ANU COMP3620/6320 Inference in FOL

Chapter 9: "AI: A Modern Approach"

Lecturer: Scott Sanner

Slides Adapted from Instructor's Material

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- 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., ∀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) ∧ *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 (for finitely bounded domains) 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 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 knowledge base (KB):
  ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
  King(John)
  ∀y Greedy(y)
  Brother(Richard,John)

  Query: *Evil*(*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.

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)	
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)	{fail}

- 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} ∈ θ then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {var/x} to θ
```

Generalized Modus Ponens (GMP)

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

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', \; …, \;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, qθ follows by ordinary Modus Ponens

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 …

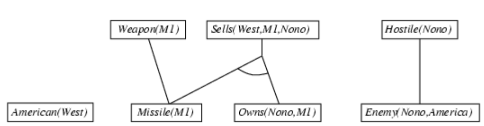
Enemy(Nono,America)

Forward chaining algorithm

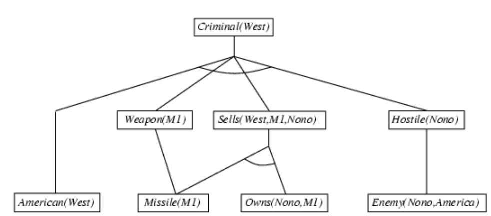
```
function FOL-FC-ASK(KB, α) returns a substitution or false
  repeat until new is empty
     new ← { }
     for each sentence r in KB do
        ( p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
        for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                    for some p'1, ..., p'n 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

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 $Missile(M_1)$

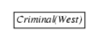
Forward chaining is widely used in deductive databases

Backward chaining algorithm

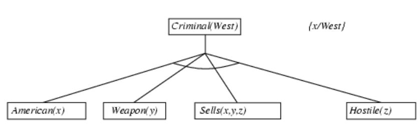
```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
  inputs: KB, a knowledge base
          goals, a list of conjuncts forming a query
          θ, the current substitution, initially the empty substitution { }
  local variables: ans, a set of substitutions, initially empty

  if goals is empty then return {θ}
  q' ← SUBST(θ, FIRST(goals))
  for each r in KB where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
          and θ' ← UNIFY(q, q') succeeds
     ans ← FOL-BC-ASK(KB, [p1, ..., pn|REST(goals)], COMPOSE(θ, θ')) ∪ ans
  return ans
```

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

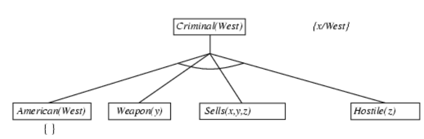
Backward chaining example



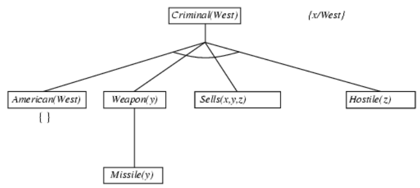
Backward chaining example



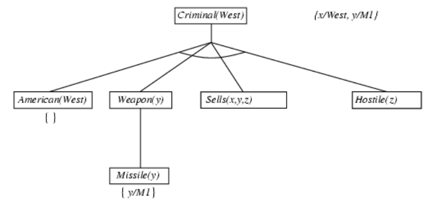
Backward chaining example



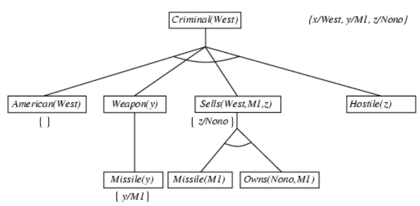
Backward chaining example



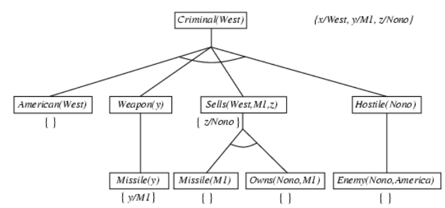
Backward chaining example



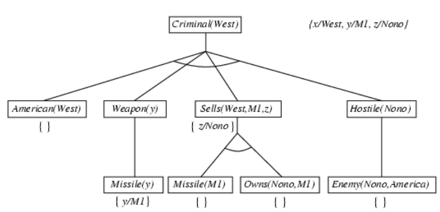
Backward chaining example



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
  - $\Rightarrow$ 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

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=[]
```

Generalization of Resolution from Propositional to First-order Logic

Skolemization

- We need to…
  - Remove all existental quantifiers while *preserving* satisfiability
  - Note: this does not yield a logically equivalent KB!
- Do this via Skolemization
  - A more general form of existential instantiation
  - Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables
- $\exists x\, \exists y\; Loves(x, y) \;\rightarrow\; Loves(c_1, c_2)$
- $\exists x\, \forall y\; Loves(x, y) \;\rightarrow\; Loves(c_1, y)$
- $\forall x\, \exists y\; Loves(x, y) \;\rightarrow\; Loves(x, f(x))$

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. Push ¬ in: $\neg\forall x\; p \equiv \exists x\; \neg p$, $\neg\exists x\; p \equiv \forall x\; \neg p$
  $\forall x\, [\exists y\; \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\; Loves(y,x)]$
  $\forall x\, [\exists y\; \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\; Loves(y,x)]$
  $\forall x\, [\exists y\; Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\; Loves(y,x)]$

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

Resolve multiple literals or use factoring rule for completeness

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

- Clauses are assumed to be standardized apart – share no variables

- For example,

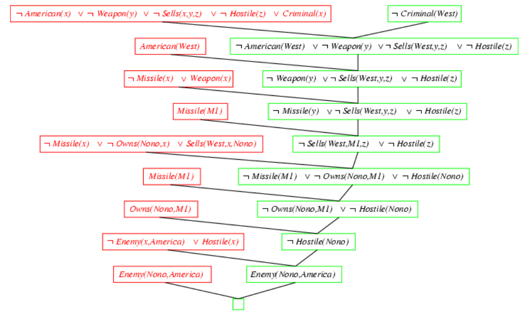
$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

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

Need paramodulation rule for completeness with equality

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

Resolution proof: definite clauses



Completeness

- Basic motivation from Herbrand's theorem
  - (Roughly) If a refutation of a set of clauses exists, ⇒ ∃ refutation with finite ground clauses
- Resolution with most general unifier (MGU)
  - Represents most general resolution
  - Covers resolutions over infinite number of (propositional) ground clauses
  - The ground refutation (if it exists) must be subsumed by this "lifted" resolution procedure

Godel's Incompleteness Theorem

- Wait…
  - Resolution rule + factoring + paramodulation is complete for FOL with equality
- But Godel said FOL is incomplete
  - With + and * functions
- Reason is that + and * must be axiomatized
  - But cannot bound the number of axioms needed to find a refutation (if it exists)
- So FOL with equality and +, * is incomplete