Artificial Intelligence

Inference in First-Order Logic

Christopher Simpkins

KENNESAW STATE UNIVERSITY

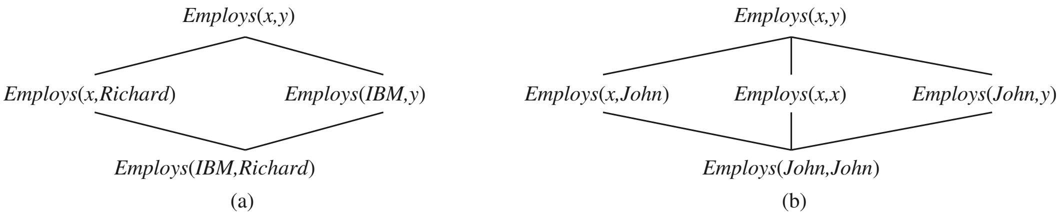
Propositional vs. First-Order Logc

Foo

Unification and First-Order Inference

```
function UNIFY(x, y, θ=empty) returns a substitution to make x and y identical, or failure
   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

function UNIFY-VAR(var, x, θ) returns a substitution
   if {var/val} ∈ θ for some val then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ for some val then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {var/x} to θ
```

Unification and First-Order Inference

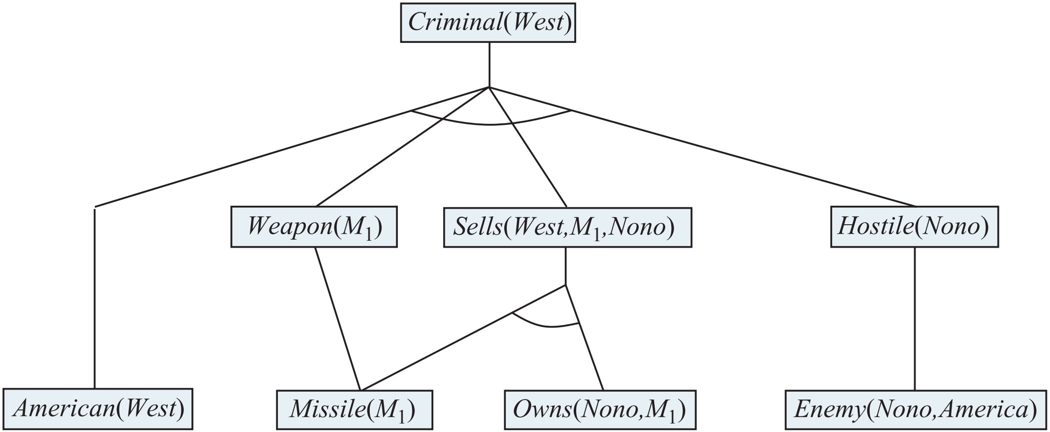


Forward Chaining

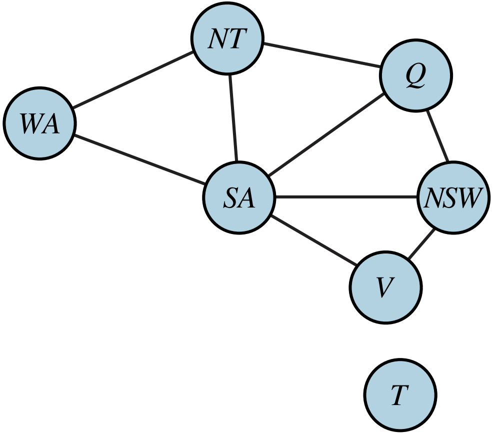
function FOL-FC-ASK(KB, α) **returns** a substitution or *false*
  inputs: KB, the knowledge base, a set of first-order definite clauses
    α, the query, an atomic sentence

  while *true* **do**
    $new \leftarrow \{\}$ // *The set of new sentences inferred on each iteration*
    for each *rule* **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
        for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or *new* **then**
          add q' to *new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *failure* **then return** ϕ
    if $new = \{\}$ **then return** *false*
    add *new* to KB

Forward Chaining



Forward Chaining



(a)

$$\begin{aligned}&Diff(wa,nt) \land Diff(wa,sa) \land \\ &\quad Diff(nt,q) \land Diff(nt,sa) \land \\ &\quad Diff(q,nsw) \land Diff(q,sa) \land \\ &\quad Diff(nsw,v) \land Diff(nsw,sa) \land \\ &\quad Diff(v,sa) \Rightarrow Colorable()\end{aligned}$$

$$Diff(Red,Blue) \quad Diff(Red,Green)$$
$$Diff(Green,Red) \quad Diff(Green,Blue)$$
$$Diff(Blue,Red) \quad Diff(Blue,Green)$$

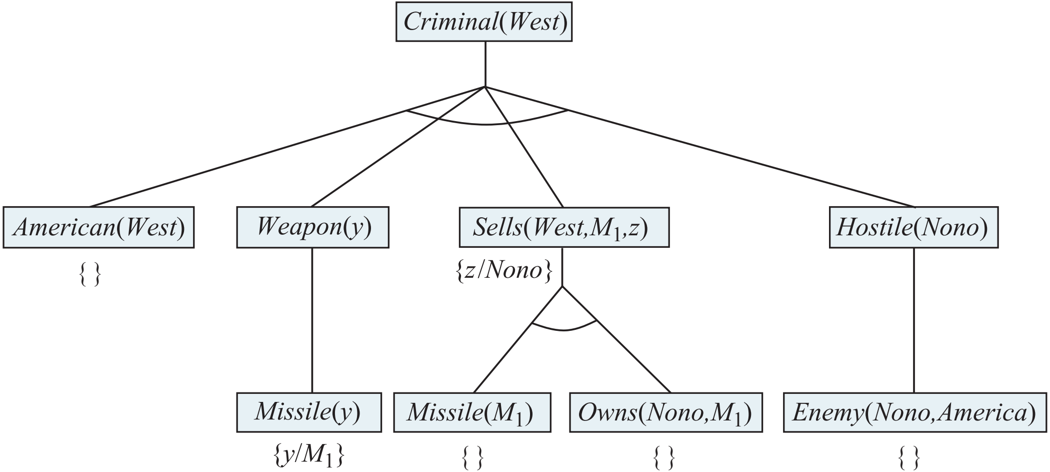
(b)

Backward Chaining

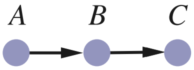
```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

function FOL-BC-OR(KB, goal, θ) returns a substitution
  for each rule in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs ⇒ rhs) ← STANDARDIZE-VARIABLES(rule)
    for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
      yield θ′

function FOL-BC-AND(KB, goals, θ) returns a substitution
  if θ = failure then return
  else if LENGTH(goals) = 0 then yield θ
  else
    first,rest ← FIRST(goals), REST(goals)
    for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
      for each θ″ in FOL-BC-AND(KB, rest, θ′) do
        yield θ″
```

KENNESAW STATE UNIVERSITY

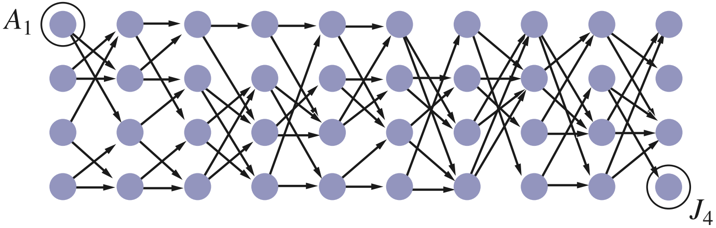
Backward Chaining



Logic Programming

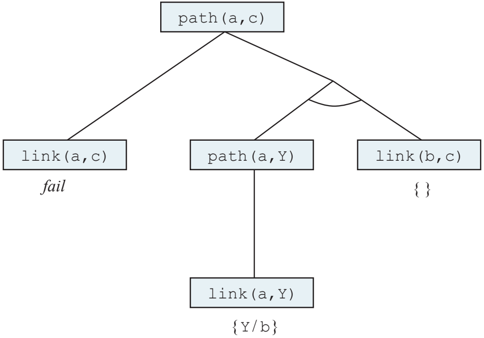


(a)

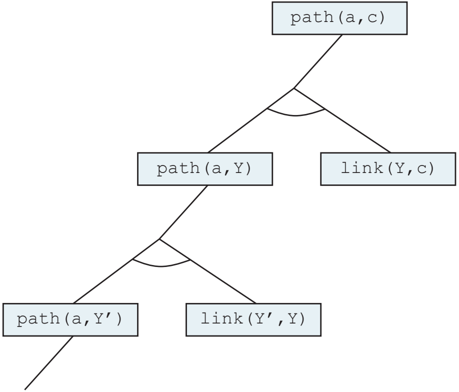


(b)

Logic Programming

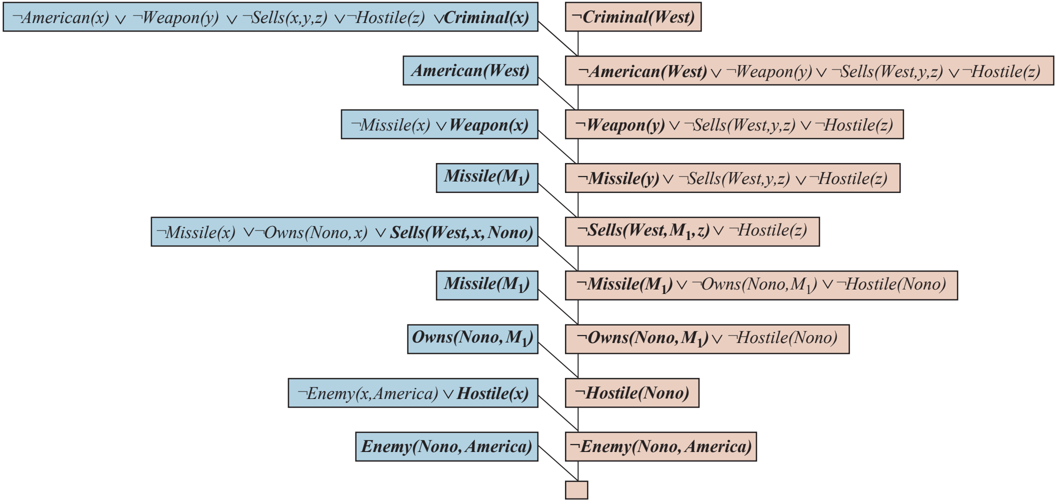


(a)

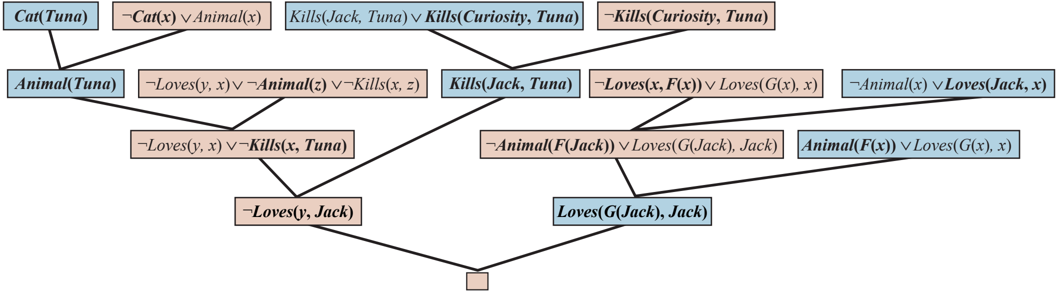


(b)

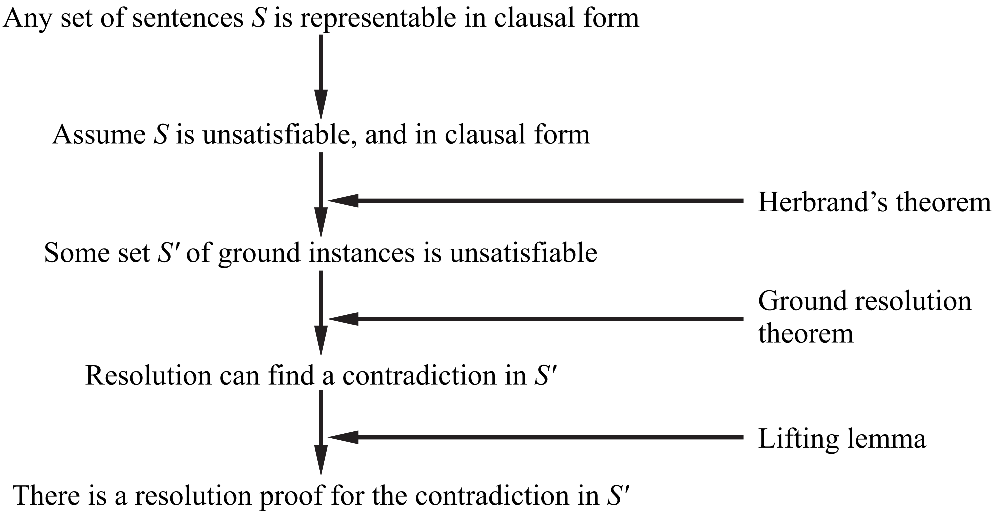
Resolution


¬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(M_1)
¬Missile(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
¬Missile(x) ∨ ¬Owns(Nono,x) ∨ Sells(West,x,Nono)
¬Sells(West,M_1,z) ∨ ¬Hostile(z)
Missile(M_1)
¬Missile(M_1) ∨ ¬Owns(Nono,M_1) ∨ ¬Hostile(Nono)
Owns(Nono,M_1)
¬Owns(Nono,M_1) ∨ ¬Hostile(Nono)
¬Enemy(x,America) ∨ Hostile(x)
¬Hostile(Nono)
Enemy(Nono,America)
¬Enemy(Nono,America)

Resolution


Cat(Tuna)
¬Cat(x) ∨ Animal(x)
Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
¬Kills(Curiosity, Tuna)
Animal(Tuna)
¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z)
Kills(Jack, Tuna)
¬Loves(x, F(x)) ∨ Loves(G(x), x)
¬Animal(x) ∨ Loves(Jack, x)
¬Loves(y, x) ∨ ¬Kills(x, Tuna)
¬Animal(F(Jack)) ∨ Loves(G(Jack), Jack)
Animal(F(x)) ∨ Loves(G(x), x)
¬Loves(y, Jack)
Loves(G(Jack), Jack)

Completeness



KENNESAW STATE UNIVERSITY

G{"o}del's Incompleteness Theorem

Foo