

CmpE 540 Principles of Artificial Intelligence

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Inference in First-Order Logic

Chapter 9
(Based mostly on the course slides
from <http://aima.cs.berkeley.edu/> and
<http://www.cmpe.boun.edu.tr/~akin/>)

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., $\forall x \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields:
 $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
 $\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$

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., $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ yields:

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

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

Reduction to propositional inference

Suppose the KB contains just the following:

$\forall x \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
 $\text{King}(\text{John})$
 $\text{Greedy}(\text{John})$
 $\text{Brother}(\text{Richard}, \text{John})$

- Instantiating the universal sentence in **all possible** ways, we have:

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
 $\text{King}(\text{John})$
 $\text{Greedy}(\text{John})$
 $\text{Brother}(\text{Richard}, \text{John})$

- The new KB is **propositionalized**: proposition symbols are

$\text{King}(\text{John})$, $\text{Greedy}(\text{John})$, $\text{Evil}(\text{John})$, $\text{King}(\text{Richard})$, 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., $\text{Father}(\text{Father}(\text{Father}(\text{John})))$

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 generates lots of irrelevant sentences.
- E.g., from:
 - $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
 - $\text{King}(\text{John})$
 - $\forall y \text{ Greedy}(y)$
 - $\text{Brother}(\text{Richard}, \text{John})$
- It seems obvious that $\text{Evil}(\text{John})$, but propositionalization produces lots of facts such as $\text{Greedy}(\text{Richard})$ 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 $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$\text{Knows}(\text{John}, x)$	$\text{Knows}(\text{John}, \text{Jane})$	
$\text{Knows}(\text{John}, x)$	$\text{Knows}(y, \text{OJ})$	
$\text{Knows}(\text{John}, x)$	$\text{Knows}(y, \text{Mother}(y))$	
$\text{Knows}(\text{John}, x)$	$\text{Knows}(x, \text{OJ})$	

- Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$

Unification

- We can get the inference immediately if we can find a substitution θ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$ \square

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$\text{Knows}(\text{John}, x)$	$\text{Knows}(\text{John}, \text{Jane})$	$\{x/\text{Jane}\}$
$\text{Knows}(\text{John}, x)$	$\text{Knows}(y, \text{OJ})$	
$\text{Knows}(\text{John}, x)$	$\text{Knows}(y, \text{Mother}(y))$	
$\text{Knows}(\text{John}, x)$	$\text{Knows}(x, \text{OJ})$	

- Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$ \square

Unification

- We can get the inference immediately if we can find a substitution θ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$ \square

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$\text{Knows}(\text{John}, x)$	$\text{Knows}(\text{John}, \text{Jane})$	$\{x/\text{Jane}\}$
$\text{Knows}(\text{John}, x)$	$\text{Knows}(y, \text{OJ})$	$\{x/\text{OJ}, y/\text{John}\}$
$\text{Knows}(\text{John}, x)$	$\text{Knows}(y, \text{Mother}(y))$	
$\text{Knows}(\text{John}, x)$	$\text{Knows}(x, \text{OJ})$	

- Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$

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 $Greedy(y)$ \square

$\theta = \{x/John, y/John\}$ 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(z₁₇,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 $Greedy(y)$ \square

$\theta = \{x/John, y/John\}$ 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(z₁₇,OJ)

Unification

- To unify $Knows(John,x)$ and $Knows(y,z)$,
 $\theta = \{y/John, x/z\}$ or $\theta = \{y/John, x/John, z/John\}$
- 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,  $\theta$ ) returns a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound
       y, a variable, constant, list, or compound
        $\theta$ , the substitution built up so far

if  $\theta = failure$  then return failure
else if  $x = y$  then return  $\theta$ 
else if VARIABLE?(x) then return UNIFY-VAR(x, y,  $\theta$ )
else if VARIABLE?(y) then return UNIFY-VAR(y, x,  $\theta$ )
else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(ARGs[x], ARGs[y], UNIFY(OP[x], OP[y],  $\theta$ ))
else if LIST?(x) and LIST?(y) then
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y],  $\theta$ ))
else return failure
    
```

The unification algorithm

```

function UNIFY-VAR(var, x,  $\theta$ ) returns a substitution
inputs: var, a variable
       x, any expression
        $\theta$ , the substitution built up so far
if {var/val}  $\in$   $\theta$  then return UNIFY(val, x,  $\theta$ )
else if {x/val}  $\in$   $\theta$  then return UNIFY(var, val,  $\theta$ )
else if OCCUR-CHECK?(var, x) then return failure
else return add {var/x} to  $\theta$ 
    
```

Generalized Modus Ponens (GMP)

$p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)$ where $p_i\theta = p_i'\theta$ for all i
 $q\theta$

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
 $p_1', \dots, p_n', (p_1 \wedge \dots \wedge p_n \Rightarrow q) \vdash q\theta$
 provided that $p_i\theta = p_i'\theta$ for all i
 - Lemma: For any sentence p , we have $p \vdash p\theta$ by UI
1. $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \vdash (p_1 \wedge \dots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \dots \wedge p_n\theta \Rightarrow q\theta)$
 2. $p_1', \dots, p_n' \vdash p_1' \wedge \dots \wedge p_n' \vdash p_1'\theta \wedge \dots \wedge p_n'\theta$
 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:

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

Nono ... has some missiles, i.e.,

$\exists x Owns(Nono,x) \wedge Missile(x)$:

$Owns(Nono,M_1)$ and $Missile(M_1)$

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

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons: \square

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile":

$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American ... \square

$American(West)$

The country Nono, an enemy of America ... \square

$Enemy(Nono,America)$ \square

Forward chaining algorithm

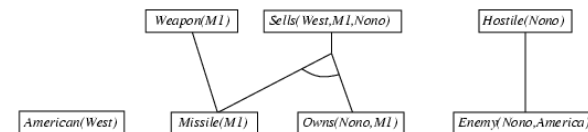
```

function FOL-FC-ASK( $KB, \alpha$ ) returns a substitution or false
  new  $\leftarrow \{ \}$ 
  repeat until new is empty
    for each sentence  $r$  in  $KB$  do
      ( $p_1 \wedge \dots \wedge p_n \Rightarrow q$ )  $\leftarrow$  STANDARDIZE-APART( $r$ )
      for each  $\theta$  such that ( $p_1 \wedge \dots \wedge p_n$ ) $\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$ 
        for some  $p'_1, \dots, p'_n$  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
  
```

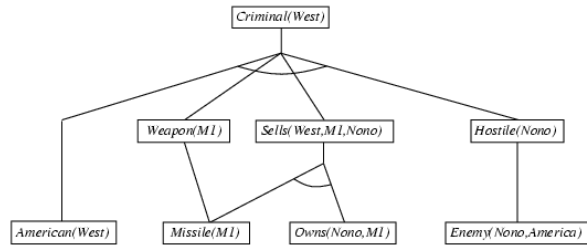
Forward chaining proof

$American(West)$ $Missile(M1)$ $Owns(Nono,M1)$ $Enemy(Nono,America)$

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 $k-1$
 \Rightarrow match each rule whose premise contains a newly added positive literal

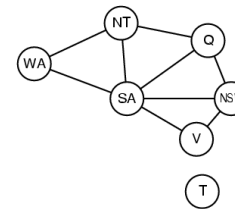
Matching itself can be expensive:

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**

Hard matching example



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

$$\begin{aligned} &Diff(Red,Blue) \quad Diff(Red,Green) \\ &Diff(Green,Red) \quad Diff(Green,Blue) \\ &Diff(Blue,Red) \quad Diff(Blue,Green) \end{aligned}$$

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

Backward chaining algorithm

```

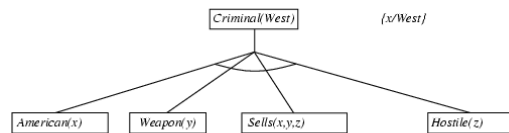
function FOL-BC-ASK(KB, goals,  $\theta$ ) returns a set of substitutions
inputs: KB, a knowledge base
       goals, a list of conjuncts forming a query
        $\theta$ , the current substitution, initially the empty substitution {}
local variables: ans, a set of substitutions, initially empty
if goals is empty then return { $\theta$ }
 $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\text{goals}))$ 
for each r in KB where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
   $\text{ans} \leftarrow \text{FOL-BC-ASK}(\text{KB}, [p_1, \dots, p_n] \text{REST}(\text{goals}), \text{COMPOSE}(\theta, \theta')) \cup \text{ans}$ 
return ans
    
```

$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2,$
 $\text{SUBST}(\theta_1, p))$

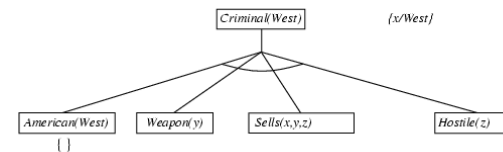
Backward chaining example

Criminal(West)

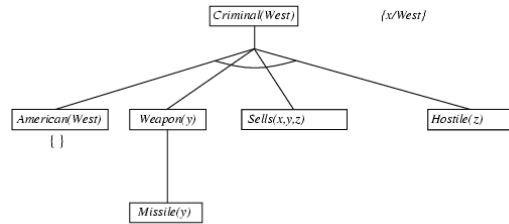
Backward chaining example



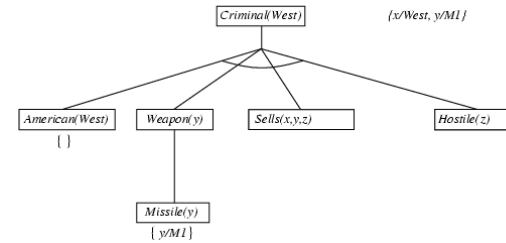
Backward chaining example



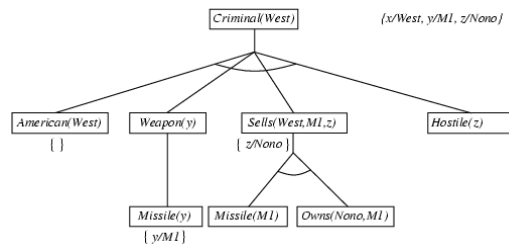
Backward chaining example



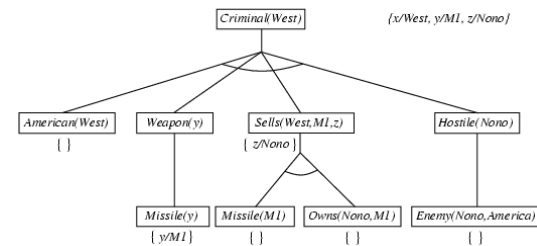
Backward chaining example



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)
- Program = set of clauses = head :- literal₁, ... 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=[]`

Procedure for doing Resolution Proof

- Negate the theorem to be proved and add the result to the list of axioms.
- Put the list of axioms into clause form.
- Until the empty clause, Nil, is produced or there is no resolvable pair of clauses, find the resolvable clauses, resolve them, and add the result to the list of clauses.
- If the empty clause is produced, report that the theorem is TRUE. If there are no resolvable clauses, report that the theorem is false.

Resolution: brief summary

- Full first-order version:

$$\frac{l_1 \vee \dots \vee l_k \quad m_1 \vee \dots \vee m_n}{(l_1 \vee \dots \vee l_{k-1} \vee l_{k+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta}$$

where $\text{Unify}(l_i, \neg m_j) = \theta$.

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

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

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

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

Conversion to CNF

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

Conversion to CNF contd.

- Standardize variables: each quantifier should use a different one

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

- Skolemize: a more general form of existential instantiation.

Each existential variable is replaced by a **Skolem function** of the enclosing universally quantified variables:

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

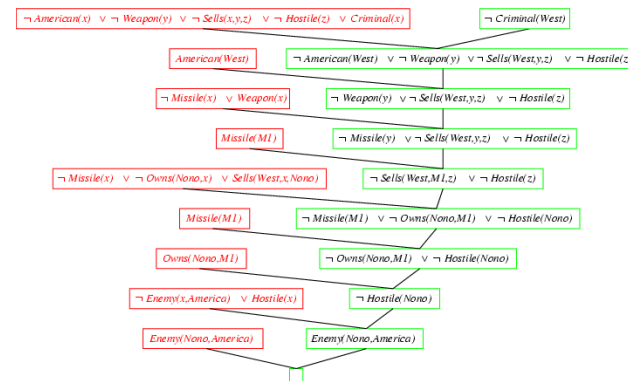
- Drop universal quantifiers:

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

- Distribute \vee over \wedge :

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

Resolution proof: definite clauses



Another Resolution Example

Theorem:

Feathers(Cikcik)
 $\forall x[\text{Feathers}(x) \Rightarrow \text{Bird}(x)]$
 $\therefore \text{Bird}(\text{Cikcik})$

Proof:

Feathers(Cikcik) (Axiom 1)
 $\neg \text{Feathers}(\text{Cikcik}) \vee \text{Bird}(\text{Cikcik})$ (axiom 2 & u.g.)
 $\neg \text{Bird}(\text{Cikcik})$ (negation of conclusion)
 $\text{Bird}(\text{Cikcik})$ (Resolve 1 and 2)
 Nil (Resolve 3 and 4 \Rightarrow Contradiction)

Conversion to CNF

Given

- The brick is on something that is not a pyramid.
- There is nothing that the brick is on and that is on the brick as well.
- There is nothing that is not a brick and the same thing as a brick.

$\forall x[\text{Brick}(x) \Rightarrow (\exists y[\text{On}(x,y) \wedge \neg \text{Pyramid}(y)]$
 $\wedge \neg \exists y[\text{On}(x,y) \wedge \text{On}(y,x)]$
 $\wedge \forall y[\neg \text{Brick}(y) \Rightarrow \neg \text{Equal}(x,y)])]$

Eliminate Implications

(Substitute $\neg E1 \vee E2$ for $E1 \Rightarrow E2$).

$\forall x[\neg \text{Brick}(x) \vee (\exists y[\text{On}(x,y) \wedge \neg \text{Pyramid}(y)]$
 $\wedge \neg \exists y[\text{On}(x,y) \wedge \text{On}(y,x)]$
 $\wedge \forall y[\neg(\neg \text{Brick}(y)) \vee$
 $\neg \text{Equal}(x,y)])]$

Move negations down to atomic formulas

(Use De Morgan's laws and negation of quantifiers)

$\forall x[\neg \text{Brick}(x) \vee (\exists y[\text{On}(x,y) \wedge \neg \text{Pyramid}(y)]$
 $\wedge \forall y[\neg \text{On}(x,y) \vee \neg \text{On}(y,x)]$
 $\wedge \forall y[\text{Brick}(y) \vee \neg \text{Equal}(x,y)])]$

Purge existential quantifiers

Use skolem functions.

- $\exists y[\text{On}(x,y) \wedge \neg \text{Pyramid}(y)]$
 - $y = \text{Magic}(x)$
 - $\text{On}(x, \text{Magic}(x)) \wedge \neg \text{Pyramid}(\text{Magic}(x))$
 - $\text{Magic}(x) = \text{Support}(x)$
- $$\forall x[\neg \text{Brick}(x) \vee (\text{On}(x, \text{Support}(x)) \wedge \neg \text{Pyramid}(\text{Support}(x)))$$
- $$\wedge \forall y[\neg \text{On}(x,y) \vee \neg \text{On}(y,x)]$$
- $$\wedge \forall y[\text{Brick}(y) \vee \neg \text{Equal}(x,y)]]$$

Rename Variables

- (No two variables should be the same. Scope of universal quantifiers!)

$$\forall x[\neg \text{Brick}(x) \vee ((\text{On}(x, \text{Support}(x)) \wedge \neg \text{Pyramid}(\text{Support}(x)))$$

$$\wedge \forall y[\neg \text{On}(x,y) \vee \neg \text{On}(y,x)]$$

$$\wedge \forall z[\text{Brick}(z) \vee \neg \text{Equal}(x, z)])]$$

Move Universal Quantifiers to the left

$$\forall x \forall y \forall z[$$

$$\neg \text{Brick}(x) \vee ((\text{On}(x, \text{Support}(x)) \wedge \neg \text{Pyramid}(\text{Support}(x)))$$

$$\wedge \neg \text{On}(x,y) \vee \neg \text{On}(y,x)$$

$$\wedge \text{Brick}(z) \vee \neg \text{Equal}(x, z))]$$

Move the disjunctions down to the literals

- Use distribution laws

$$\forall x \forall y \forall z[$$

$$(\neg \text{Brick}(x) \vee (\text{On}(x, \text{Support}(x)) \wedge \neg \text{Pyramid}(\text{Support}(x))))$$

$$\wedge (\neg \text{Brick}(x) \vee \neg \text{On}(x,y) \vee \neg \text{On}(y,x))$$

$$\wedge (\neg \text{Brick}(x) \vee \text{Brick}(z) \vee \neg \text{Equal}(x, z))]$$

Move the disjunctions down to the literals (Cont.)

$$\forall x \forall y \forall z [$$
$$\begin{aligned} &(\neg \text{Brick}(x) \vee \text{On}(x, \text{Support}(x))) \\ &\wedge (\neg \text{Brick}(x) \vee \neg \text{Pyramid}(\text{Support}(x))) \\ &\wedge (\neg \text{Brick}(x) \vee \neg \text{On}(x, y) \vee \neg \text{On}(y, x)) \\ &\wedge (\neg \text{Brick}(x) \vee \text{Brick}(z) \vee \neg \text{Equal}(x, z)) \end{aligned}$$

Eliminate the conjunctions

- Write each part of the conjunction as a separate axiom.

$$\begin{aligned} &\forall x [\neg \text{Brick}(x) \vee \text{On}(x, \text{Support}(x))] \\ &\forall x [\neg \text{Brick}(x) \vee \neg \text{Pyramid}(\text{Support}(x))] \\ &\forall x \forall y [\neg \text{Brick}(x) \vee \neg \text{On}(x, y) \vee \neg \text{On}(y, x)] \\ &\forall x \forall z [\neg \text{Brick}(x) \vee \text{Brick}(z) \vee \neg \text{Equal}(x, z)] \end{aligned}$$

Rename all the variables

$$\begin{aligned} &\forall x [\neg \text{Brick}(x) \vee \text{On}(x, \text{Support}(x))] \\ &\forall w [\neg \text{Brick}(w) \vee \neg \text{Pyramid}(\text{Support}(w))] \\ &\forall u \forall y [\neg \text{Brick}(u) \vee \neg \text{On}(u, y) \vee \neg \text{On}(y, u)] \\ &\forall v \forall z [\neg \text{Brick}(v) \vee \text{Brick}(z) \vee \neg \text{Equal}(v, z)] \end{aligned}$$

Purge the universal quantifiers

$$\begin{aligned} &\neg \text{Brick}(x) \vee \text{On}(x, \text{Support}(x)) \\ &\neg \text{Brick}(w) \vee \neg \text{Pyramid}(\text{Support}(w)) \\ &\neg \text{Brick}(u) \vee \neg \text{On}(u, y) \vee \neg \text{On}(y, u) \\ &\neg \text{Brick}(v) \vee \text{Brick}(z) \vee \neg \text{Equal}(v, z) \end{aligned}$$

How to select the right clauses to resolve?

- **Unit preference strategy:** gives preference to resolutions involving the clauses with the smallest number of literals.
- **Set of support strategy:** allows only resolutions involving the negated theorem or theorem or new clauses derived, directly or indirectly, using the negated theorem.
- **Breadth-first strategy:** first resolves all possible pairs of the initial clauses, then resolves all possible pairs of the resulting set together with the initial set, level by level.

Properties of these strategies

- All these strategies are complete because they are guaranteed to find a proof if the theorem logically follows from the axioms.
- All strategies are subject to the combinatorial explosion problem.
- All search strategies are subject to a version of the halting problem, for search is not guaranteed to terminate unless there actually is a proof.

Analysis

- All complete proof procedures of the first-order predicate calculus are subject to the halting problem.
- Gödel's Completeness Theorem.
- If a sentence is **true** given a set of axioms, there is a procedure that will determine this.
- If the sentence is **false**, then there is no guarantee that a procedure will ever determine this—i.e., it may never halt.
- Complete proof procedures are **semidecidable**.