Inference in first-order logic: just a taste

Dr. Melanie Martin

CS 4480

November 2, 2012

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 ∀X person(X) ⇒ likes(X, sun)
  - Infer person(john) ⇒ likes(john,sun)
- Existential Instantiation:
  - Given ∃x likes(x, sun)
  - Infer: likes(S1, 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
Subst(Θ, α) apply substitution Θ to sentence α
where {v/g} means substitute ground term g for variable b

- E.g., ∀x *King*(*x*) ∧ *Greedy*(*x*) ⇒ *Evil*(*x*) yields:
  King(*John*) ∧ *Greedy*(*John*) ⇒ *Evil*(*John*)
  King(*Richard*) ∧ *Greedy*(*Richard*) ⇒ *Evil*(*Richard*)
  King(*Father*(*John*)) ∧ *Greedy*(*Father*(*John*)) ⇒ *Evil*(*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., ∃*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

Reduction to propositional inference

Suppose the KB contains just the following:
∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
King(John)
Greedy(John)
Brother(Richard,John)

- Instantiating the universal sentence in all possible ways, we have:
  King(John) ∧ Greedy(John) ⇒ Evil(John)
  King(Richard) ∧ Greedy(Richard) ⇒ 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
 Depth1: john, richard
 Depth 2: father(john), father(richard)

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:
  ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
  King(John)
  ∀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.
- However, 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)*

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 exits)

Unification

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

θ = {x/John,y/John} works

- Unify(α,β) = θ if αθ = βθ

p	q	θ
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 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,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 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,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 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,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 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,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_{17},AJ)

Unification

When there is more than one 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.
  – 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

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

where pi' θ=piθ ∀i

• p1' is King(John)	p1 is King(x)
• p2' is Greedy(y)	p2 is Greedy(x)
• Θ is {x/John,y/John}	q is Evil(x)

- qθ 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) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

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

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile“:

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

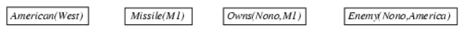
The country Nono, an enemy of America …

Enemy(Nono,America)

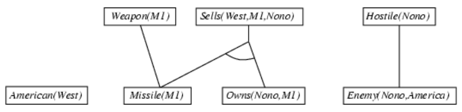
Forward chaining algorithm

```
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
            $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)
            for each $\theta$ 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($\theta, q$)
                if $q'$ is not a renaming of a sentence already in $KB$ or $new$ then do
                    add $q'$ to $new$
                    $\phi \leftarrow$ UNIFY($q', \alpha$)
                    if $\phi$ is not $fail$ then return $\phi$
        add $new$ to $KB$
    return $false$
```

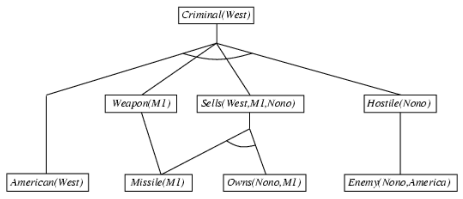
Forward chaining proof



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 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*

⇒ 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)*

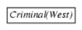
Forward chaining is widely used in deductive databases

Backward chaining algorithm

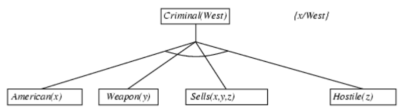
```
function FOL-BC-ASK($KB$, goals, $\theta$) returns a set of substitutions
   inputs: $KB$, a knowledge base
           $goals$, a list of conjuncts forming a query
           $\theta$, the current substitution, initially the empty substitution $\{\ \}$
   local variables: $ans$, a set of substitutions,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

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

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

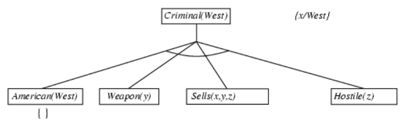
Backward chaining example



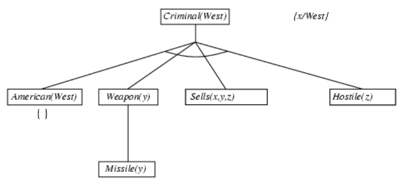
Backward chaining example



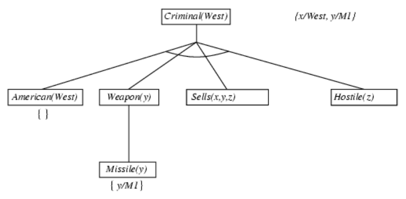
Backward chaining example



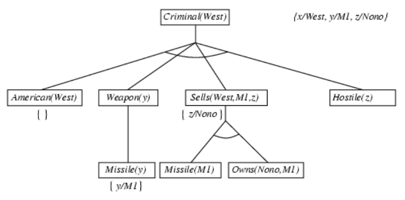
Backward chaining example



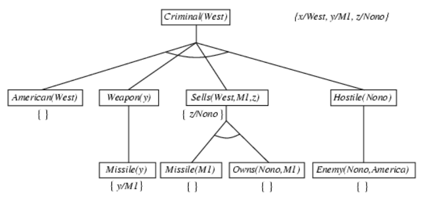
Backward chaining example



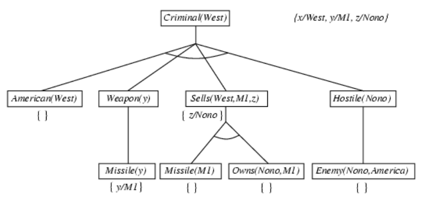
Backward chaining example



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 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 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:

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

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

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

with θ = {x/Ken}

- Apply resolution steps to CNF(KB ∧ ¬α); complete for FOL

Conversion to CNF

- Everyone who loves all animals is loved by someone:
  $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
- 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 ¬ inwards: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\exists x\ p \equiv \forall x\ \neg p$

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

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:

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

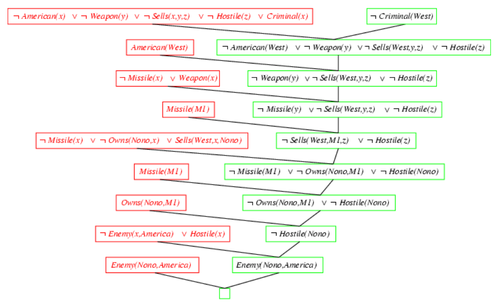
5. Drop universal quantifiers:

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

6. Distribute ∨ over ∧ :

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

Resolution proof: definite clauses


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