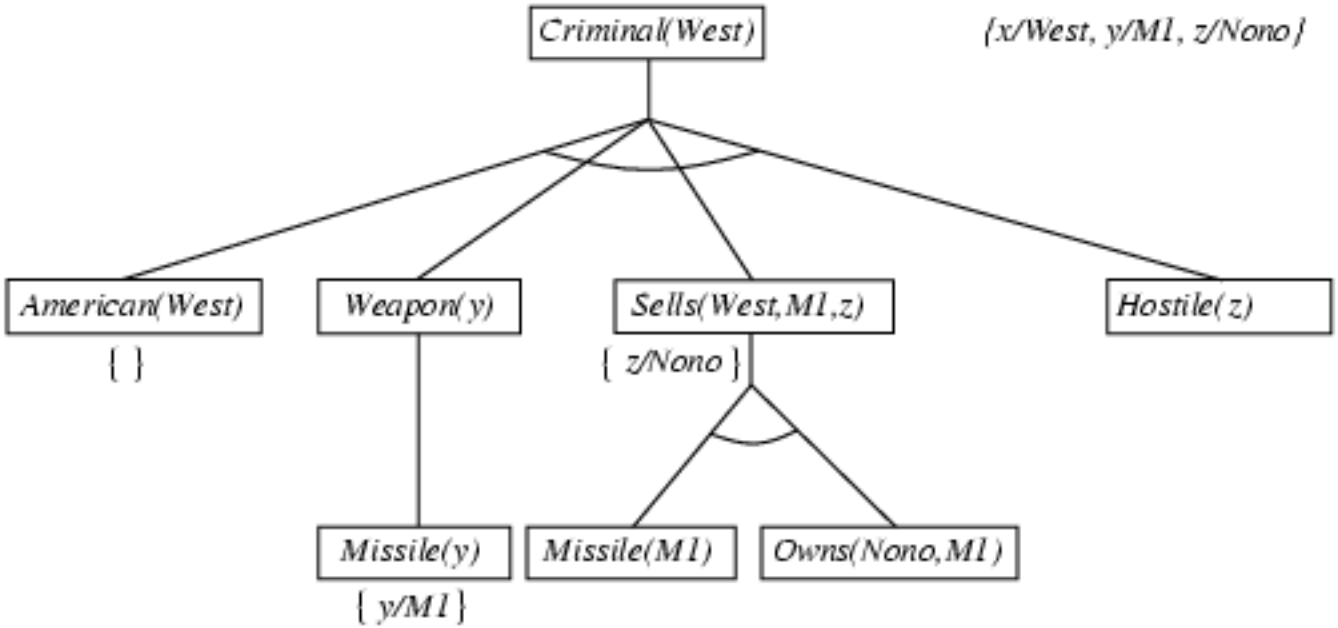
Backward chaining example



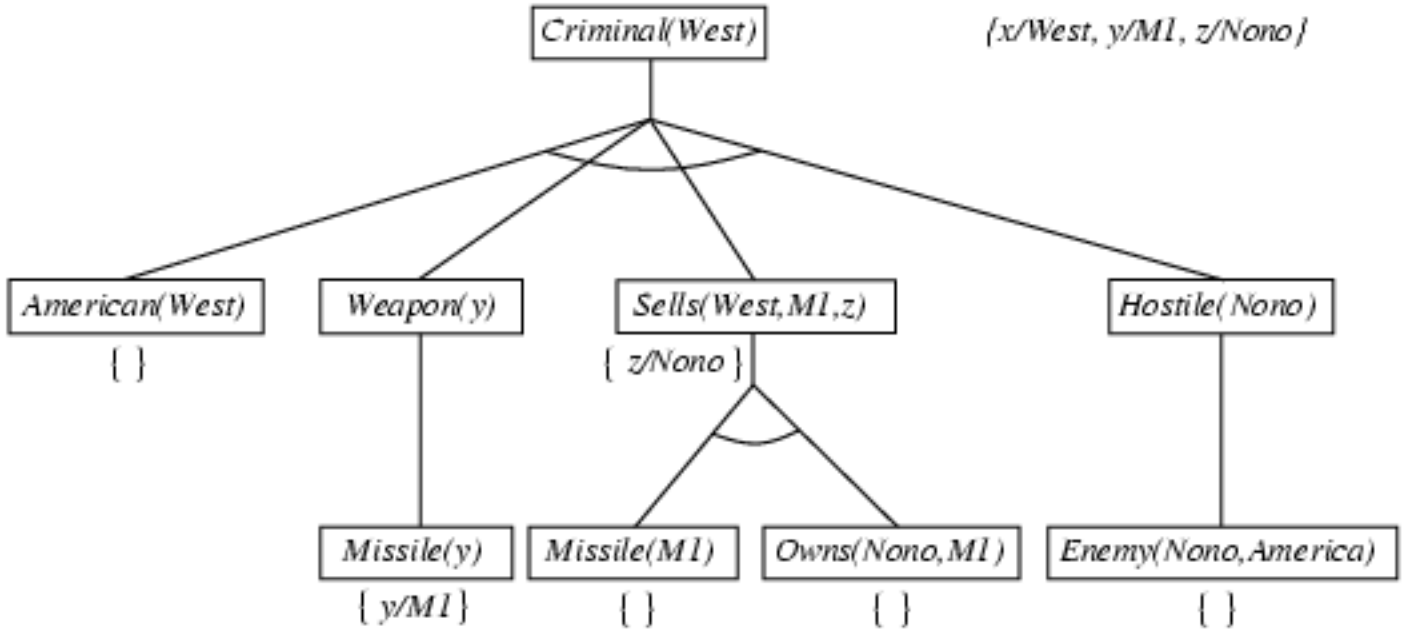
Backward chaining example



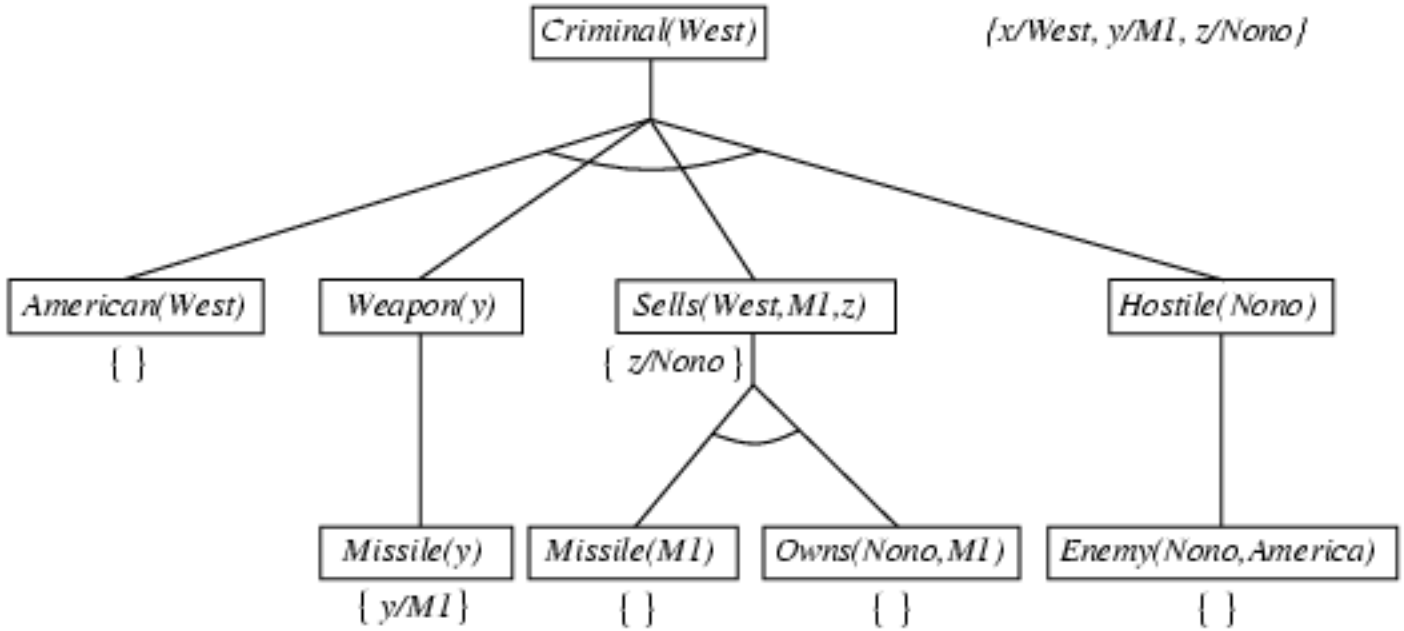
Backward chaining example



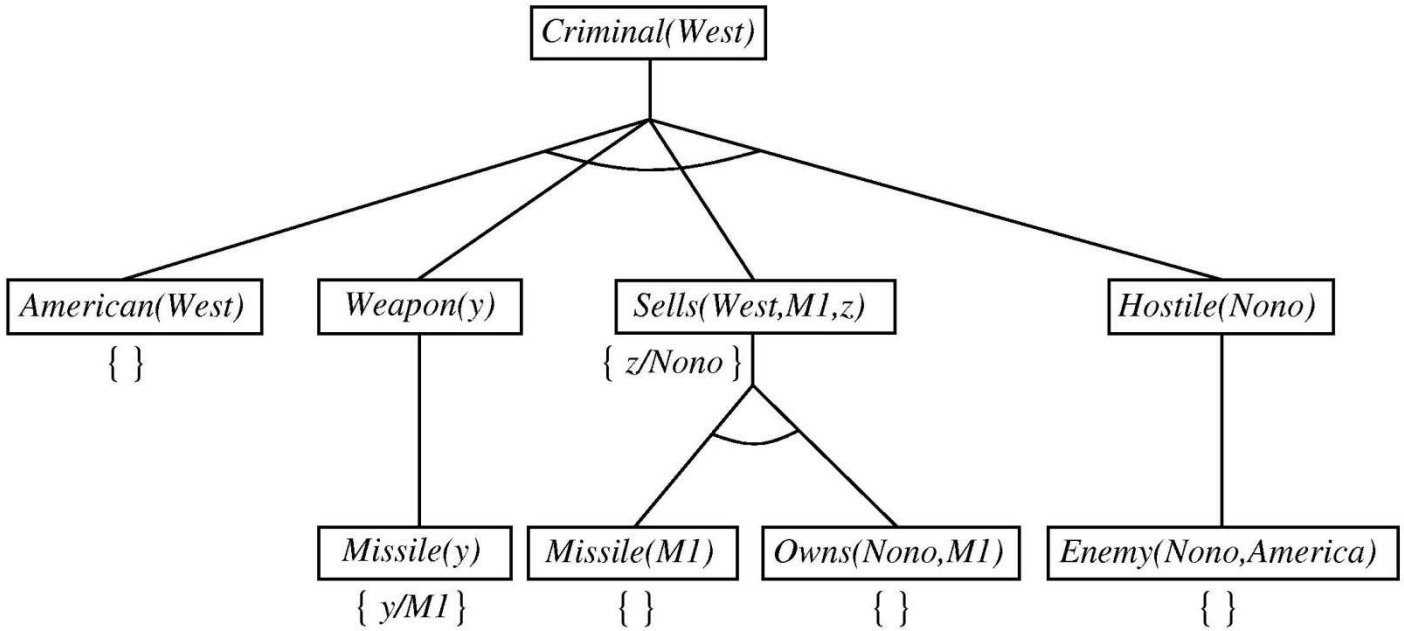
Backward chaining example



Backward chaining example



Backward Chaining Proof Tree



The tree should be read depth-first, left-to-right
Note: once one sub-goal in a conjunction succeeds, its substitution is applied to subsequent goals.

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

Resolution Strategy

- We know that repeated applications of the inference rule will find the proof if one exists → what about the efficiency of this process?
- We look back at 4 strategies used to guide the search for the proof

Unit preference ⇒ prefer solutions where one of the sentences is a single literal (a unit clause)

Why? For the proof, we try to produce a very short sentence

True ⇒ False

Increases the speedup, but not the branching factor!

It might be a good idea to prefer inferences that produce shorter sentences

Resolution Strategies

- Strategies that help find proofs efficiently.
- Unit Preference
  - Prefers sentences that are a single literal (unit clauses)
  - The idea is that we are trying to produce an empty clause, so it might be a good idea to prefer inferences that produce shorter clauses
  - For example, resolving a unit sentence A with any other sentence $B \vee \neg A \vee C$ always yields a shorter clause: $B \vee C$

Set of Support

- Preferences that try certain resolutions first are helpful. It is more effective to try to eliminate some potential resolutions altogether.
- *The set of support* is a sub-set of sentences that are combined with other sentences and the resolvent is added to the set of support.
- If we chose a set of support such that the remainder of the sentences are jointly satisfiable, then set-of-support resolution will be complete.
- Common approach: use the negated query as the set of support, on the assumption that the original knowledge base is consistent.

Input Resolution

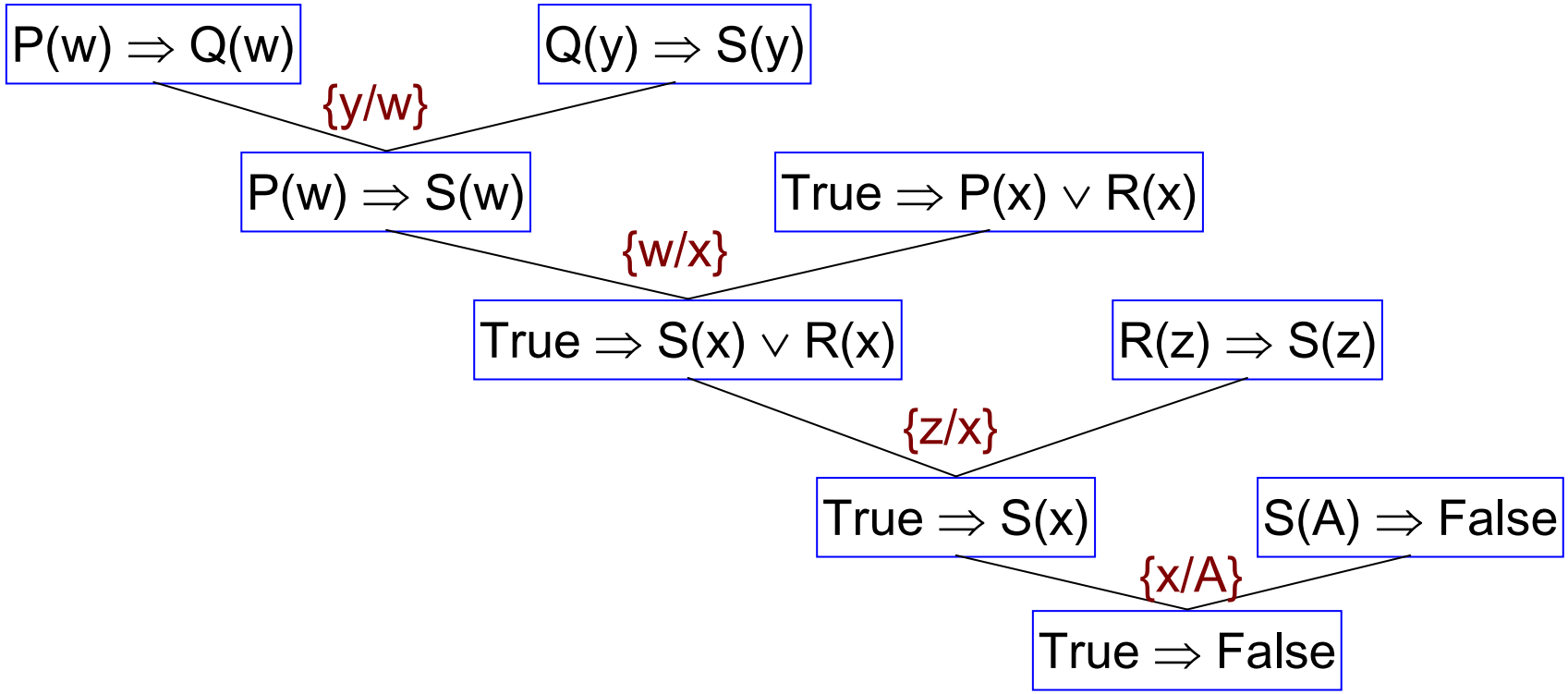
Every resolution combines one of the input sentences (from the KB or the query) with some other sentence.

- Subsumption

Eliminates all sentences that are subsumed by an existing sentence in the KB. For example, if P(x) is in the KB, then there is no sense in adding P(A) and even less sense in adding P(A) $\vee$ Q(B). It helps keep the KB small, and thus helps keep the search space small.

Input Resolution

- Every resolution combines one of the input sentences with some other sentences (from the KB or the query)



Demodulation Rule → the other way of dealing with equality

- Definition: → Informally:

- For equality x = y
+ any sentence with a nested term that unifies x,
it derives the same sentence with y substituted
for the nested term

Formally:

$\forall x,y,z$ where UNIFY(x,z) = θ:

x = y, (...z ...)

(...SUBST(θ,y)...)

Subsumption

- This method eliminates all sentences that are Subsumed by an existing sentence in the KB

  → These sentences are more specific than those in the KB

- Example: $P(x) \in KB$

  $\Rightarrow$ P(A) should not be added

  $P(A) \vee Q(B)$ should not be added!

- Subsumption keeps the KB small

Theorem Provers

- **OTTER** (Organized Techniques for Theorem-proving and Effective Research) (McCune 1992)
- In preparing a problem for Otter, the user must divide the knowledge into four parts:
  1. A set of clauses known as the set of support (SoS) which define the important facts about the problem. Every resolution step resolves a member of the set of support against another axiom, so the search is focused on the set of support.
  2. A set of usable axioms that are outside the set of support. These provide background knowledge about the problem area. The boundary between what is part of the problem and what is background (thus in usable axioms) is up to the user's judgment.
  3. A set of equations known as rewrites or demodulators.
  4. A set of parameters and clauses that defines the control strategy. The user specifies a *heuristic function* to control the search and a *filtering function* to eliminate some subgoals as un-interesting.

```
procedure OTTER(sos, usable)
   inputs: sos, a set of support—clauses defining the problem (a global variable)
           usable, background knowledge potentially relevant to the problem

   repeat
      clause ← the lightest member of sos
       move clause from sos to usable
      PROCESS(INFER(clause, usable), sos)
   until sos = [ ] or  a refutation has been found
```

```
function INFER(clause, usable) returns clauses

   resolve clause with each member of usable
   return the resulting clauses after applying FILTER
```

```
procedure PROCESS(clauses, sos)

   for each clause in clauses do
      clause ← SIMPLIFY(clause)
      merge identical literals
      discard clause if it is a tautology
      sos ← [clause–sos]
      if clause has no literals then a refutation has been found
      if clause has one literal then look for unit refutation
```