Basic Things

Tautology	=>	Always true	eg	the sea is wet
Contradiction	=>	Always false	eg	the sea is dry
Satisfiable	=>	Sometimes True	eg	hats are blue

a -> b = ¬a v b

WFF – Well Formed Formula

Variables (Atomic propositions) are wff's
if s and t are wff then so is
(¬ t)
(s ^ t)
(s v t)
(s -> t)

Interpretations

Assignments of true and false to variables

eg

v(p) = true
v(q)= false

v(p v q)
=> true v false => true

v is a model for the wff a is v(a) is true

A Model is an assignment of true and false to the variables when the outcome is true.

ALSO
Satisfiable iff it has a models
Tautology iff ALL interpretations are models
contradiction iff it has NO Models

Basically just interpret iff as if as its when theres a wff interpretation kinda thing.

Logical Equivalence

Same truth table
a = b (TECHNICALLY 3 LINE EQUALS)

Logical Consequence

Every model of B is a model of A denoted B J A

SETS these still apply basically the same, IS the same

Logical Consequence

S => L
if all models of S are models of L then YEY true

Resolution

You have a collection of statements which all apply to the same thing

dog v cat (dog or cat)
dog v ¬cat (dog or not cat)

so from that you can deduce its NOT a cat so you can just throw that away

=> dog

also

dog v cat
¬cat

same applies you can just throw cat away

=> dog

Proof By Refutation

If you can prove the -ve / negated / inverted / opposite version of something is false then it must be true

eg: the sea is wet
therefore we write the sea is not wet
thats obviously false
=> the sea is wet
seems stupid but works well for complicated things

Right way you do it
S => {L} only iff S u {¬L} is false <- Thats UNION not u

{p,q} => {p ^ q}
What this is saying is that p^q is a logical consequence of p, q

Therefore to prove this we need to prove p^q and we have the other statements p, q
therefore we need to negate what were trying to prove and union it all
{p, q, ¬(p^q)}

noooo theres no disjunction (throw awayable stuff) so convert to cnf
(break the line change the sign !) to put or's in
{p, q, ¬p v ¬q}

resolve p and the big thing on right
{p, q, ¬p v ¬q, ¬q} <- notice p was thrown away

resolve q and the thing on right
{p, q, ¬p v ¬q, □} <- notice ¬q was thrown away and a square box was put in instead of a blank term.

Hey this means we have a square boxy thing AKA its false => original thing is true

CNF – Conjunctive Normal Form

Literal - p or ¬p where p is atomic (variable)
Clause - disjunction of literals
Horn Clause - Max 1 Positive literal
Denial - No Positive literals

Linear Resolution

First Order Logic

Declaring Variables

$\forall$D For All D

$\exists D$ Any number of D

Making Predicate Things

Take the condition and choose some thing sensible

ie

All Cats hate some Dogs

so hate(Who, What) / hate(A,B) - A hates B

=> All Cats hate some Dogs

1. Declare any variables
2. Write the statements

$\forall C \exists D$ hate(C,D)

Resolution in FOL

Example

Like X or wont eat X
$\forall X$ like(X) v ¬eat(X)
like(fish)
eat(fish)

{like(fish), eat(fish), ¬(like(X) v ¬eat(X))}

resolve 1 and 3
{like(fish), eat(fish), ¬like(X), eat(X)}
(this limits X to fish, X is unified to fish)

Converting Prolog to FOL

Say we have prolog

f(X, Y) :- m(x), p(X, Y).

in FOL this becomes

$\forall X \forall Y$ f(X, Y) ← m(X) ^ p(X,Y)

Example

notel(X, nil).
notel(X, tr(Y,L,R)) :- dif(X,Y), notel(X,L), notel(X,R).

$\forall X$ notel(X, nil)
($\forall X \forall Y \forall L \forall R$ notel(X, tr(Y,L,R))) ←(dif(X, Y) ^ notel(X,L) ^ notel(X,R))

Bigger Example !

fun(0,0).
fun(s(s(X)), s(R)) :- fun(X, R).

gives us the things
fun(0, 0) ←
fun(s(s(X)), s(R)) ← fun(X, R)

we have fun(s(s(s(s(0)))), s(s(0)))

NOTE THE USE OF UNIFIERS IN THIS SO READ THAT FIRST
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(s(s(0)))), s(s(0))) }
resolve 2, 3
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(s(s(0)))), s(s(0))), ¬fun(s(s(0)), s(0))}

resolve 2, 4
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(s(s(0)))), s(s(0))), ¬fun(0, 0)}
resolve 1, 4
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(s(s(0)))), s(s(0))), □}

Its False its a contradiction therefor original statement was true.

fun(0, 0) ←
fun(s(s(X)), s(R)) ← fun(X, R)

fun(s(s(X)), X)
G1 => X
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(G1)), G1)}   {G1 / s(R)}
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(R), R)}   { R/ s(T) }
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(T)), s(T))}   {T / 0}
=> G1 / s(R) => R/s(T) => T / 0 => R / s(0) => G1/s(s(0))

fun(s(s(X)), X)
G1 => X
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(G1)), G1)}   {G1 / s(R)}
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(R), R)}   { R/ s(T) }
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(s(s(T)), s(T))}   {T / 0}

fun(X, s(X))
X / G1
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(G1, s(G1))}   {G1 / s(s(X)}
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(X, s(s(X)))}   {X / s(s(X2)}
{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X,R), ¬fun(X2, s(s(X2)))}   {X2 / s(s(X3)}

Just gets bigger with the gap not closing

Another Example

plus(0, X, X).
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).

gives us the things
plus(0, X, X) ←
plus(s(X), Y, s(Z)) ← plus(X, Y, Z)

we have
plus(s(s(0)), s(s(0)), s(s(s(s(0))))).

This can be seen to be
{plus(0,X,X), plus(s(X), Y, s(Z)) v ¬plus(X,Y,Z), ¬plus(s(s(0)), s(s(0)), s(s(s(s(0)))))}

resolve 2, 3
{plus(0,X,X), plus(s(X), Y, s(Z)) v ¬plus(X,Y,Z), ¬plus(s(s(0)), s(s(0)), s(s(s(s(0))))), ¬plus(s(0), s(s(0)), s(s(s(0))))}

resolve 2, 4
{plus(0,X,X), plus(s(X), Y, s(Z)) v ¬plus(X,Y,Z), ¬plus(s(s(0)), s(s(0)), s(s(s(s(0))))), ¬plus(0, s(s(0)), s(s(0)))}

resolve 1,4
{plus(0,X,X), plus(s(X), Y, s(Z)) v ¬plus(X,Y,Z), ¬plus(s(s(0)), s(s(0)), s(s(s(s(0))))), □}

Finding unifiers

Variable / Value
=>

p(X)	p(a)	->	X/a		
p(s(0))	p(Y)	->	Y/s(0)		
p(b,s(0))	p(X,Y)	->	X/b, Y/s(0)		
p(Y,Z)	P(a,a)	->	Y/a, Z/a		
P(Y,Z)	P(Z,Z)	->	Z/Z, Y/Z <- Z/Z same peh ignore	->	Y/Z
P(a)	P(b)	->	NO YOU CANT DO IT		

Compositions if the variable occurs else where you can put them together

P(X,X,B)	p(a,Y,Y)	->	X/a, X/Y, B/Y	->	X/a, Y/a, B/a

P(X)	P(s(0))	->	X/s(0)
P(X)	P(f(a,s(0)))	->	X/f(a,s(0))
P(X)	P(s(Y))	->	X/s(Y)
P(X)	P(f(Y,s(Z)))	->	X/f(Y,s(Z))
P(X)	P(s(X))	->	X/s(X) -> Fails occur check as you cud have X/s(s(s(s(...

f(f(X))	f(g(X))	->	Doesn't work different terms		
f(s(X),s(Z))	f(Y, s(0))	->	Y/s(X), s(Z)/s(0)	->	Y/s(X), Z/0
f(X,X)	f(Y, s(0))	->	X/Y, X/s(0)	->	Y/s(0), X/s(0)

Disagreement Tuple

Work from the left until you find 2 terms that are different thats em

f(s(X),s(Z))	f(Y, s(0))	->	{Y, s(X)}
f(Y,X)	f(Y, s(0))	->	{X, s(0) }

See proper notes for more info on resolution

FOL to CNF

Converting FOL to CNF

Example
∀ *X* (fast(X) → (¬slow(X) ^ ¬immobile(X)) ^ fast(X) ← (¬slow(X) ^ ¬immobile)))

Becomes
¬fast(X) v (¬slow(X) ^ ¬immobile(X)) ^ fast(X) v ¬(¬slow(X) ^ ¬immobile(X))
¬fast(X) v (¬slow(X) ^ ¬immobile(X)) ^ fast(X) v slow(X) v immobile(X)

becomes
{
¬fast(X) v ¬slow(X) ,
¬fast(X) v ¬immobile(X),
fast(X) v slow(X) v immobile(X)
}

Tutorials

a) Which of the following are wffs (A,B,C are atoms; precedence rules are not to be used):

(A v B ^ C)	-> NO	-> (A v (B ^ C))	-> A v B ^ C	
(¬(B ^ C))	-> YES		-> ¬(B ^ C)	
(A v (B ^ C))	-> YES		-> A v B ^ C	
(¬B ^ C)	-> YES		-> ¬B ^ C	
¬(B ^ C)	-> YES		-> ¬(B ^ C)	
A -> (B ^ C)	-> YES		-> A -> B ^ C	
¬(¬B)	-> YES		-> ¬¬B	-> B
(A -> (B -> C))	-> YES		-> A -> (B -> C)	
((A -> B) -> C)	-> YES		-> (A -> B) -> C	
(¬(¬(B)) ^ C)	-> YES		-> ¬¬B ^ C	
((¬A) -> (¬B))	-> YES		-> ¬A -> ¬B	
((A -> ¬¬B) v C)	-> NO double ¬¬		-> (A -> ¬¬B) v C	
(A ^ ((B ^ C))	-> YES		-> A ^ B ^ C	

Unless it rains there will be a severe drought
    ¬rains -> drought

If there is a drought standpipes will be needed
    drought -> standpipes

The house will be finished only if the outstanding bill is paid or if the proprietor works on it himself.

    finished -> paid v works

The company is responsible if and only if the computer was installed since January and is a VAX.

    responsible -> itsnew ^ itsvax

It is not the case that Jack and Jill are both going up the hill.

¬(Jack ^ Jill)

James will work hard and get a first or James will belong to the dramatic society.

First v Drama

I'll be back by two and will bring the shopping if and only if it doesn't rain.

¬rain -> shopping

The car is old but it runs OK.

old ^ ok

(P ^ Q -> R) -> (P -> R) ^ (Q -> R)

((P ^ Q) -> R) -> (P -> R) ^ (Q -> R)

¬(¬(P ^ Q) v R) v (¬P v R) ^ (¬Q v R)

¬(P v Q v R) v (¬P v R) ^ (¬Q v R)

TBC

Translate the following wff into conjunctive normal form:

Move the ANDS out using A v (B ^ C) = (A v B) ^ (A v C)

((rains v snows) -> umbrella)

¬(rains v snows) v umbrella

(¬rains ^ ¬snows) v umbrella

(¬rains v umbrella) v (¬snows v umbrella)

=> (¬rains v umbrella), (¬snows v umbrella)

(P ^ Q -> R) -> (P -> R) ^ (Q -> R)

¬(¬(P ^ Q) v R) v ((¬P v R) ^ (¬Q v R))

((P ^ Q) ^ ¬R) v ((¬P v R) ^ (¬Q v R))

(P ^ Q ^ ¬R) v ((¬P v R) ^ (¬Q v R))

(P v ((¬P v R) ^ (¬Q v R))) ^
(Q v ((¬P v R) ^ (¬Q v R))) ^
(¬R v ((¬P v R) ^ (¬Q v R)))

(P v (¬P v R)) ^ (P v (¬Q v R)) ^
(Q v (¬P v R)) ^ (Q v (¬Q v R)) ^
(¬R v (¬P v R)) ^ (¬R v (¬Q v R))

(T) ^ (P v (¬Q v R)) ^
(Q v (¬P v R)) ^ (T) ^
(T) ^ (T)
=> P v ¬Q v R , Q v ¬P v R

(A -> B) v (B -> A)
¬A v B v ¬B v A => T

Prove (A -> B) v (B -> A) is a tautology
=> ¬A v B v ¬B v A
for it to be a tautology we have {A, B, ¬A, ¬B} as models i s'pose
{A, B, ¬A, ¬B, ¬(¬A v B v ¬B v A)}
{A, B, ¬A, ¬B, ¬(¬A v B v ¬B v A)} resolve 1,5
{A, B, ¬A, ¬B, ¬(B v ¬B v A)} resolve 2, 5
{A, B, ¬A, ¬B, ¬(B v A)} resolve 3, 5
{A, B, ¬A, ¬B, ¬(B)} resolve 2, 5
{A, B, ¬A, ¬B, □} False therefor original was true its a tautology

{A, B, ¬A, ¬B, ¬((¬A v B) v (¬B v A))}
{A, B, ¬A, ¬B, (¬(¬A v B) ^ ¬(¬B v A))}
{A, B, ¬A, ¬B, ((A ^ ¬B) ^ (B ^ ¬A))}

show
(P -> Q) ^ (R -> S) |= (P -> S) v (R -> Q)
(¬P v Q) ^ (¬R v S) |= (¬P v S) v (¬R v Q)

{(¬P v Q) ^ (¬R v S), ¬((¬P v S) v (¬R v Q)) }
{(¬P v Q) ^ (¬R v S), ¬(¬P v S) ^ ¬(¬R v Q) }

{(¬P v Q) ^ (¬R v S), (P ^ ¬S) ^ (R ^ ¬Q) }
{(¬P v Q) ^ (¬R v S), P ^ ¬S ^ R ^ ¬Q }

{(¬P v Q), (¬R v S), P, ¬S, R, ¬Q }
resolve 1, 3   {Q, (¬R v S), P, ¬S, R, ¬Q }
resolve 1, 6   {□, (¬R v S), P, ¬S, R, ¬Q }
a blank clause, its false => it true

show that lay_eggs -> feathers is a logical consequence of
((bird -> (feathers ^ flies)) and ((¬reptile ^ lays_eggs) -> bird) ^ ¬reptile)
((¬bird v (feathers ^ flies)) ^ (¬(¬reptile ^ lays_eggs) v bird) ^ ¬reptile)
(((¬bird v feathers) ^ (¬bird v flies)) ^ ((reptile v ¬lays_eggs v bird) ^ ¬reptile)
{¬bird v feathers, ¬bird v flies, reptile v ¬lays_eggs v bird, ¬reptile, ¬(¬lay_eggs v feathers)}
{¬bird v feathers, ¬bird v flies, reptile v ¬lays_eggs v bird, ¬reptile, lay_eggs ^ ¬feathers}
{¬bird v feathers, ¬bird v flies, reptile v ¬lays_eggs v bird, ¬reptile, lay_eggs, ¬feathers}
resolve 1, 6   {¬bird, ¬bird v flies, reptile v ¬lays_eggs v bird, ¬reptile, lay_eggs, ¬feathers}
resolve 5, 3   {¬bird, ¬bird v flies, reptile v bird, ¬reptile, lay_eggs, ¬feathers}
resolve 1, 3   {¬bird, ¬bird v flies, reptile, ¬reptile, lay_eggs, ¬feathers}
resolve 3, 4   {¬bird, ¬bird v flies, □, ¬reptile, lay_eggs, ¬feathers}
a blank cause therefore false so original was true

show feathers is a logical consequence of
(silly —› ¬silly) **and** (¬ silly —› silly)

{(silly -> ¬silly), (¬silly -> silly), ¬feathers}
{¬silly v ¬silly, ¬¬silly v silly, ¬feathers}
{¬silly v ¬silly, silly v silly, ¬feathers}
{¬silly, silly, ¬feathers}

Therefore we can resolve silly with ¬silly
{□, silly, ¬feathers}
This is due to the original statements contradicting each other, ie its a contradiction as it cant be SILLY and NOT SILLY.

1. Nobody is both male and female.

¬(∃*X* male(X) ^ female(X))

2. Anyone who likes himself likes someone.

∀*X* likes(X, X) →∃*Y* likes(X, Y)

3. None of Mary's lovers' lovers love her. ??

4. Every prize was won by a girl.

∀*P* prize(P) →∃*G* girl(G) ^ won(P, G)

5. One girl won all the prizes.

∃*G*∀*P* prize(P) ^ girl(G) ^ won(P, G)

WRONG FULLY MAYBE WHO KNOWS NOT ME

∀*X* (brick(X) → ((∃*Y* (on(X, Y) ^ ¬pyramix(Y))) ^
(¬ ∃ Y (on(X, Y) ^ on (Y,X))) ^
(∀*Y* (¬brick(Y) → ¬equal(X,Y)))))

∀*X* (¬brick(X) v ((∃*Y* (on(X, Y) ^ ¬pyramix(Y))) ^
(∀ Y (¬on(X, Y) v ¬on (Y,X))) ^
(∀*Y* (brick(Y) v ¬equal(X,Y)))))

Rename Variables

∀*X* (¬brick(X) v ((∃*Y* (on(X, Y) ^ ¬pyramix(Y))) ^
(∀ W (¬on(X, W) v ¬on (W,X))) ^
(∀*Z* (brick(Z) v ¬equal(X,Z)))))

Right now we can just move foralls to the front BUT exists are important

need to use Skolem functions

so just say Y = something(X)

∀*X*∀*W*∀*Z* (¬brick(X) v ((on(X, fred(X)) ^ ¬pyramix(fred(X))) ^
(¬on(X, W) v ¬on (W,X)) ^ (brick(Z) v ¬equal(X,Z)))

∀*X*∀*W*∀*Z* (¬brick(X) v ((on(X, fred(X)) ^ ¬pyramix(fred(X)))) ^
(¬brick(X) v (¬on(X, W) v ¬on (W,X))) ^ (¬brick(X) v (brick(Z) v ¬equal(X,Z)))

((¬brick(X) v on(X, fred(X))) ^
(¬brick(X) v ¬pyramix(fred(X))) ^
(¬brick(X) v ¬on(X, W) v ¬on (W,X)) ^
(¬brick(X) v brick(Z) v ¬equal(X,Z)))

((¬brick(X) v on(X, fred(X))) ^
(¬brick(X) v ¬pyramix(fred(X))) ^
(¬brick(X) v ¬on(X, W) v ¬on (W,X)) ^
(¬brick(X) v brick(Z) v ¬equal(X,Z)))

Past Paper

1a) What is a Unifier ?

If a substitution makes two terms equivalent then it is a unifier.

Most General Unifier is the ???

The unifiers in prolog influence backtracking as prolog backtracks to a point where there is a choice for a unifier.

b) What is the occur check, the occur check is a check to make sure that there are no nested unifiers that will never end eg X/s(X) as this will build up forever X/s(s(s(X))) etc

:- tail_of_list(Y, Y)

tail_of_list(T, cons(X, T)) :-

this would unify

{Y/T, Y/cons(X, T) } which in a more general way is { Y/T, Y/cons(X, Y)}

The problem with this is it can build up as so

{Y/T, Y/cons(X, Y)}

{ Y/T, Y/cons(X, cons(X, Y))}

{ Y/T, Y/cons(X, cons(X, cons(X, Y)))}

{ Y/T, Y/cons(X, cons(X, cons(X, cons(X, Y))))}

etc

c)

i) Paul loves people who love music or who do not love logic.

$\forall X$ loves(paul, X) → loves(X, music) v ¬loves(X,logic)

ii) Maggie loves people that love her.

$\forall X$ loves(maggie, X) → loves(X, maggie) iii)Peter loves those people, and only those people who love Maggie.

$\forall X$ loves(peter, X) → loves(X, maggie)

d) (p -> q) -> (p -> ¬q)

¬(¬p v q) v (¬p v ¬q)

(p ^ ¬q) v (¬p v ¬q)

(¬p v ¬q v p) ^ (¬p v ¬q v ¬q)

T ^ (¬p v ¬q) -> Satisfiable

2)

a)

plus(0, X, X)

plus(s(X), Y, s(Z)) <- plus(X, Y, Z)

{plus(0, X, X), ¬plus(s(X), Y, s(Z)) v plus(X, Y, Z)}

prove

plus(s(s(0)), s(s(0)), s(s(s(s(0))))).

{plus(0, X, X), plus(s(X), Y, s(Z)) v ¬plus(X, Y, Z), ¬plus(s(s(0)), s(s(0)), s(s(s(s(0)))))}

resolve 2, 3 { X/s(0), Y/s(s(0)), Z/s(s(s(0))) }

{plus(0, X, X), plus(s(X), Y, s(Z)) v ¬plus(X, Y, Z), ¬plus(s(0), s(s(0)), s(s(s(0)))}

resolve 2, 3 { X/0, Y/s(s(0)), Z/s(s(0)) }

{plus(0, X, X), plus(s(X), Y, s(Z)) v ¬plus(X, Y, Z), ¬plus(0, s(s(0)), s(s(0))}

resolve 1, 3

{plus(0, X, X), plus(s(X), Y, s(Z)) v ¬plus(X, Y, Z), □}

b)

{plus(0, X, X), plus(s(X), Y, s(Z)) v ¬plus(X, Y, Z), ¬plus(s(s(0)), s(0), G1)}

resolve 2, 3 { X/s(0), Y/s(0), Z/s(Z2), G1/s(Z2) }

{plus(0, X, X), plus(s(X), Y, s(Z2)) v ¬plus(X, Y, Z2), ¬plus(s(0), s(0), G2)}

resolve 2,3 { X/0, Y/s(0), Z2/s(Z3), G3/s(Z3) }

{plus(0, X, X), plus(s(X), Y, s(Z3)) v ¬plus(X, Y, Z3), ¬plus(0, s(0), G3)}

resolve 1,3 {X/s(0), Z3/s(0)}

{plus(0, X, X), plus(s(X), Y, s(Z)) v ¬plus(X, Y, Z), □}

G1 / s(Z2)

Z2 / s(Z3)

Z3 / s(0)

G1 / s(s(s(0)))

c)

d)

p(X, X)	p(X, a)	{X/X, X/a}	{X/a}
p(s(0), X, Y)	p(s(X), 0, s(0))	{X/0, X/0, Y/s(0)}	{X/0, Y/s(0)}
r(X,Y,X)	r(Y, s(Y), X)	{X/Y, Y/s(Y) <- FAIL occur check	
r(a, b, X, Y)	r(X, Y, Z, b)	{X/a, Y/b, X/Z, Y/b}	{X/a, Y/b, Z/a}
r(a, b, X, Y)	r(X, Y, Z, Z)	{X/a, Y/b, Z/X, Z/Y}	<- Fails Z cant unify to 2 things
r(X,Y,V)	r(Y, s(Z), Z)	{X/Y, Y/s(Z), V/Z}	{X/s(V), Y/s(V), Z/V}

3)

a grandparent(X,Y) :- parent(X, _T), parent(_T, Y).

b eligible(X, C) :- birthplace(X, C).

eligible(X, C) :- parent(_P, X), birthplace(_P, C).

eligible(X, C) :- grandparent(_P, X), birthplace(_P, C).

c

eligible('Joe Smith', 'England').

???

d check(L, C)

check([], C).

check([P|L], C) :- eligible(P, C), check(L, C).

DOES THIS WORK ????

e not_eligible(X, C).

not_eligible(X, C) :- eligible(X, C), fail.

not_eligible(X, C).

4) ????

2002/2003

1a) What is a computed answer ?

A computed answer is generated by using proof by refutation and generating the most general unifiers at each step of the resolution and by restricting the resolution to the goals in the top level. Most general unifiers are computed at each stage then the mgu between the selected literal and the head clause is computed and then applied to the current goal and body of the clause.

b)

i (p ^ ¬p) -> q	¬(p ^ ¬p) v q	¬p v p v q	= Tautology as ¬p v p
ii (¬q -> ¬p) v (p -> q)	(q v ¬p) v (¬p v q)	q v ¬p	= Satisfiable
iii ((p ^ r) v q) -> (p ^ q ^ r)	¬((p ^ r) v q) v (p ^ q ^ r)	(¬(p ^ r) ^ ¬q) v (p ^ q ^ r)	
((¬p v ¬r) ^ ¬q) v (p ^ q ^ r)	er cant be bothered		

iv same

c)

notel(X, nil).

notel(X, tr(Y, L, R)) :- dif(X, Y), notel(X, L), notel(X, R).

notel(X, nill)

(notel(X, tr(Y, L, R)) v ¬(dif(X,Y) ^ notel(X, L) ^ notel(X, R))

=

$\forall X$ notel(X, nil)

($\forall X \forall Y \forall L \forall R$ notel(X, tr(Y,L,R))) ← (dif(X, Y) ^ notel(X,L) ^ notel(X,R))

d)

i	p(Z, X)	p(X, a)	{Z/X, X/a}	{Z/a, X/a}
ii	p(c, a, X, Y)	p(X, Y, Z, a)	{X/c, Y/a, Z/X, Y/a}	{X/c, Y/a, Z/c}
iii	p(a, b, X, Y)	p(X, Y, V, V)	{X/a, Y/b, V/X, V/Y}	Fails
iv	p(X, Y, Z)	p(Y, s(Y), X)	{X/Y, Y/s(Y)	<- Fails occur check
v	p(s(0), s(0))	p(s(X), R)	{X/0, R/s(0)}	
vi	p(Z, Y, V)	p(Y, s(W), W)	{Z/Y, Y/s(W), V/W}	{Z/s(V), Y/s(V), V/W}

2a Easy!

b

fun(0,0).

fun(s(s(X)), s(R)) <- fun(X, R).

convert into thingy style

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R)}

trying to prove fun(s(s(s(s(0)))), s(s(0)))

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(s(s(s(s(0)))), s(s(0)))}

resolve 2,3 {X/s(s(0), R/s(0)}

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(s(s(0)), s(0))} resolve 2,3 {X/0, R/0}

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(0,0)}

resolve 1,3

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), □}

c

fun(s(s(X)), X)

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R)}

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(s(s(G1)), G1)} {G1/s(R)}

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(s(R), R)} {R/s(T)}

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(s(s(T)), s(T))} {T/0}

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(0,0)}

G1/s(R) => R/s(T) => T/0

G1/s(s(0))

{fun(0,0), fun(s(s(X)), s(R)) v ¬fun(X, R), ¬fun(G1, s(G1))} {G1/s(s(X))}

{fun(0,0), fun(s(s(X2)), s(R)) v ¬fun(X2, R), ¬fun(X, s(s(X))} {X/s(s(X2))}

{fun(0,0), fun(s(s(X3)), s(R)) v ¬fun(X3, R), ¬fun(X2, s(s(s(X2))} {X/s(s(X2))}

Just gets bigger gap never closing

d)

i ∀ X (like(X, cm202) → like(X, logic))

ii ∃ X (likes(X, cm202))

iii ∀ X likes(X, cm202) → (likes(X, cm217) v likes(X, cm215))

iv ∀ C ∀ X likes(X, C) ∀ Y likes(Y, C) → likes(X, Y)

e)

∀ X (fast(X) → (¬slow(X) ^ ¬immobile(X)) ^ fast(X) ← (¬slow(X) ^ ¬immobile(X)))

¬fast(X) v (¬slow(X) ^ ¬immobile(X)) ^ fast(X) v ¬(¬slow(X) ^ ¬immobile(X)))

(¬fast(X) v ¬slow(X)) ^ (¬fast(X) v ¬immobile(X)) ^ fast(X) v (slow(X) v immobile(X)))

(¬fast(X) v ¬slow(X)) ^ (¬fast(X) v ¬immobile(X)) ^ fast(X) v (slow(X) v immobile(X)))

Not sure bout this hmm

(¬fast(X) v immobile(X) ^ (¬fast(X) v slow(X) ^ fast(X) slow(X) v immobile(X)

3. a

conc(A, B, C)

```
conc([], B, B).
conc([A | L] , B, [A | C]) :- conc(L,B,C).
```

b)

conc([a,b], [c,d,e], L).

conc([a,b],[c,d,e],G1).	L/G1	
conc([b],[c,d,e],[G2 | G3]).	G1/[G2|G3]	G2/a
conc([], [c,d,e], [G4 | G5]).	G3/[G4|G5]	G4/b
conc([],[c,d,e], G6).	G5/G6	G6/[c,d,e]

Works back up

G5/[c,d,e]

G3/[b,c,d,e]

G1/[a,b,c,d,e]

L / [a,b,c,d,e]

c) split(X,L,LT,GTE)

```
split(X, [A|L], [A|LT], GTE) :- A<X, split(X,L,LT,GTE).
split(X, [A|L], LT, [A|GTE]) :- A>=X, split(X,L,LT,GTE).
```

2b

```
fun(0,0).
fun(s(s(X)),s(R)) <-fun(X,R).
```

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R)}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), ¬fun(s(s(s(s(0)))), s(s(0))} resolve 2,3 {X/s(s(0), R/s(0)}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), ¬fun(s(s(0)), s(0)} resolve 2,3 {X/s(s(0), R/s(0)}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), ¬fun(0, 0} resolve 2,3 {X/0, R/0}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), □} resolve 1,3

c

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), fun(s(s(G1)), G1)} resolve(2,3) {G1/s(R)}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), fun(s(R), R)} resolve(2,3) {R/s(R2)} {fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), fun(s(R), R)} resolve(2,3) {R/s(R2)}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), fun(s(s(R2)), s(R2)} resolve(2,3) {R2/T}

{fun(0,0), fun(s(s(X), s(R)) v ¬fun(X,R), fun(T,T)} resolve 1,3 { T/0 }

conc([a,b], [c,d,e], L).	
conc([G1 | b], [c,d,e], [G1 | G2]).	{G1/a, L/[G1|G2]}
conc([G3], [c,d,e], [G3 | G4]).	{G3/b, G2/[G3|G4]}
conc([], G5, G4).	{G5/[c,d,e], G4/G5}

L/[G1|G2]

L/[G1|G3|G4]

L/[G1|G3|G5]

L/[a,b,c,d,e]