

## Inference in first-order logic: just a taste

Dr. Melanie Martin

CS 4480

Based on slides from

<http://aima.eecs.berkeley.edu/2nd-ed/slides-ppt/>

## Outline

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

## Inference with Quantifiers

- Universal Instantiation:
  - Given  $\forall X \text{ person}(X) \Rightarrow \text{likes}(X, \text{sun})$
  - Infer  $\text{person}(\text{john}) \Rightarrow \text{likes}(\text{john}, \text{sun})$
- Existential Instantiation:
  - Given  $\exists x \text{ likes}(x, \text{sun})$
  - Infer:  $\text{likes}(S1, \text{sun})$
  - S1 is a "Skolem Constant" that is not found anywhere else in the KB and refers to (one of) the individuals that likes sun.

## 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  $g$   
 ground term: term without variables  
 $\text{Subst}(\Theta, \alpha)$  apply substitution  $\Theta$  to sentence  $\alpha$   
 where  $\{v/g\}$  means substitute ground term  $g$  for variable  $v$

- 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:

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

- 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:

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$   
 $\text{King}(\text{John})$   
 $\text{Greedy}(\text{John})$   
 $\text{Brother}(\text{Richard}, \text{John})$

- Instantiating the universal sentence in **all possible** ways, we have:

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

- The new KB is **propositionalized**: proposition symbols are  $\text{King}(\text{John})$ ,  $\text{Greedy}(\text{John})$ ,  $\text{Evil}(\text{John})$ ,  $\text{King}(\text{Richard})$ , 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  $\alpha$  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

Depth 1: john, richard  
 Depth 2: father(john), father(richard)

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.
- However, we can get the inference immediately if we can find a substitution  $\theta$  such that  $\text{King}(x)$  and  $\text{Greedy}(x)$  match  $\text{King}(\text{John})$  and  $\text{Greedy}(y)$

## Unification

Want to use “lifted” inference rules

- Lifted from PL to POL
- Lifted inference rules require finding substitutions that make different logical expressions look identical: Unification
  - Key to FOL inference algorithms
  - Takes two sentences
  - Returns a unifier (if one exists)

## Unification

- AskVars(Knows(John, x)): who does John know?

$\theta = \{x/\text{John}, y/\text{John}\}$  works

- Unify( $\alpha, \beta$ ) =  $\theta$  if  $\alpha\theta = \beta\theta$

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,AJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,AJ)	

## Unification

- We can get the inference immediately if we can find a substitution  $\theta$  such that  $\text{King}(x)$  and  $\text{Greedy}(x)$  match  $\text{King}(\text{John})$  and  $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$  works

- Unify( $\alpha, \beta$ ) =  $\theta$  if  $\alpha\theta = \beta\theta$

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	$\{x/\text{Jane}\}$
Knows(John,x)	Knows(y,AJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,AJ)	

## Unification

- We can get the inference immediately if we can find a substitution  $\theta$  such that  $King(x)$  and  $Greedy(x)$  match  $King(John)$  and  $Greedy(y)$

$\theta = \{x/John, y/John\}$  works

- Unify( $\alpha, \beta$ ) =  $\theta$  if  $\alpha\theta = \beta\theta$

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	$\{x/Jane\}$
Knows(John,x)	Knows(y,AJ)	$\{x/AJ, y/John\}$
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,AJ)	

## Unification

- We can get the inference immediately if we can find a substitution  $\theta$  such that  $King(x)$  and  $Greedy(x)$  match  $King(John)$  and  $Greedy(y)$

$\theta = \{x/John, y/John\}$  works

- Unify( $\alpha, \beta$ ) =  $\theta$  if  $\alpha\theta = \beta\theta$

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	$\{x/Jane\}$
Knows(John,x)	Knows(y,AJ)	$\{x/AJ, y/John\}$
Knows(John,x)	Knows(y,Mother(y))	$\{y/John, x/Mother(John)\}$
Knows(John,x)	Knows(x,AJ)	

## Unification

- We can get the inference immediately if we can find a substitution  $\theta$  such that  $King(x)$  and  $Greedy(x)$  match  $King(John)$  and  $Greedy(y)$

$\theta = \{x/John, y/John\}$  works

- Unify( $\alpha, \beta$ ) =  $\theta$  if  $\alpha\theta = \beta\theta$

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	$\{x/Jane\}$
Knows(John,x)	Knows(y,AJ)	$\{x/AJ, y/John\}$
Knows(John,x)	Knows(y,Mother(y))	$\{y/John, x/Mother(John)\}$
Knows(John,x)	Knows(x,AJ)	{fail}

- Standardizing apart eliminates overlap of variables, e.g.,  $Knows(z, y, AJ)$

## Unification

When there is more than one unifier:

- To unify  $Knows(John, x)$  and  $Knows(y, z)$ ,  
 $\theta = \{y/John, x/z\}$  or  $\theta = \{y/John, x/John, z/John\}$
- The first unifier is **more general** than the second.
  - Fewer restrictions
- There is a single **most general unifier** (MGU) that is unique up to renaming of variables.  
MGU =  $\{y/John, x/z\}$

## Generalized Modus Ponens

- This is a general inference rule for FOL that does not require instantiation
- GMP “lifts” MP from propositional to first-order logic
- Key advantage of lifted inference rules over propositionalization is that they make only substitutions which are required to allow particular inferences to proceed

$$\frac{p1', p2', \dots, pn', (p1 \wedge p2 \wedge \dots \wedge pn \Rightarrow q)}{q\theta}$$

where  $\pi_i' \theta = \pi_i \theta \forall i$

- $p1'$  is King(John)       $p1$  is King(x)
- $p2'$  is Greedy(y)       $p2$  is Greedy(x)
- $\theta$  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

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

$Owns(Nono,M_i) \wedge Missile(M_i)$   
 ... 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 ...  
 $Enemy(Nono,America)$

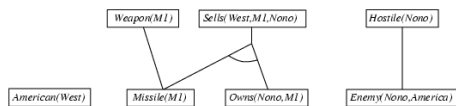
## Forward chaining algorithm

```
function FOL-FC-Ask(KB,α) returns a substitution or false
  repeat until new is empty
    new ← {}
    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
```

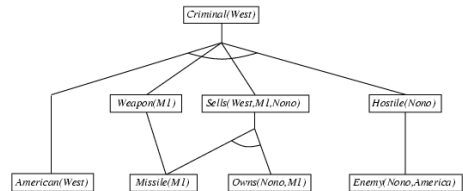
## Forward chaining proof

American(West)   Missile(M<sub>1</sub>)   Owns(Nono, M<sub>1</sub>)   Enemy(Nono, 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  $k-1$   
 $\Rightarrow$  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 algorithm

```

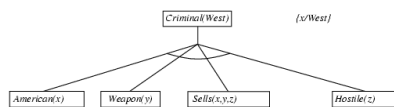
function FOL-BC-ASK( $KB, goals, \theta$ ) returns a set of substitutions
  inputs:  $KB$ , a knowledge base
          $goals$ , a list of conjuncts forming a query
          $\theta$ , the current substitution, initially the empty substitution { }
  local variables:  $ans$ , a set of substitutions, initially empty
  if  $goals$  is empty then return {  $\theta$  }
   $q' \leftarrow SUBST(\theta, FIRST(goals))$ 
  for each  $r$  in  $KB$  where  $STANDARDIZE-APART(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow UNIFY(q, q')$  succeeds
     $ans \leftarrow FOL-BC-ASK(KB, [p_1, \dots, p_n], REST(goals), COMPOSE(\theta, \theta')) \cup ans$ 
  return  $ans$ 
    
```

$SUBST(COMPOSE(\theta_1, \theta_2), p) =$   
 $SUBST(\theta_2, SUBST(\theta_1, p))$

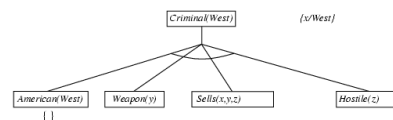
## Backward chaining example

$Criminal(West)$

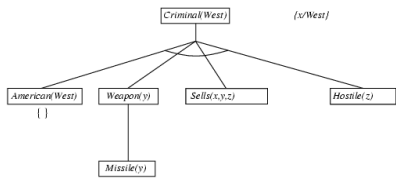
## Backward chaining example



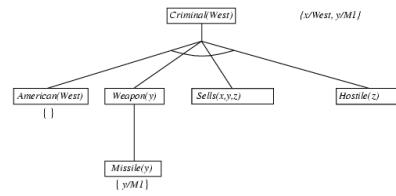
## Backward chaining example



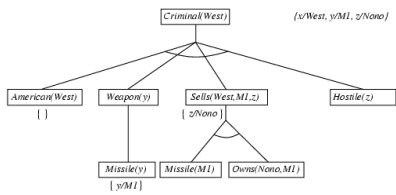
### Backward chaining example



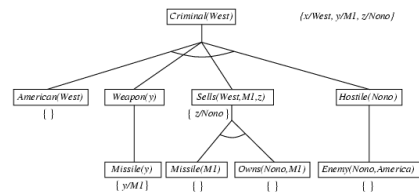
### Backward chaining example



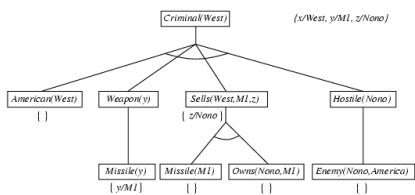
### Backward chaining example



### 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
  - => 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**

## 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 LPS
- Program = set of clauses = head :- literal<sub>1</sub>, ... literal<sub>n</sub>.  
`criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`
- Depth-first, left-to-right 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)
- Closed-world assumption ("negation as failure")
  - e.g. `given alive(X) :- not dead(X).`
  - `alive(joe) succeeds if dead(joe) fails`

## 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:
 
$$\frac{\ell_1 \vee \dots \vee \ell_r \quad m_1 \vee \dots \vee m_n}{(\ell_1 \vee \dots \vee \ell_{r-1} \vee \ell_{r+1} \vee \dots \vee \ell_r \vee m_1 \vee \dots \vee m_{n-1} \vee m_{n+1} \vee \dots \vee m_n) \theta}$$
 where  $\text{Unify}(\ell_r, m_i) = \theta$ .
- The two clauses are assumed to be standardized apart so that they share no variables.
- For example  $\neg \text{Rich}(x) \vee \text{Unhappy}(x), \text{Rich}(\text{Ken})$   

$$\frac{\neg \text{Rich}(x) \vee \text{Unhappy}(x), \text{Rich}(\text{Ken})}{\text{Unhappy}(\text{Ken})}$$
 with  $\theta = \{x/\text{Ken}\}$
- Apply resolution steps to CNF(KB ∧ ¬α); 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 ¬ 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.

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

## Resolution proof: definite clauses

