



## Lecture 16 of 42

### Logic Programming Discussion: Machine Problem 4

Friday, 28 September 2007

William H. Hsu  
Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/v9v3>  
Course web site: <http://www.kddresearch.org/Courses/Fall-2007/CIS730>  
Instructor home page: <http://www.cis.ksu.edu/~bhsu>

**Reading for Next Class:**  
Review Chapter 9, Russell & Norvig 2<sup>nd</sup> edition



## Lecture Outline

- **Reading for Next Class: Chapter 9 review, R&N 2e**
- **Today**
  - \* Logic programming in real life
  - \* Industrial-strength Prolog
  - \* Lead-in to MP4
- **Next Week: Knowledge Representation and Ontologies**





## Logic Programming (Prolog) Examples

Depth-first search from a start state X:

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

No need to loop over S: successor succeeds for each

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=[]
```

Adapted from slides by S. Russell, UC Berkeley



## Completeness of Resolution

Any Set of Sentences S Is Representable in Clausal Form (Last Class)

Assume S Is Unsatisfiable, and in Clausal Form

(By Herbrand's Theorem) Some Set S' of Ground Instances is Unsatisfiable

(By Ground Resolution Theorem) Resolution Derives  $\perp$  From S'

(By Lifting Lemma)  $\exists$  A Resolution Proof S  $\vdash_n \perp$

Figure 9.13 p. 301 R&N 2e





## Conjunctive Normal (aka Clausal) Form: Conversion (Nilsson) and Mnemonic

- Implications Out
- Negations Out
- Standardize Variables Apart
- Existentials Out (Skolemize)
- Universals Made Implicit
- Distribute *And* Over *Or* (i.e., Disjunctions In)
- Operators Out
- Rename Variables
- A Memonic for *Star Trek: The Next Generation* Fans

**Captain Picard:**

**I'll Notify Spock's Eminent Underground Dissidents On Romulus**

**I'll Notify Sarek's Eminent Underground Descendant On Romulus**

Adapted from slides by S. Russell, UC Berkeley



## Logic Programming as Horn Clause Resolution

Basis: backward chaining with Horn clauses + bells & whistles  
Widely used in Europe, Japan (basis of 5th Generation project)  
Compilation techniques  $\Rightarrow$  10 million LIPS

Program = set of clauses = head :- literal<sub>1</sub>, ... literal<sub>n</sub>.

Efficient unification by open coding

Efficient retrieval of matching clauses by direct linking

Depth-first, left-to-right backward chaining

Built-in predicates for arithmetic etc., e.g., X is Y\*Z+3

Closed-world assumption ("negation as failure")

e.g., not PhD(X) succeeds if PhD(X) fails

Adapted from slides by S. Russell, UC Berkeley





## Logic Programming – Tricks of The Trade [1]: Dealing with Equality

### Problem

- \* How to find appropriate inference rules for sentences with =?
- \* Unification OK without it, but...
- \*  $A = B$  doesn't force  $P(A)$  and  $P(B)$  to unify

### Solutions

- \* Demodulation
  - ⇒ Generate substitution from equality term
  - ⇒ Additional sequent rule: p. 284 R&N
- \* Paramodulation
  - ⇒ More powerful
  - ⇒ Generate substitution from WFF containing equality constraint
  - ⇒ e.g.,  $(x = y) \vee P(x)$
  - ⇒ Sequent rule sketch: p. 284 R&N



## Logic Programming – Tricks of The Trade [2]: Resolution Strategies

### Unit Preference

- \* **Idea:** Prefer inferences that produce shorter sentences (compare: Occam's Razor)
- \* **How?** Prefer unit clause (*single-literal*) resolvents
- \* **Reason:** trying to produce a short sentence ( $\perp \equiv \text{True} \Rightarrow \text{False}$ )

### Set of Support

- \* **Idea:** try to eliminate some potential resolutions (prevention as opposed to cure)
- \* **How?** Maintain set SoS of resolution results and always take *one resolvent* from it
- \* **Caveat:** need right choice for SoS to ensure completeness

### Input Resolution and Linear Resolution

- \* **Idea:** "diagonal" proof (proof "list" instead of proof tree)
- \* **How?** Every resolution combines some input sentence with some other sentence
- \* **Input sentence:** *in original KB or query*
- \* **Generalize to linear resolution:** include any *ancestor in proof tree* to be used

### Subsumption

- \* **Idea:** eliminate sentences that are more specific than others
- \* E.g.,  $P(x)$  subsumes  $P(A)$





## Universal instantiation (UI)

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

$$\forall v \alpha$$

$$\text{Subst}(\{v/g\}, \alpha)$$

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

- E.g.,  $\forall x \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$  yields:

$$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$

$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

$$\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$$

⋮

⋮

⋮



## Existential instantiation (EI)

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

$$\exists v \alpha$$

$$\text{Subst}(\{v/k\}, \alpha)$$

- E.g.,  $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$  yields:

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

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:

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.,  $\text{Father}(\text{Father}(\text{Father}(\text{John})))$





## Reduction contd.

Theorem: Herbrand (1930). If a sentence  $\alpha$  is entailed by an FOL KB, it is entailed by a **finite** subset of the propositionalized KB

Idea: For  $n = 0$  to  $\infty$  do  
create a propositional KB by instantiating with depth- $n$  terms  
see if  $\alpha$  is entailed by this KB

Problem: works if  $\alpha$  is entailed, loops if  $\alpha$  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:  
 $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$   
 $\text{King}(\text{John})$   
 $\forall y \text{ Greedy}(y)$   
 $\text{Brother}(\text{Richard}, \text{John})$
- it seems obvious that  $\text{Evil}(\text{John})$ , but propositionalization produces lots of facts such as  $\text{Greedy}(\text{Richard})$  that are irrelevant
- With  $p$   $k$ -ary predicates and  $n$  constants, there are  $p \cdot n^k$  instantiations.





## The unification algorithm

```
function UNIFY( $x, y, \theta$ ) returns a substitution to make  $x$  and  $y$  identical
  inputs:  $x$ , a variable, constant, list, or compound
          $y$ , a variable, constant, list, or compound
          $\theta$ , the substitution built up so far

  if  $\theta = \text{failure}$  then return failure
  else if  $x = y$  then return  $\theta$ 
  else if VARIABLE?( $x$ ) then return UNIFY-VAR( $x, y, \theta$ )
  else if VARIABLE?( $y$ ) then return UNIFY-VAR( $y, x, \theta$ )
  else if COMPOUND?( $x$ ) and COMPOUND?( $y$ ) then
    return UNIFY(ARGS[ $x$ ], ARGS[ $y$ ], UNIFY(OP[ $x$ ], OP[ $y$ ],  $\theta$ ))
  else if LIST?( $x$ ) and LIST?( $y$ ) then
    return UNIFY(REST[ $x$ ], REST[ $y$ ], UNIFY(FIRST[ $x$ ], FIRST[ $y$ ],  $\theta$ ))
  else return failure
```



## The unification algorithm

```
function UNIFY-VAR( $var, x, \theta$ ) returns a substitution
  inputs:  $var$ , a variable
          $x$ , any expression
          $\theta$ , the substitution built up so far

  if  $\{var/val\} \in \theta$  then return UNIFY( $val, x, \theta$ )
  else if  $\{x/val\} \in \theta$  then return UNIFY( $var, val, \theta$ )
  else if OCCUR-CHECK?( $var, x$ ) then return failure
  else return add  $\{var/x\}$  to  $\theta$ 
```



## 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, 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( $KB, \alpha$ ) returns a substitution or false
  repeat until new is empty
     $new \leftarrow \{ \}$ 
    for each sentence  $r$  in  $KB$  do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)$ 
      for each  $\theta$  such that  $(p_1 \wedge \dots \wedge p_n)\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$ 
        for some  $p'_1, \dots, p'_n$  in  $KB$ 
           $q' \leftarrow \text{SUBST}(\theta, q)$ 
          if  $q'$  is not a renaming of a sentence already in  $KB$  or new then do
            add  $q'$  to new
             $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
            if  $\phi$  is not fail then return  $\phi$ 
    add new to  $KB$ 
  return false

```



## Forward chaining proof

$\text{American}(\text{West})$

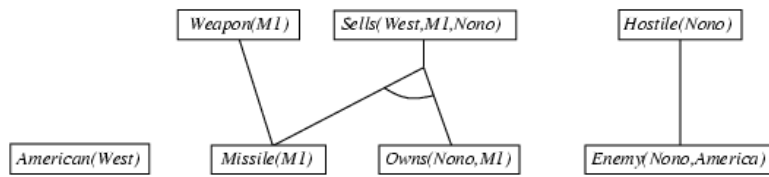
$\text{Missile}(\text{M1})$

$\text{Owns}(\text{Nono}, \text{M1})$

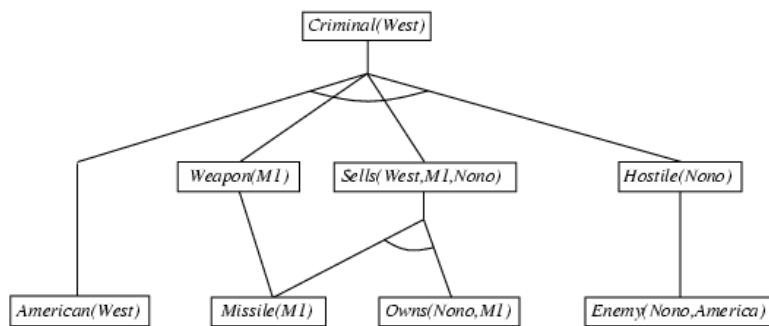
$\text{Enemy}(\text{Nono}, \text{America})$



## 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  $\alpha$  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

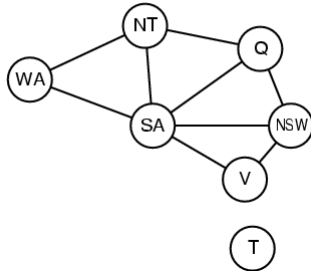
\* e.g., query *Missile(x)* retrieves *Missile(M<sub>i</sub>)*

Forward chaining is widely used in **deductive databases**





## Hard matching example



$$\text{Diff}(wa,nt) \wedge \text{Diff}(wa,sa) \wedge \text{Diff}(nt,q) \wedge \\ \text{Diff}(nt,sa) \wedge \text{Diff}(q,nsw) \wedge \text{Diff}(q,sa) \wedge \\ \text{Diff}(nsw,v) \wedge \text{Diff}(nsw,sa) \wedge \text{Diff}(v,sa) \Rightarrow \\ \text{Colorable}()$$

$$\begin{array}{ll} \text{Diff}(\text{Red},\text{Blue}) & \text{Diff}(\text{Red},\text{Green}) \\ \text{Diff}(\text{Green},\text{Red}) & \text{Diff}(\text{Green},\text{Blue}) \\ \text{Diff}(\text{Blue},\text{Red}) & \text{Diff}(\text{Blue},\text{Green}) \end{array}$$

- $\text{Colorable}()$  is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard



## Backward chaining algorithm

S

```
function FOL-BC-ASK(KB, goals,  $\theta$ ) returns a set of substitutions
  inputs: KB, a knowledge base
         goals, a list of conjuncts forming a query
          $\theta$ , the current substitution, initially the empty substitution {}
  local variables: ans, a set of substitutions, initially empty

  if goals is empty then return { $\theta$ }
   $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\text{goals}))$ 
  for each r in KB where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
     $\text{ans} \leftarrow \text{FOL-BC-ASK}(\text{KB}, [p_1, \dots, p_n] \text{REST}(\text{goals}), \text{COMPOSE}(\theta, \theta')) \cup \text{ans}$ 
  return ans
```



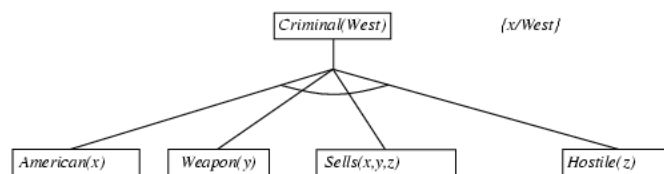


## Backward chaining example

*Criminal(West)*

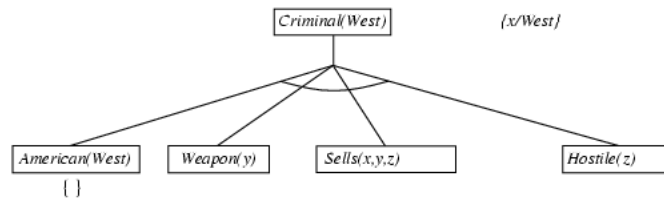


## Backward chaining example

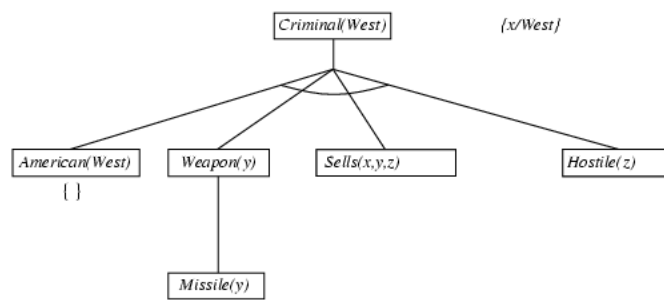




## Backward chaining example

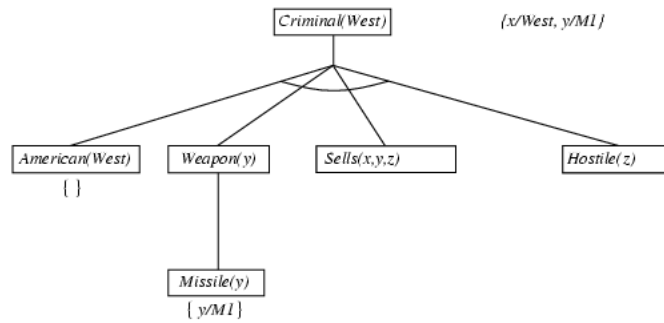


## Backward chaining example

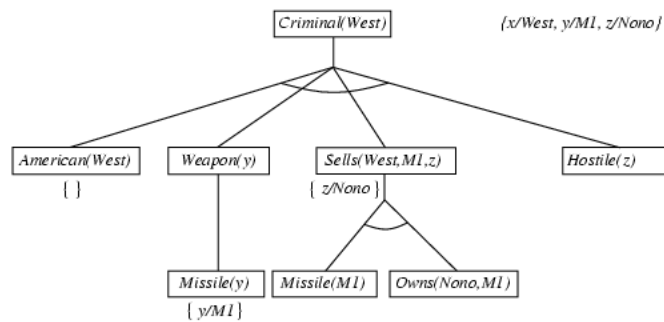




## Backward chaining example

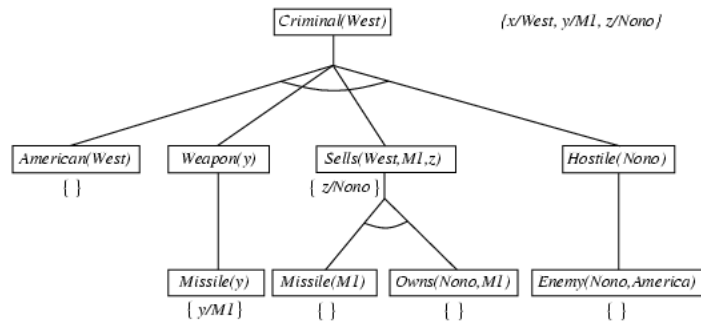


## Backward chaining example

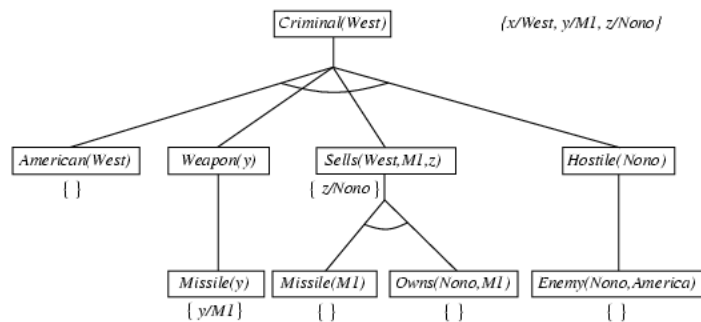




## Backward chaining example



## Backward chaining example





## 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



## 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=[]
```





## Resolution: brief summary

- Full first-order version:

$$(l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n \\ (l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta \\ \text{where } \text{Unify}(l_i, \neg m_j) = \theta.$$

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,

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

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

- Apply resolution steps to  $\text{CNF}(\text{KB} \wedge \neg \alpha)$ ; complete for FOL



## Conversion to CNF

- Everyone who loves all animals is loved by someone:

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

1. Eliminate biconditionals and implications

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

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

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

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

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



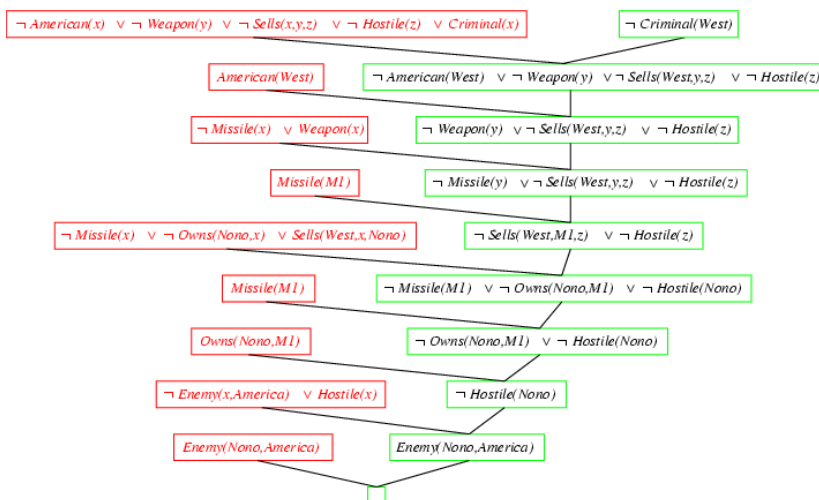


## Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one  
 $\forall x [\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists z \text{Loves}(z,x)]$
4. Skolemize: a more general form of existential instantiation.  
 Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:  
 $\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$
5. Drop universal quantifiers:  
 $[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$
6. Distribute  $\vee$  over  $\wedge$  :  
 $[\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)] \wedge [\neg \text{Loves}(x,F(x)) \vee \text{Loves}(G(x),x)]$



## Resolution proof: definite clauses





## Summary Points

- Previously: FOL, Forward/Backward Chaining, Resolution, Unification
- Today: Prolog and Decidability
  - \* Review: resolution inference rule
    - ⇒ Single-resolvent form
    - ⇒ General form
  - \* Application to logic programming
  - \* Review: decidability properties
    - ⇒ FOL-SAT
    - ⇒ FOL-NOT-SAT (language of unsatisfiable sentences; complement of FOL-SAT)
    - ⇒ FOL-VALID
    - ⇒ FOL-NOT-VALID
  - \* Unification
- Next Week
  - \* Intro to knowledge representation
  - \* Ontologies



## Terminology

- Properties of Knowledge Bases (KBs)
  - \* Satisfiability and validity
  - \* Entailment and provability
- Properties of Proof Systems
  - \* Soundness and completeness
  - \* Decidability, semi-decidability, undecidability
- Resolution
- Refutation
- Satisfiability, Validity
- Prolog: Tricks of The Trade
  - \* Demodulation, paramodulation
  - \* Unit resolution, set of support, input / linear resolution, subsumption
  - \* Indexing (table-based, tree-based)

