Based on AIMA PPT slides

Artificial intelligence 1:Inference in first-order logic

Lecturer: Tom Lenaerts

Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA)

Université Libre de Bruxelles

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:

$$Crown(C_1) \wedge OnHead(C_1, John)$$

  provided C_1 is a new constant symbol, called a Skolem constant

Existential instantiation cont'd

- UI can be applied several times to *add* new sentences; the new KB is logically equivalent to the old.
- EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old but is satifiable if the old KB was satisfiable.

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.

- *CLAIM*: A ground sentence is entailed by a new KB iff entailed by the original KB.
- *CLAIM*: Every FOL KB can be propositionalized so as to preserve entailment
- *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 α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {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 α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {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 α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {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 α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {x/John,y/John} works

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

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 α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {x/John,y/John} works

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

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)*,

  α = {y/John, x/z } or α = {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 $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **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)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where $p_i'\theta = p_i\theta$ 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)
$q\theta$ 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. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. From 1 and 2, $q\theta$ 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:

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

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

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)

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

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)

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:

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)

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“:

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)

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 …

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)

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 …

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)

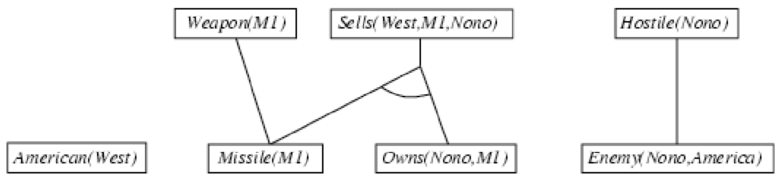
Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

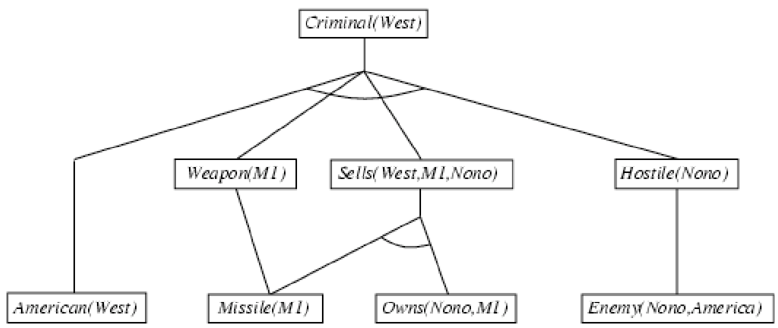
  repeat until *new* is empty
    $new \leftarrow \{\}$
    for each sentence r **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
      for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, 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', α)
          if ϕ is not *fail* **then return** ϕ
    add *new* to KB
  return *false*

Forward chaining proof

Forward chaining proof



Forward chaining proof



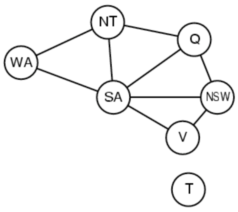
Properties of forward chaining

- Sound and complete for first-order definite clauses.
  - **Cfr. Propositional logic proof.**
- *Datalog* = first-order definite clauses + *no functions* (e.g. crime KB)
  - **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)***
- Matching conjunctive premises against known facts is NP-hard.
- Forward chaining is widely used in deductive databases

Hard matching example



Diff(wa,nt) ∧ *Diff(wa,sa)* ∧ *Diff(nt,q)* ∧
Diff(nt,sa) ∧ *Diff(q,nsw)* ∧ *Diff(q,sa)* ∧
Diff(nsw,v) ∧ *Diff(nsw,sa)* ∧ *Diff(v,sa)* ⇒
Colorable()

Diff(Red,Blue) *Diff (Red,Green)*
Diff(Green,Red) *Diff(Green,Blue)*
Diff(Blue,Red) *Diff(Blue,Green)*

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

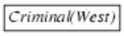
Backward chaining algorithm

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

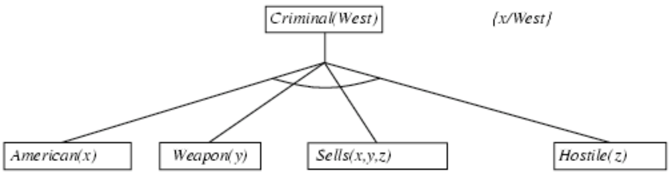
  if *goals* is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST(*goals*))
  for each *r* **in** *KB* where STANDARDIZE-APART(*r*) = $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
   ans ← FOL-BC-ASK(*KB*, $[p_1, \ldots, p_n|$REST(*goals*)],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

SUBST(COMPOSE(α_1, α_2), p) = SUBST(α_2, SUBST(α_1, p))

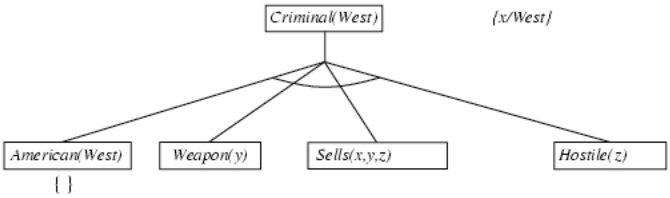
Backward chaining example



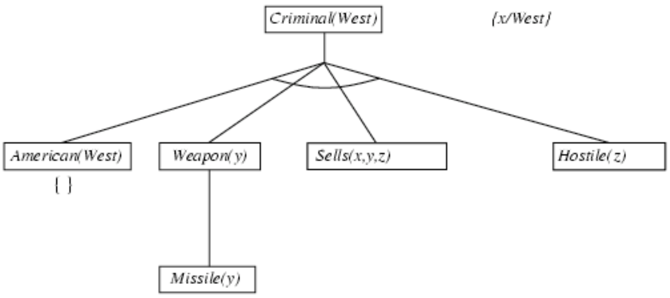
Backward chaining example



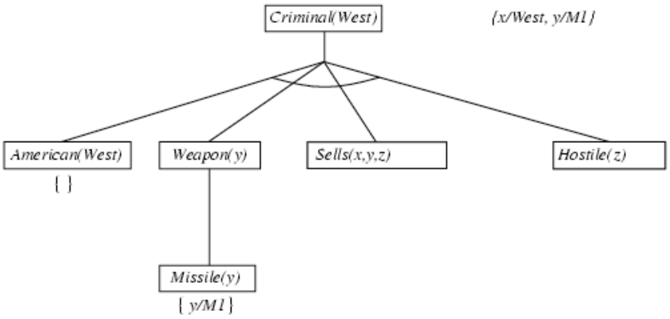
Backward chaining example



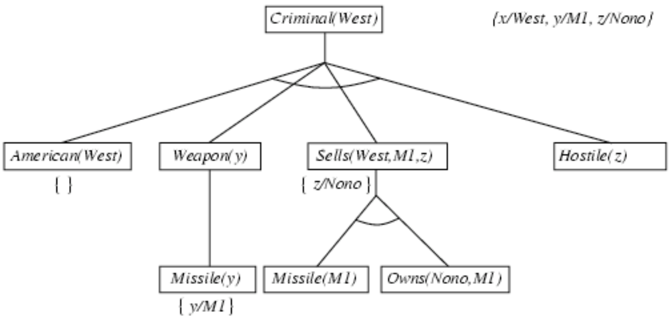
Backward chaining example



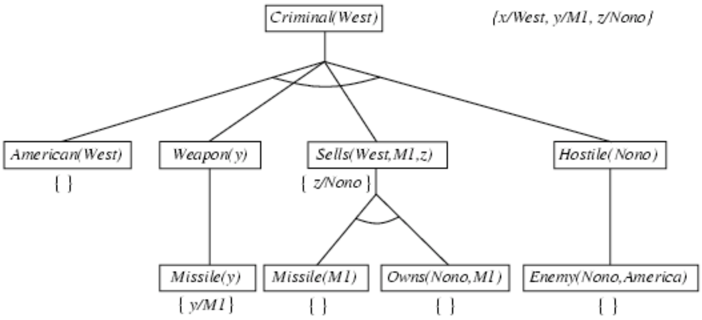
Backward chaining example



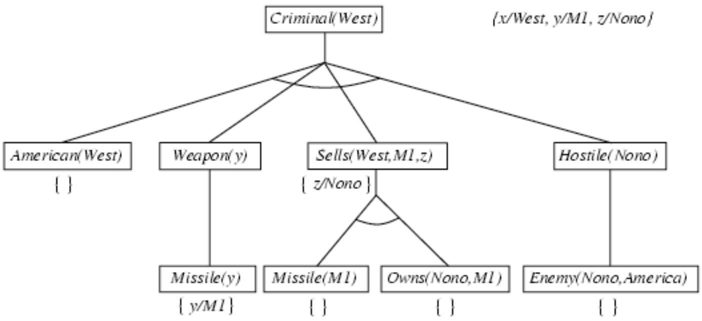
Backward chaining example



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

- Logic programming
  - **Identify problem**
  - **Assemble information**
  - **<coffee break>**
  - **Encode info in KB**
  - **Encode problem instances as facts**
  - **Ask queries**
  - **Find false facts.**
- Procedural programming
  - **Indentify problem**
  - **Assemble information**
  - **Figure out solution**
  - **Program solution**
  - **Encode problem instance as data**
  - **Apply program to data**
  - **Debug procedural errors**

Should be easier to debug Capital(NY, US) than x=x+2

Logic programming: Prolog

- *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).
```

- Efficient unification and retrieval of matching clauses.
- 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:

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

  where `Unify`$(l_i, \neg m_j) = \theta$.
- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,

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

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

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

Conversion to CNF

- Everyone who loves all animals is loved by someone:
  $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
- Eliminate biconditionals and implications
  $\forall x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$
- 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.

- Standardize variables: each quantifier should use a different one:
  ∀x [∃y *Animal*(*y*) ∧ ¬*Loves*(*x*,*y*)] ∨ [∃z *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:
  ∀x [*Animal*(*F*(*x*)) ∧ ¬*Loves*(*x*,*F*(*x*))] ∨ *Loves*(*G*(*x*),*x*)

- Drop universal quantifiers:
  [*Animal*(*F*(*x*)) ∧ ¬*Loves*(*x*,*F*(*x*))] ∨ *Loves*(*G*(*x*),*x*)

- Distribute ∨ over ∧ :
  [*Animal*(*F*(*x*)) ∨ *Loves*(*G*(*x*),*x*)] ∧ [¬*Loves*(*x*,*F*(*x*)) ∨ *Loves*(*G*(*x*),*x*)]

Resolution proof: definite clauses