Inference in first-order logic

Chapter 9

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 person(X) $\Rightarrow$ likes(X, sun)
  - Infer person(john) $\Rightarrow$ likes(john,sun)
- Existential Instantiation:
  - Given $\exists$x likes(x, chocolate)
  - Infer: likes(S1, chocolate)
  - 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*

- E.g., $\forall$x *King*(*x*) $\wedge$ *Greedy*(*x*) $\Rightarrow$ *Evil*(*x*) yields:
  King(*John*) $\wedge$ *Greedy*(*John*) $\Rightarrow$ *Evil*(*John*)
  King(*Richard*) $\wedge$ *Greedy*(*Richard*) $\Rightarrow$ *Evil*(*Richard*)
  King(*Father*(*John*)) $\wedge$ *Greedy*(*Father*(*John*)) $\Rightarrow$ *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., $\exists x\ Crown(x) \wedge OnHead(x,John)$ 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:

$\forall x$ 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 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:
  $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
  King(John)
  $\forall y$ 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.

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)

More on Standardizing apart

- knows(john,X).
- knows(X,elizabeth).
- These ought to unify, since john knows everyone, and everyone knows elizabeth.
- Rename variables to avoid such name clashes

Note:

all X p(X) == all Y p(Y)

All X (p(X) ^ q(X)) == All X p(X) ^ All Y p(Y)

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 }

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

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)

Another Example

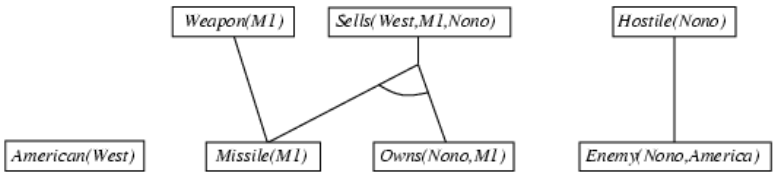
- ∀x,y,z parent(x,y) ∧ parent(y,z) ⇒ grandparent(x,z)
- parent(james, john), parent(james, richard), parent(harry, james)
- We can derive:
  - Grandparent(harry, john), bindings: {x/harry,y/james,z/john}
  - Grandparent(harry, richard), bindings: {x/harry,y/james,z/richard}

Forward chaining algorithm

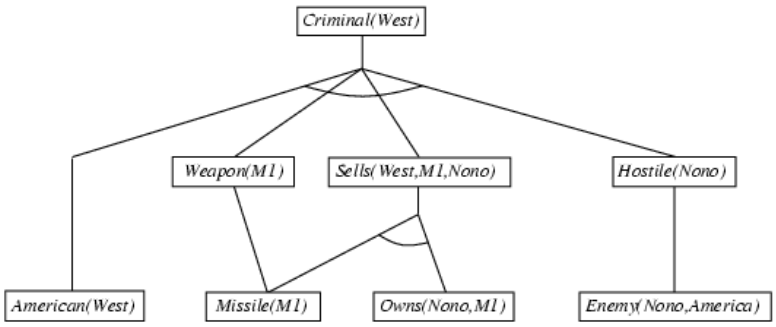
```
function FOL-FC-Ask(KB, α) returns a substitution or false
  repeat until new is empty
    new ← { }
    for each sentence r in KB do
      (p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Apart(r)
      for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                  for some p'1, ..., p'n in KB
        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
- Forward chaining is widely used in deductive databases

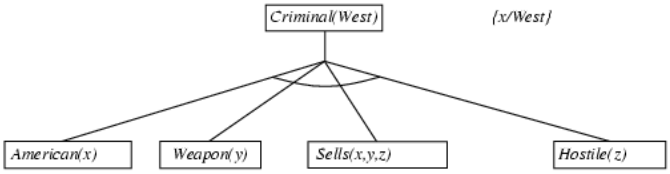
Backward chaining algorithm

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

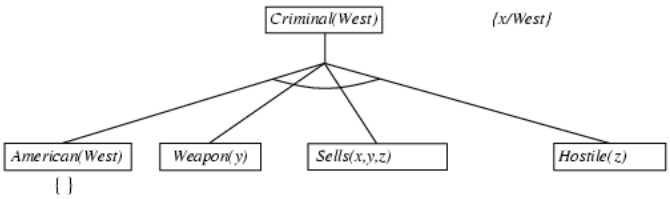
  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST($goals$))
  for each r **in** KB where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
    $ans \leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$,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

Backward chaining example

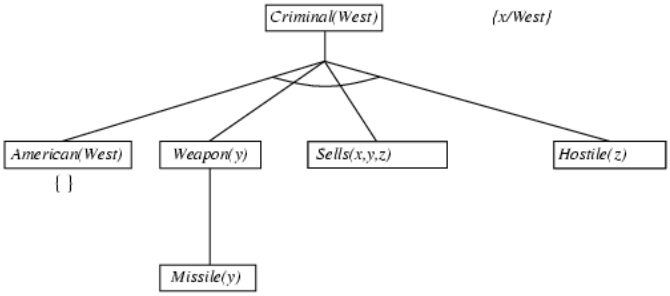
Backward chaining example



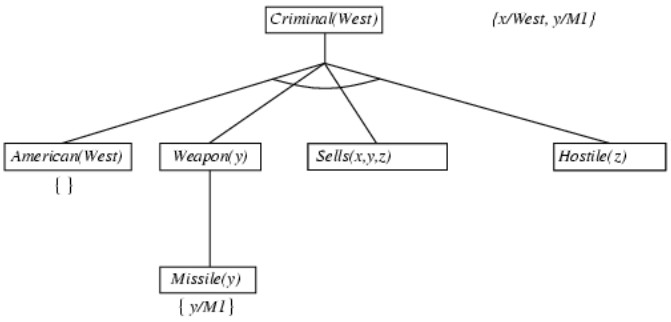
Backward chaining example



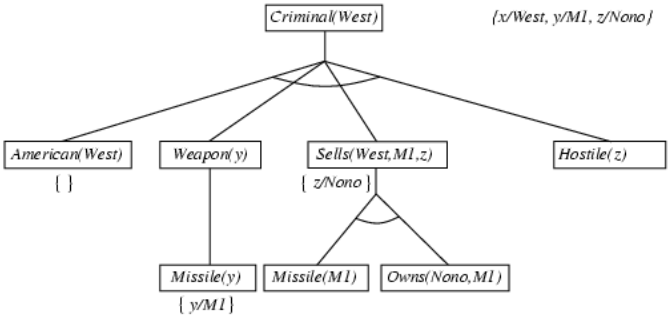
Backward chaining example



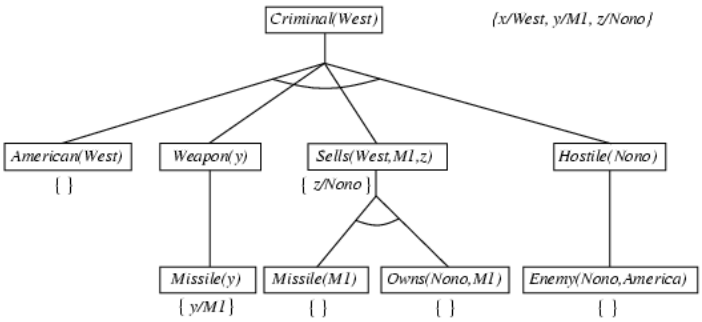
Backward chaining example



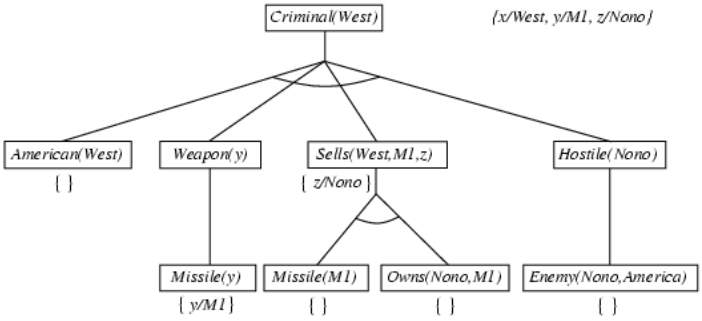
Backward chaining example



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 $\Rightarrow$ 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

neighbor(canada,us)
neighbor(mexico,us)
neighbor(pakistan,india)

?- neighbor(canada,india).
no

In full first-order logic, you would have to be able to infer “~neighbor(canada,india)" for "neighbor(canada,india)" to be false.

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 $\texttt{Unify}(\ell_i, \neg m_j) = \theta$.

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

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

  with θ = {x/Ken}

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

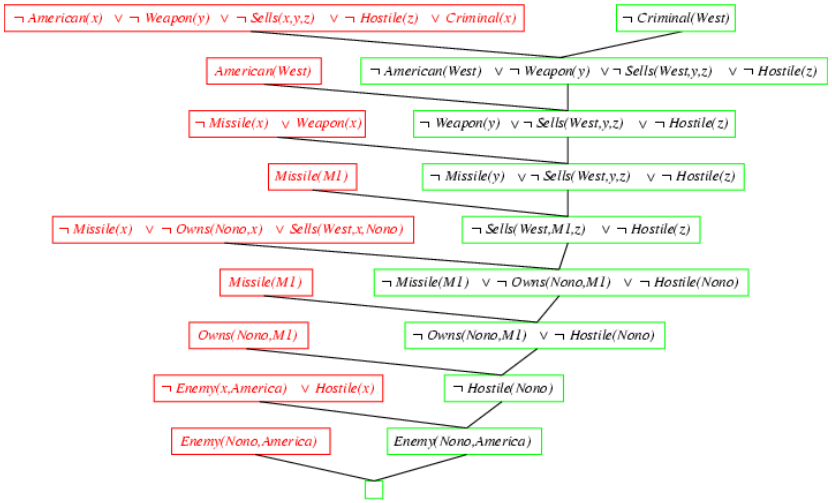
Conversion to CNF

- Everyone who loves all animals is loved by someone:
  $\forall x\ [\forall y\ \mathit{Animal}(y) \Rightarrow \mathit{Loves}(x,y)] \Rightarrow [\exists y\ \mathit{Loves}(y,x)]$
- 1. Eliminate biconditionals and implications
  $\forall x\ [\neg\forall y\ \neg\mathit{Animal}(y) \vee \mathit{Loves}(x,y)] \vee [\exists y\ \mathit{Loves}(y,x)]$
- 2. Move $\neg$ inwards: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\ \exists x\ p \equiv \forall x\ \neg p$
  $\forall x\ [\exists y\ \neg(\neg\mathit{Animal}(y) \vee \mathit{Loves}(x,y))] \vee [\exists y\ \mathit{Loves}(y,x)]$
  $\forall x\ [\exists y\ \neg\neg\mathit{Animal}(y) \wedge \neg\mathit{Loves}(x,y)] \vee [\exists y\ \mathit{Loves}(y,x)]$
  $\forall x\ [\exists y\ \mathit{Animal}(y) \wedge \neg\mathit{Loves}(x,y)] \vee [\exists y\ \mathit{Loves}(y,x)]$

Conversion to CNF contd.

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

Resolution proof: definite clauses



- Resolution is complete. If you don't want to take this on faith, study pp. 300-303
- Strategies (heuristics) for efficient resolution exist, see details in text if interested
  - Unit preference. If a clause has only one literal, use it first.
  - Set of support. Identify "useful" rules and ignore the rest. (p. 305)
  - Input resolution. Intermediately generated sentences can only be combined with original inputs or original rules.
  - Subsumption. Prune unnecessary facts from the database.

Inference Methods (Review)

- Unification (prerequisite)
- Forward Chaining
  - Production Systems
- Backward Chaining
  - Logic Programming (Prolog)
- Resolution
  - Transform to CNF
  - Generalization of Prop. Logic resolution

Pages to skip/skim

- Storage and Retrieval (pp. 278-279)
- Efficient forward chaining (starts p. 283) through Irrelevant facts (ends p. 287)
- Efficient implementation of logic programs (starts p. 290) through Constraint logic programming (ends p. 295)
- Completeness of resolution (pp. 300-303)
- Extending Prolog (pp. 307-308)