Set 8:
Inference in First-order logic

ICS 271 Fall 2017

Chapter 9: Russell and Norvig

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., $\forall x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$ yields:

$King(John) \land Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$
$King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John))$

Obtained by substituting {x/John}, {x/Richard} and {x/Father(John)}

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:

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

  provided C_1 is a new (not used so far) constant term, called a Skolem constant

- Skolemization : ∃ elimination
  - ∀x ∃y Loves(y,x)
  - Incorrect inference : ∀x Loves(A,x) – y may be different for each x
  - Correct inference : ∀x Loves(f(x),x)

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.

- 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 contd.

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 (will terminate) if α is entailed, loops forever 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 King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
  King(John)
  $\forall$y Greedy(y)
  Brother(Richard,John)

- Given query "Evil(x) 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.

Generalized Modus Ponens (GMP)

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

King(John), Greedy(y), (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*)
θ is {x/John,y/John} q is *Evil*(*x*)
q θ 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 sentence *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

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)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ
  - note : replace variables with terms!

p	q	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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},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)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},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)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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))	
Knows(John,x)	Knows(x,OJ)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},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)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},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)*

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

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)	∅

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

- 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 }

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
    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

The unification algorithm

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

Unification

- Basic task : unify
  - $p_1, p_2, \ldots, p_n$
  - $q_1, q_2, \ldots, q_n$
- Proceed left to right, carry along current substitution θ
- Compare p_i with q_i,
  - predicates must match
  - apply existing substitution
  - unify instantiated pair, producing θ_i
  - add new substitution to existing $\theta = \theta \cup \theta_i$
  - occurs check

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, cont.

... 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,M_1*) and Missile(*M_1*)*

… 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)

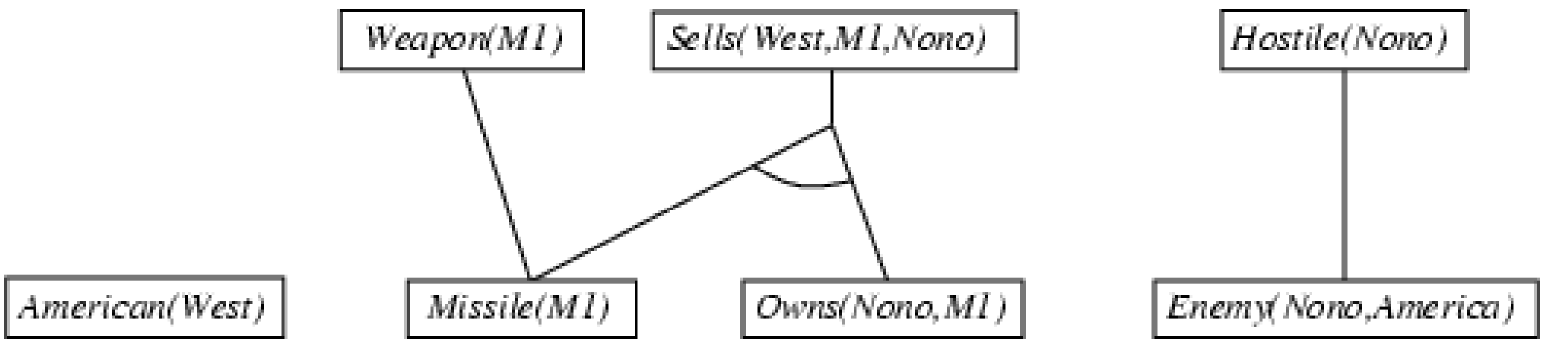
Forward chaining algorithm

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**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        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

Forward chaining proof

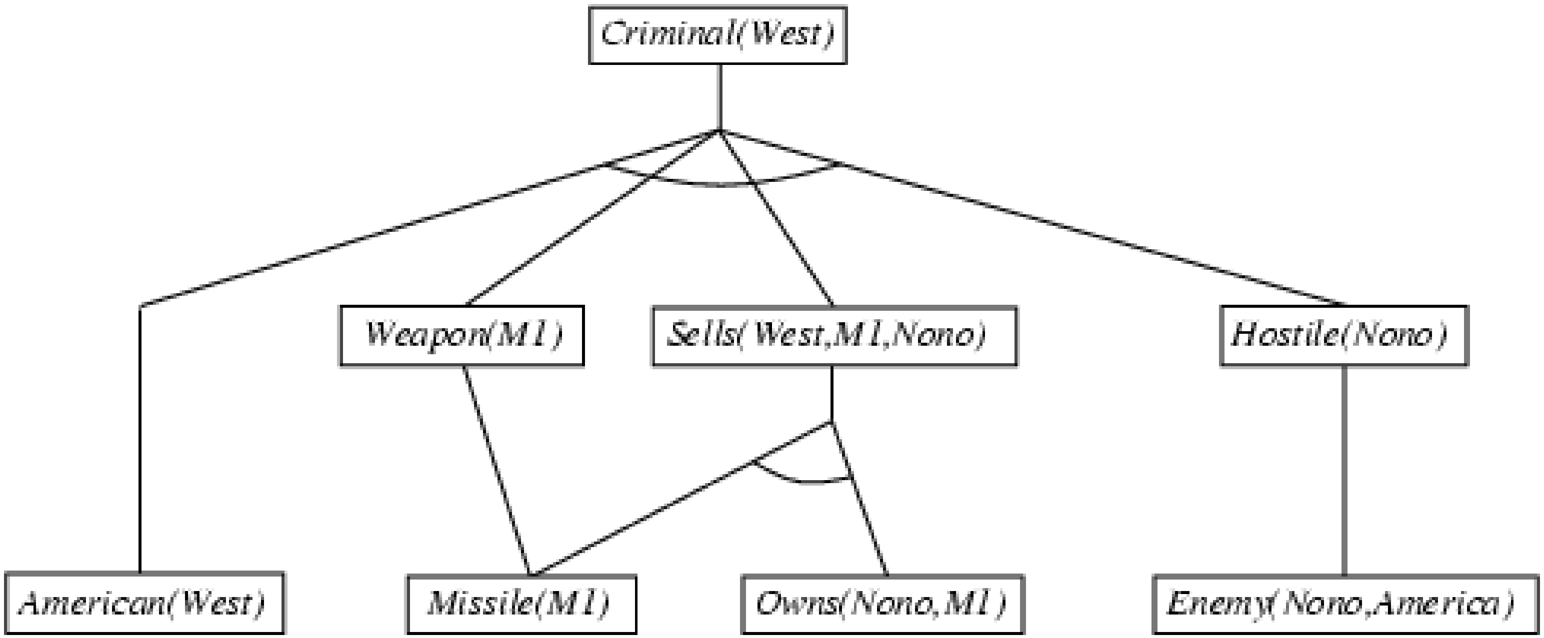


Enemy(x,America) ⇒ Hostile(x)

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

Missile(x) ⇒ Weapon(x)

Forward chaining proof



American(x) $\wedge$ *Weapon(y)* $\wedge$ *Sells(x,y,z)* $\wedge$ *Hostile(z)* $\Rightarrow$ *Criminal(x)*

Calendar

November 2017						
S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

December 2017						
S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

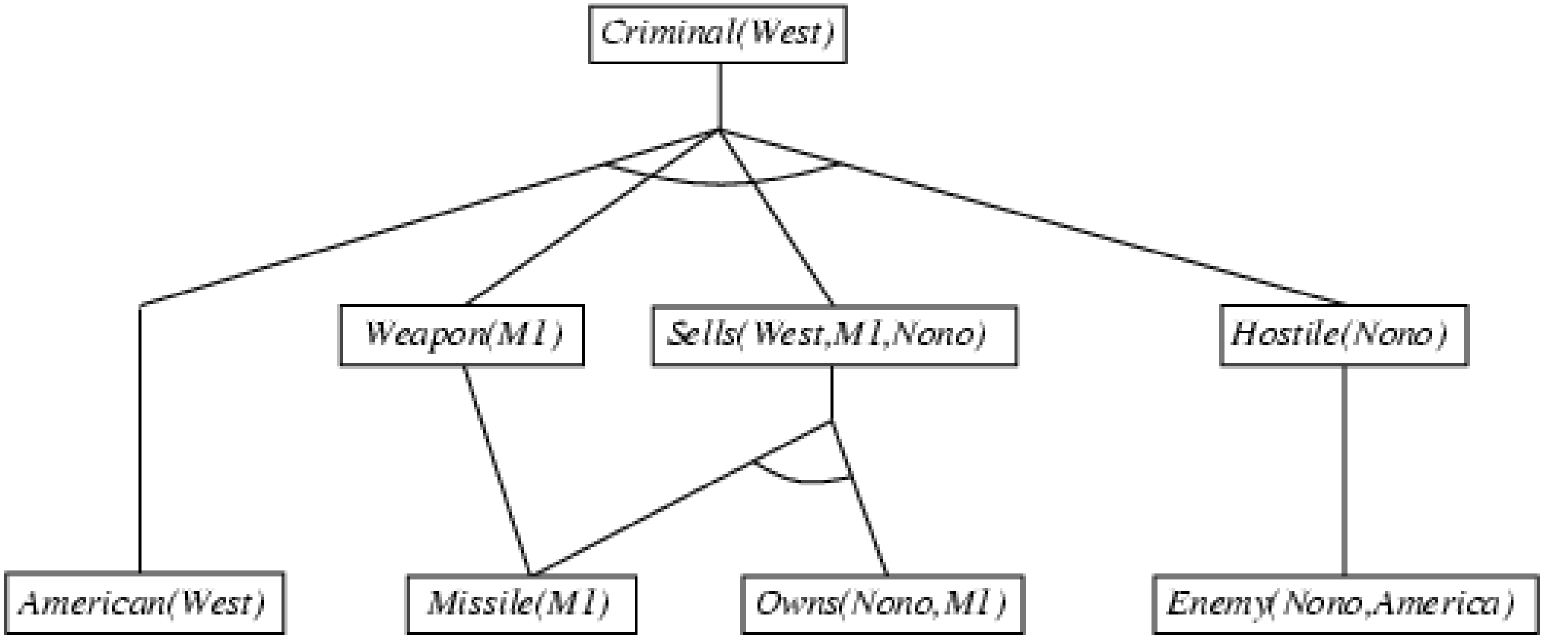
Summary so far

- Reduction by propositionalization
  - Eliminate ∀ and ∃
  - With fn symbols infinitely many ground terms
  - Theorem: If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB
  - Semi-decidable
  - Very slow in practice
- Generalized Modus Ponens
  - Inference with definite clauses
  - Replace instantiation step with unification

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i\theta \text{ for all } i$$

- UNIFY(p,q)=θ where SUBST(θ,p)=SUBST(θ,q)
- Forward chaining in FOL : lifted version of FC in PL

Forward chaining proof



- *American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
- *Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
- *Missile(x) ⇒ Weapon(x)*
- *Enemy(x,America) ⇒ Hostile(x)*
- *Owns(Nono,M1) and Missile(M1)*
- *American(West)*
- *Enemy(Nono,America)*

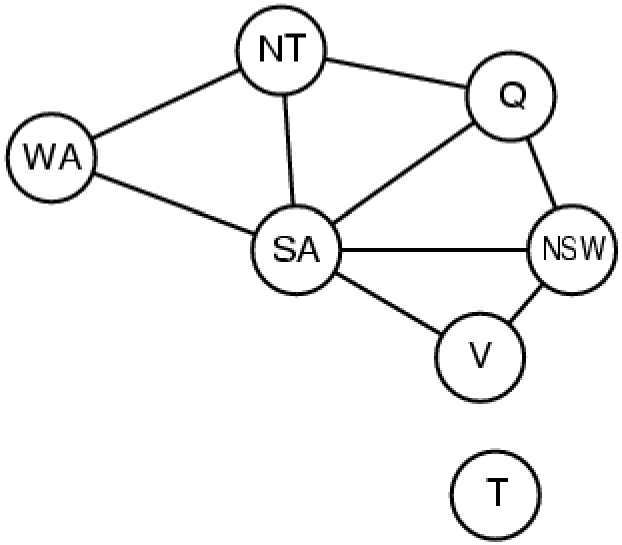
Properties of forward chaining

- Forward chaining is widely used in deductive databases

- Sound and complete for first-order definite clauses
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable

- Datalog = first-order definite clauses + no functions
  - FC terminates for Datalog in finite number of iterations ($p \cdot n^k$ ground terms)

Matching facts against rules : Hard matching example



Diff(wa,nt) ∧ *Diff(wa,sa)* ∧ *Diff(nt,q)* ∧ *Diff(nt,sa)* ∧ *Diff(q,nsw)* ∧ *Diff(q,sa)* ∧ *Diff(nsw,v)* ∧ *Diff(nsw,sa)* ∧ *Diff(v,sa)* ⇒ *Colorable()*

Diff(Red,Blue) *Diff (Red,Green)*
Diff(Green,Red) *Diff(Green,Blue)*
Diff(Blue,Red) *Diff(Blue,Green)*

- ***Colorable*() is inferred iff the CSP has a solution**
- **CSPs include 3SAT as a special case, hence matching is NP-hard**
- Query complexity vs. data complexity

Efficiency of forward chaining

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i\,\theta \text{ for all } i$$

- Pattern matching itself can be expensive:
  - Use indexing to unify sentences that have a chance of unifying
    - Knows(x,y) vs Brother(u,v)
  - Database indexing allows O(1) retrieval of known facts
  - e.g., query *Missile(x)* retrieves *Missile*(M_1)

Efficiency of forward chaining

$$\frac{p_1', p_2', \ldots , p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i\,\theta \text{ for all } i$$

- Matching rules against known facts

Conjunct ordering problem
Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West, x, Nono)

NP-hard in general, but can use heuristics used for CSPs

Rule-matching tractable when CSP is tractable

Efficiency of forward chaining

1. 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

2. Retain partial matches and complete them incrementally as new facts arrive

Efficiency of forward chaining

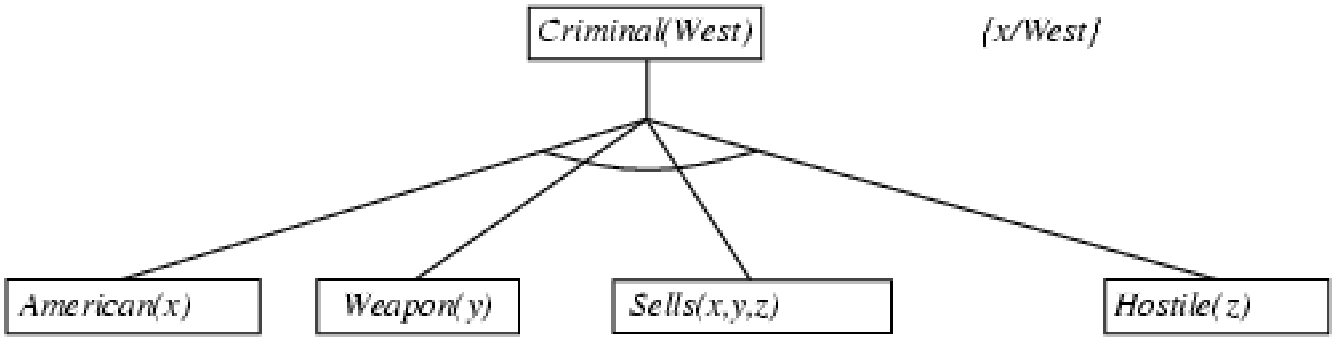
Forward chaining infers everything, most of which can be irrelevant to the goal

- Solution : allow only those bindings that are relevant to the goal
  - Use generic backward chaining
- Add Magic(x) extra conjunct to rules and Magic(c) to the KB
  - E.g. Magic(West)

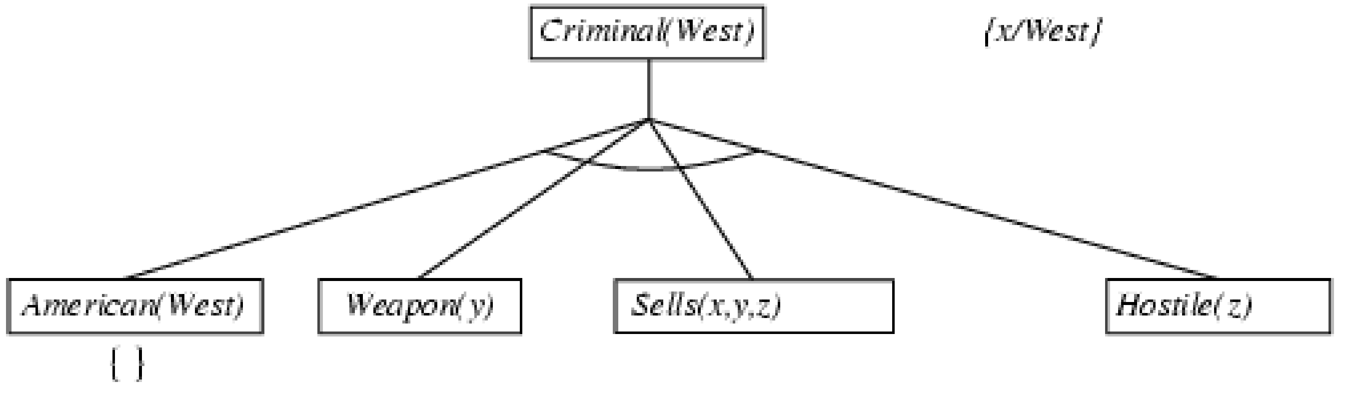
Backward chaining example

$Criminal(West)$

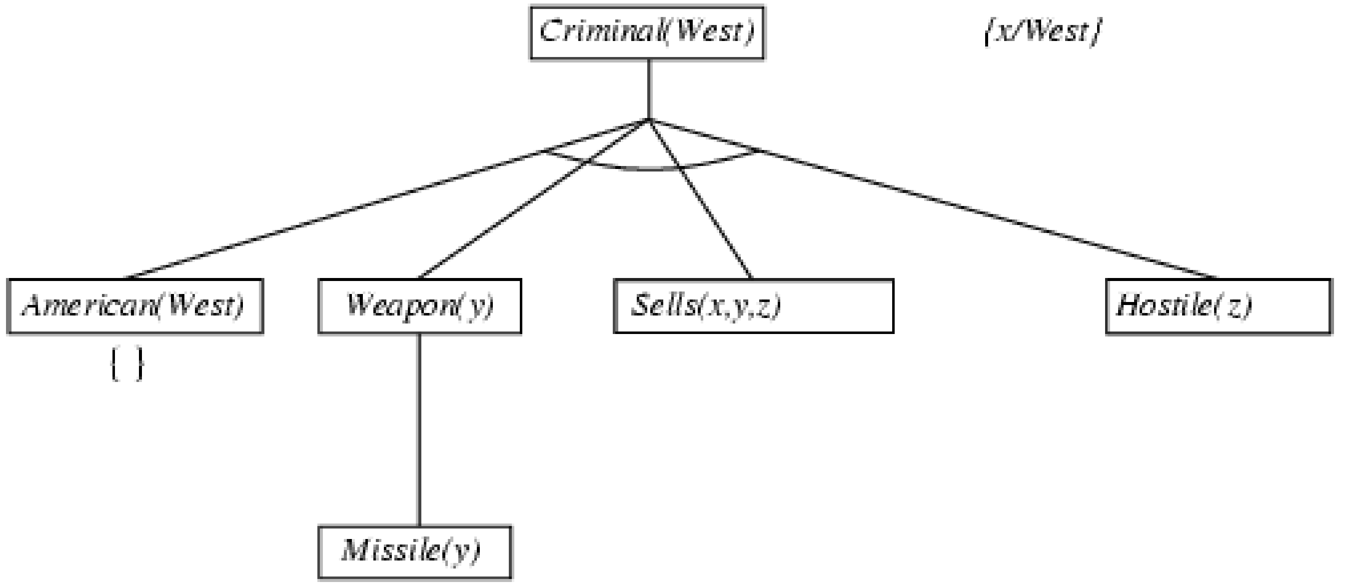
Backward chaining example



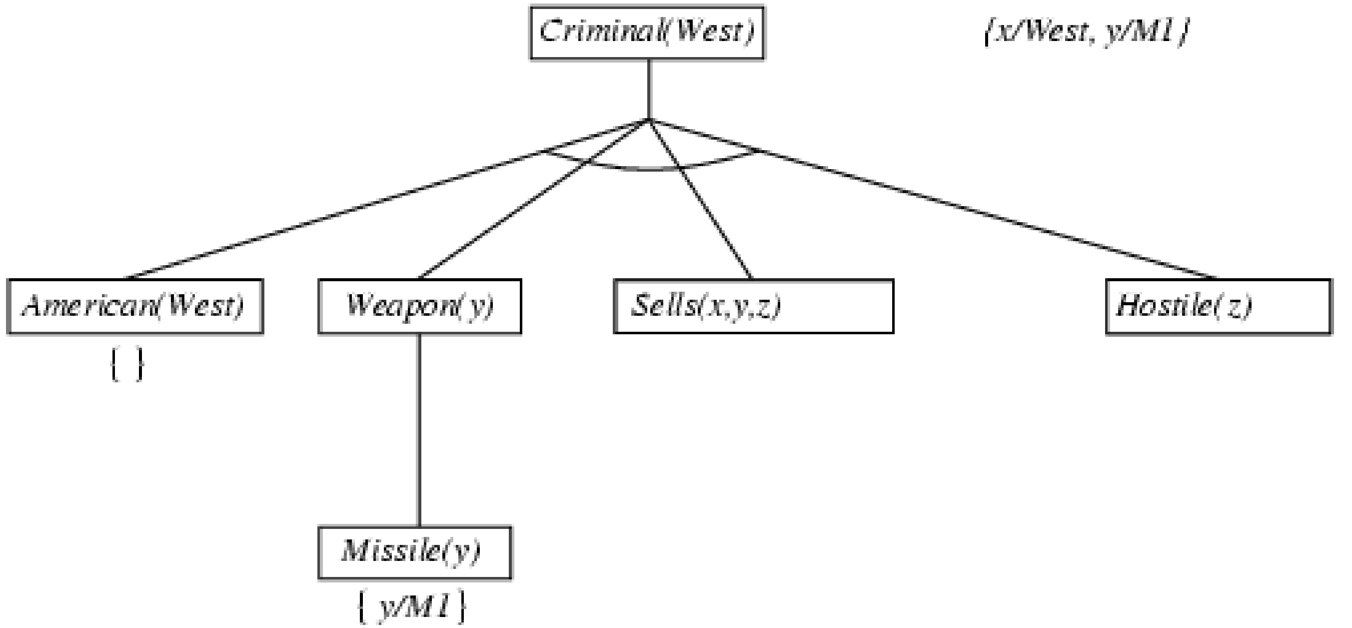
Backward chaining example



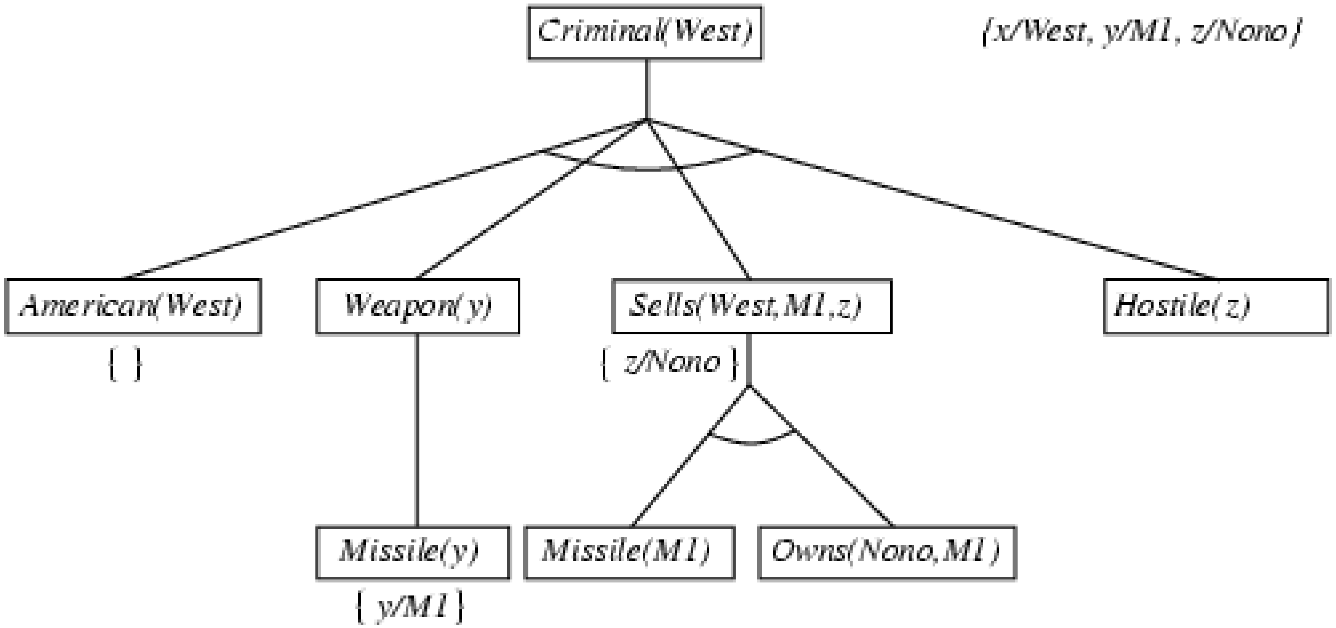
Backward chaining example



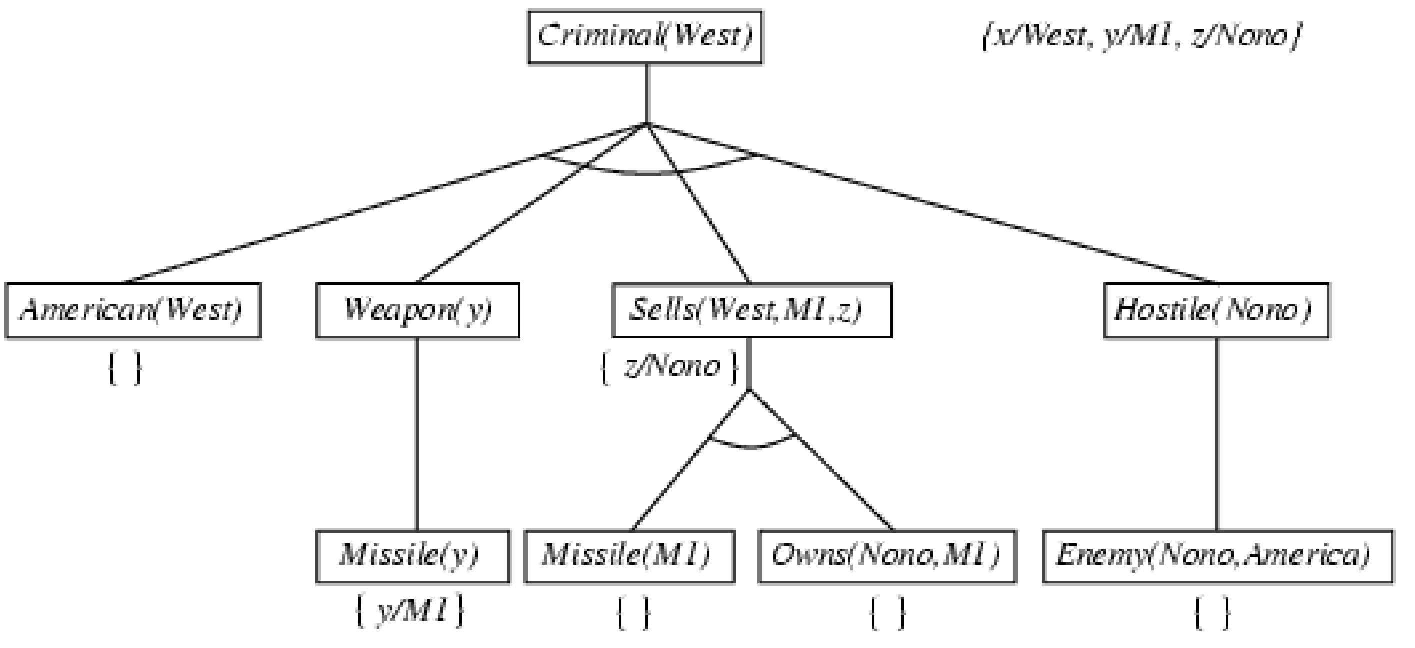
Backward chaining example



Backward chaining example



Backward chaining example



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

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

Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
  - But not in size of data (bindings)
- 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 (Prolog)

Prolog

- Appending two lists to produce a third:

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

- query: `append(A,B,[1,2]) ?`

- answers:

```
A=[]     B=[1,2]
A=[1]    B=[2]
A=[1,2]  B=[]
```

Logic programming: Prolog

- Algorithm = Logic + Control

- Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques ⇒ 60 million LIPS

- Program = set of clauses = `head :- literal`$_1$`, … literal`$_n$`.`

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- Depth-first, left-to-right (within rule), top-down (within rule-set) backward chaining
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)
- No occurs-check in unification – may produce results not entailed
- No checks for infinite loops – incomplete even for definite clauses
- Prolog : no caching; Tabled Logic Programming : memoization
- Database semantics :
  - Unique names assumption
  - Closed-world assumption ("negation as failure")
    - e.g., given `alive(X) :- not dead(X).`
    - `alive(joe)` succeeds if `dead(joe)` fails
  - Closed domain assumption

Resolution: brief summary

- Full first-order version:

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

where `Unify`$(\ell_i, \neg m_j) = \theta$.

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

- Apply resolution steps to CNF(KB $\wedge \neg\alpha$); complete (with factoring) 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

  $A \Leftrightarrow B$ becomes $(A \Rightarrow B) \wedge (B \Rightarrow A)$
  $A \Rightarrow B$ becomes $\neg A \vee B$

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

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

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

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: 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 $\lor$ over $\land$:

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

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., $\exists x\ Owns(Nono,x) \wedge Missile(x)$:

$Owns(Nono,M_1) \wedge Missile(M_1)$

… 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“:

$Enemy(x,America) \Rightarrow Hostile(x)$

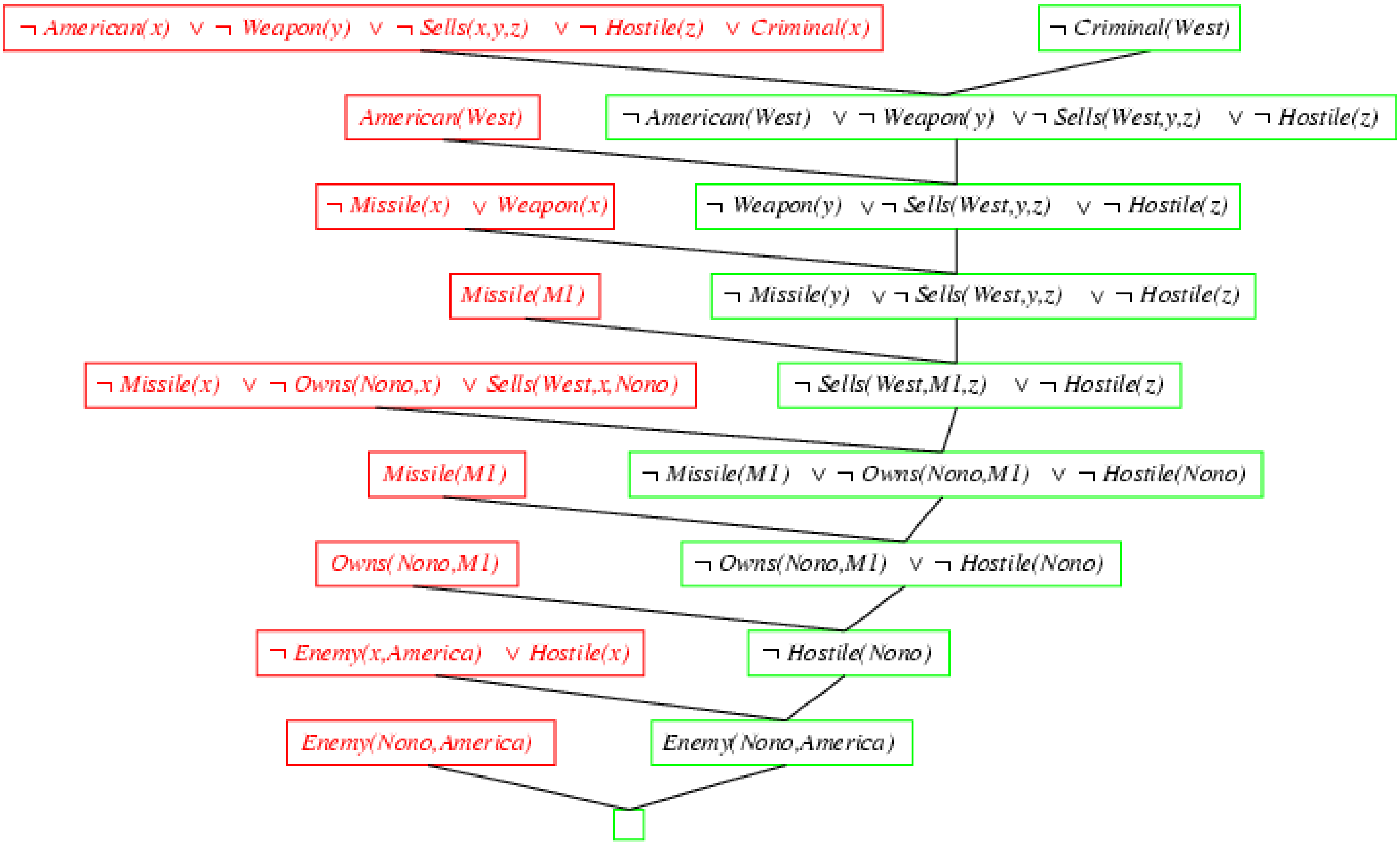
West, who is American …

$American(West)$

The country Nono, an enemy of America …

$Enemy(Nono,America)$

Resolution proof: definite clauses



Efficient Resolution

- Resolution proofs can be long
- Strategies :
  - Unit Preference
  - Set of support
  - Input resolution
    - Complete for Horn clauses
  - Linear Resolution
    - Complete in general

Converting to clause form (Try this example)

$$\forall x, y\, P(x) \wedge P(y) \wedge I(x,27) \wedge I(y,28) \rightarrow S(x,y)$$

$$P(A), P(B)$$

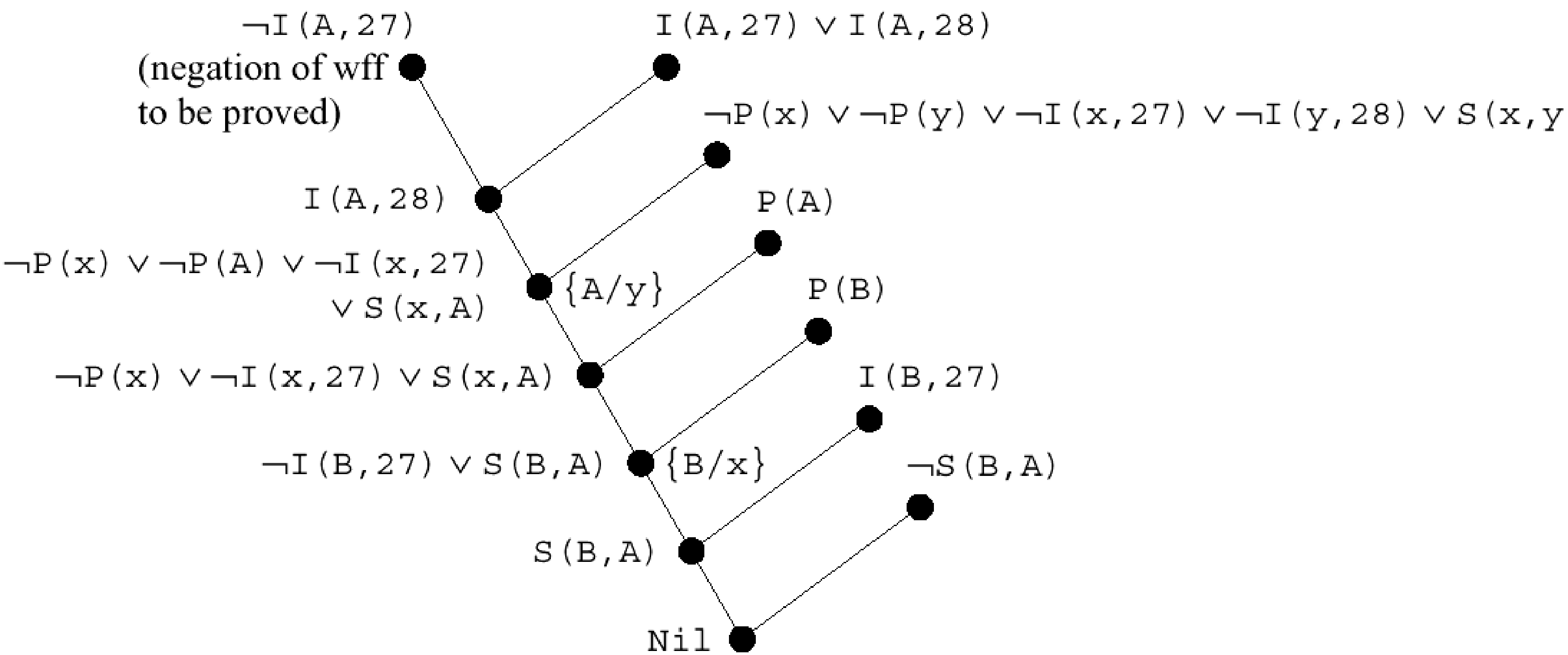
$$I(A,27) \vee I(A,28)$$

$$I(B,27)$$

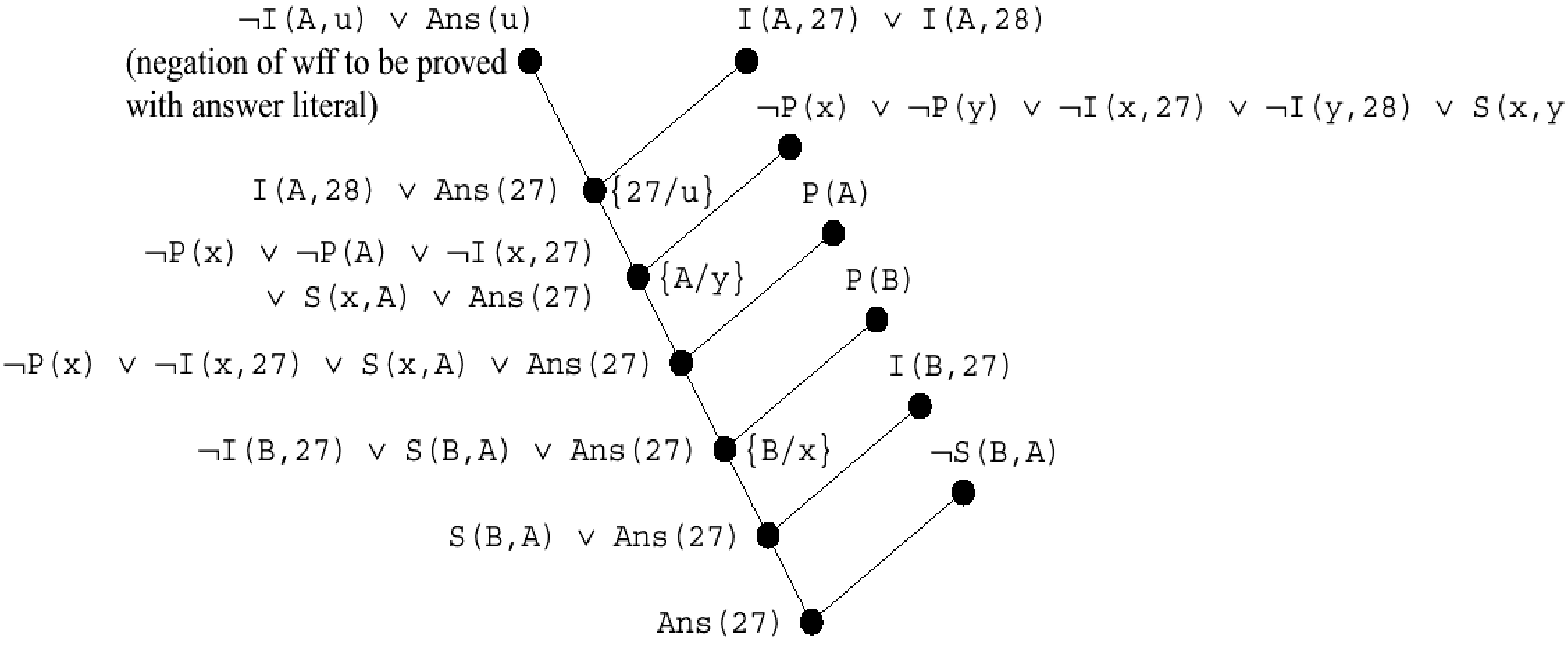
$$\neg S(B,A)$$

Prove I(A,27)

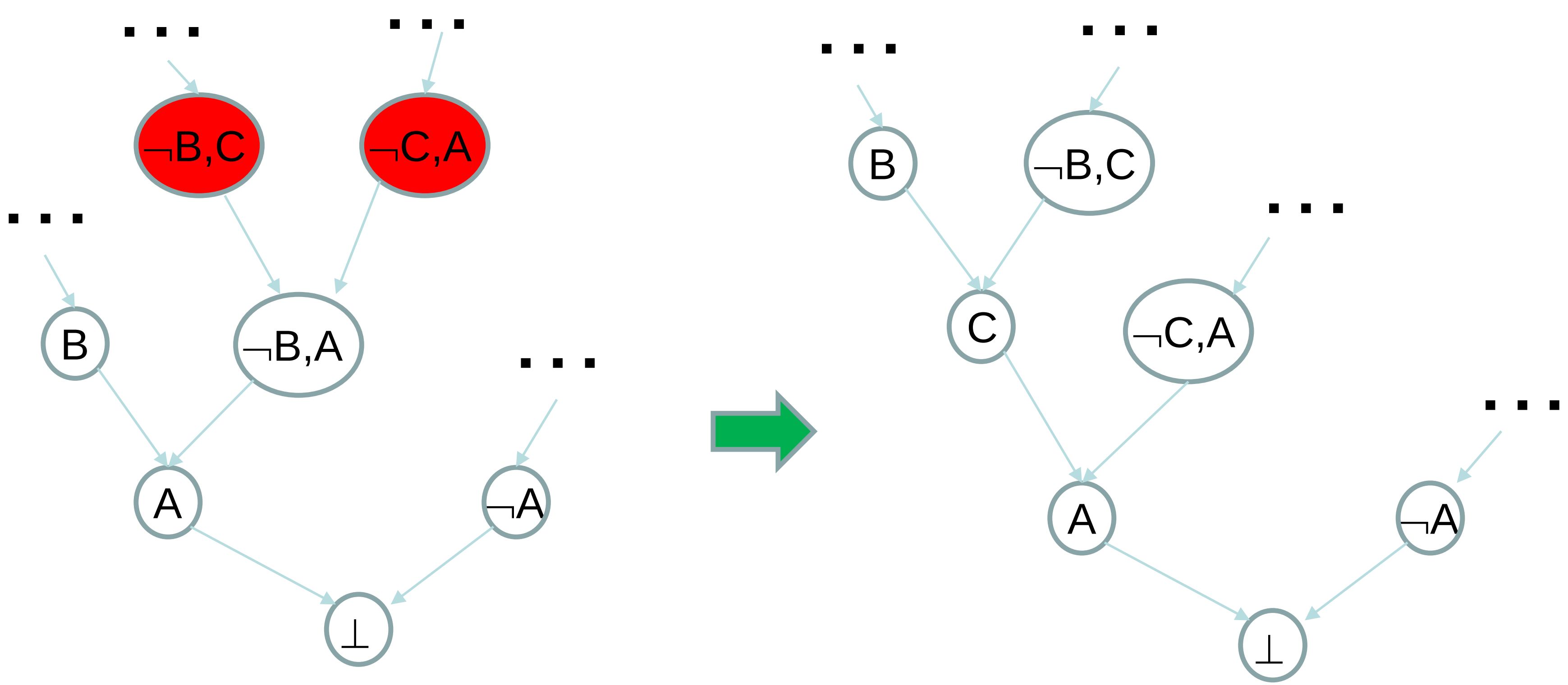
Example: Resolution Refutation Prove *I(A,27)*



Example: Answer Extraction



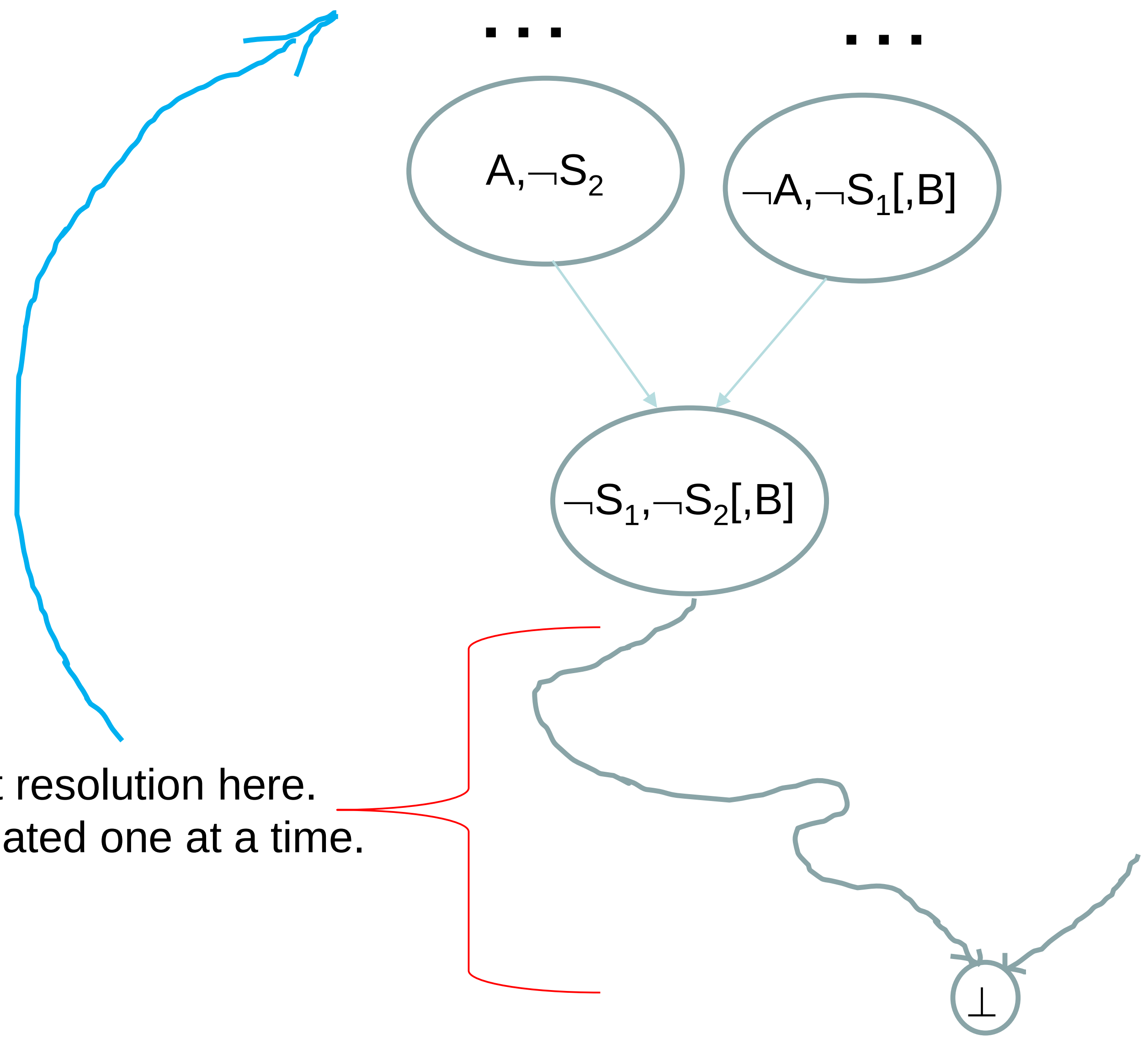
Unit resolution is (refutation) complete for Horn clauses : example



(Non-unit) resolution is complete!

Can transform any non-unit resolution proof to unit resolution proof

Unit resolution is (refutation) complete for Horn clauses : general case



Only unit resolution here.
S_2 eliminated one at a time.