VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB REPORT on

Artificial Intelligence

(22CS5PCAIN)

Submitted by

SHANA DIYA SUJIT(1BM21CS196)

in partial fulfillment for the award of the degree of BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Nov-2023 to Feb-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled "Artificial Intelligence" carried out by SHANA DIYA SUJIT(1BM21CS196), who is bonafide student of B.M.S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a Artificial Intelligence (22CS5PCAIN) work prescribed for the said degree.

Dr. PALLAVI G.B.

Assistant Professor

Department of CSE

BMSCE, Bengaluru

Dr. Jyothi S Nayak

Professor and Head

Department of CSE

BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	Implement Tic –Tac –Toe Game.	1 - 6
2	Solve 8 puzzle problems.	7 - 10
3	Implement Iterative deepening search algorithm.	11 - 14
4	Implement A* search algorithm.	15 - 19
5	Implement vaccum cleaner agent.	20 - 22
6	Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not.	23 - 24
7	Create a knowledge base using prepositional logic and prove the given query using resolution	25 - 29
8	Implement unification in first order logic	30 - 35
9	Convert a given first order logic statement into Conjunctive Normal Form (CNF).	36 - 37
10	Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.	38 - 42

OBSERVATION BOOK PICTURES:

1. Tic Tac Toe

-	Page
	LABI
15/11/23	TIC TAC TOE
	ales despend of that the special states and the
	in ford random
	toors = 1' for - in range lost
	poor = 1 for - in range (10)
	Ret marro tellor (letter for):
	global brand
	board (pos) = litter
	C- X
	Let space (for)
	Let space Free (for): return board (for) == '
	1 1 9 1/4/18 18 18 18 18 18 18 18 18 18 18 18 18 1
	Let print Pourd (pourd):
	Act print ('11') # print ('1+ board [1] + '1' + board [2] +'1
	wint (+ bard[1] + 1 + bard[2]
	bran (3)
1.	
	1 + [2] broad + 1 + [4] broad + 1) + mind
	privile (161)
	# point (1)
	print ()
	print (1 + poord(+) + 11 + poord(8)+1.
	Loard 18]

() grant quer Jos fort is Niance (to, le) Special from I a do y letter in comments (what == le and into t = le and total == le) is (what == le and total == le and total == le) is (what == le and total == le and total == le) for the on 10 x 12.

for a long = board (:)

board copy = board (:) (boli] == le and bolis == le and bo [] == le) of (1011) = 1 and basic? = 10 and basic? = 10 a (1012) = 10 and basic? = 10 and basic? = 10 as (1013) = 10 and basic? = 10 and basic? = 10 as (bo (1) = 10 and basic? = 10 and basic? = 10) Coly your Charles fi det flagermore (). converesper = 12 in in free Manue if LETRIL HI W dolal books nin = True return every- Rondon Connect Con more - injust ('rebut pue so place 11 5 in SnessHeThores replan = Edges Dem - li to i in fractilet Morce if in me ve = int (more) 1 <= mare 2 = 9; [83 46] 11 space Is Free (move): invertetter ('x', mayo) f edge den: return solo D Ran dom corner Op else : print (" Sonry , occupied") separa none else: Let wood-Randon (le); print (" syme inside range") s ln = len(li) " = randon randrange (In) except Value Error: return lefr] print (" type within range") 14TXIOLIL y according (book d) orbite true:

g ancycle: baged (Agin 2)

g ancycle: bard (Fig. 1)

have d = "T" for - be rought (be) produce (" relaine") while not is Board full (board) if not alleman (board of payorhave () OUTPUT: print ("hory o war) complied proud o in 2 I not a winner (board 'x') Mars - comprove () print ("Tie") ingret Letter ('D' grove) print (comp pare 0 print sound (board) print ("X mon") Drip Boad o in A

2. 8 Puzzle Breadth First Search Algorithm

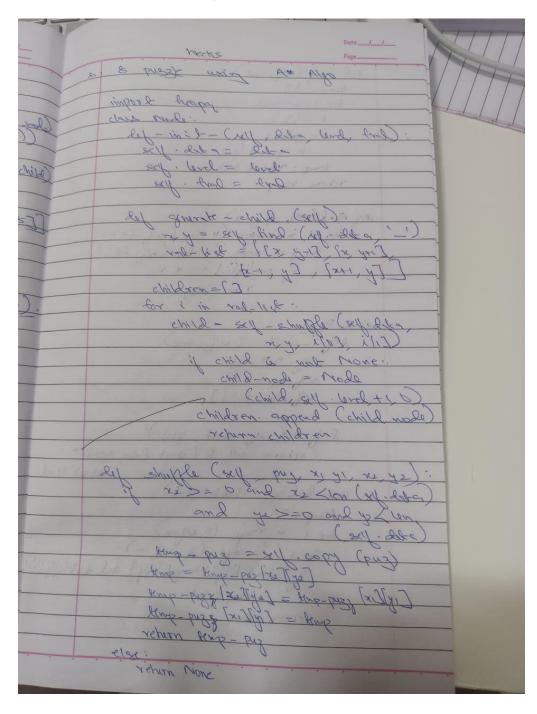
	29/11/23	WEEKS	Date	1
"+ chr (core)	3)	8 Puzzle from win	PS-C	1
cleaned				
	inf	back unably as up	142	
	int	By so bolon of Ira	756	
	into	40 210	7,0	
	Rel	ble (src darret)	N 18/4	
		Junia appen & (src)	22 456 078	
An		que us appen & (src)	in I was	
	TA ALLS			
The Year	00-2	< (way all stills		
	110000	source = que ue fo		
		exp. append (sour	(<u>0</u>)	
Mr (cal-)		buint (some)		
Ironed 1)	it	Source == torpo		
		Doint (' runers') 22 29	
Atr(cot)		return	- 10 10	
ATTIONES	2132		Zach.	1
		bos = 1		-
	0.0	for = fossible-m	ones (2011/10 seb)	+
	1 3		4-127	-
	4	Dog in gram so		4
		I have not "	near and more	
	1 1 7		rat in group.	
	1100	Julie - a	ppend (move)	
	1 =-1	land of	XX	
	det	Bresilde - mores (ste	He Nie :	
		= . slate. indox (0)		
	1 50 %			
		2=()	Park .	-
	1 Stall and	1 10 m lo	(,2]	
		I append (ii	JA mulson	
	11	b not in Is	7 8).	
		& b not in 16	(8)	
2		i abbing	·	
3			The second second	

Pogs	100
('2) Luce in lo 3 6 7 ((2') d)	Ont but .
or (can) to abbrary (5,)	Sac = 1, 9 3 H 2 6 0 1 8]
La Casa	100 - 1123 4 c c 7 80]
(3) 1 1 1 mas in [36 8]:	Les (sac taget)
V Spring (8)	(1,23,45,607,8)
	1123056178
Sus-091 = ()	[1 a a 4 3 4 7 0 8]
for in l:	(028186478)
m) for an append (gen likely is)	[128566478]
Curi to 5- con abbus to love more and	[1234067587]
Actual or way is some or wat	[[087324]]
in he an it was an not	Sycus Sycus
Send (yester was p);	Carry Carry I - American American Carry
THE SEN CHER WY DE	
Kins = Keyk, copy	
1 d a 2 1 1 7 cm /67	fall la
Rup 16+3 thip 16) = Kup 16+3	fait with
1 m = = 'd': Aup (b-1] lup (1) = frup (b) Aup (b-1)	
they (b-1) long (b) = temp 16	The state of the s
Jan 10-12	
The state of the s	
1 m = = 'u'	
[d] aust = [d] with [s. 1)	
Aug (6-3) Aug (6-3)	/ 22 D 10 Lond 1
Carlo Maria James Barrella - 436	
1 m == 2;	
Deux (b+1) Penolo]= Augh]	
Sup (b+1) Aug (b) = Aug (b) / Aug (b+1)	
xwb(px,7	The said self
return leap	The second of th
- Caral Man of the	and wall and a little amount of
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	

3. 8 Puzzle Iterative Deepening Search Algorithm

A fight freshm by Standise leaguing school and to make it make it will able to make it middle able to make a supple freshman by the stand able to make the make the make the stand able to make the make the make the supple fresh and to the stand able to make the mak	PACK IN THE PROPERTY OF THE PR	Dols_11_
Infect thought is the fell infect which stakes in the fell of the stakes the fell of the stakes and the stakes in the fell of the stakes and the stakes are the stakes and the stakes are the stakes and the stakes are	North of Dive Serversing	schury Course &
Language transport as the file of the state of the file of the fil	effort & frule freshmen by warden	man in the mores of
Sort love days hour situal atales): I work a travel The contract of the cont	1	100 100 10 1000
for the last special site of the state of th	miles many as the	del an Color more Work).
There There was a fine of the state of the s	in fixe fundad is for	
There There was a fine of the state of the s	as (as four limit visited stake):	11 was = = 10, 1
Sand a series and series of the series of th	led the less	
The first and the series of th	NOUS TOK	[e-HANG and [AMAGIGANA
The state of the second core of the state of	1 hm & <= 0 :	1/2 max == 2 2 1
real red softe append (are) The red of the married are resided where I depth land = should be applicated to the problem of th	Thirn Folse	
real red whole append (are) mark = fewerld murrhal rec which affects for more in request for more in request return frage return frage for more in request return frage for more in frage return frage for more in frage for more in frage for more in for it to get in the property of the proper		[sex field] good [smild] good
mark = fruit mark (re Waird while) for more in more if ega (more simple sime -) visited while) return trace return trace for more in more return trace return trace for should white visited while) if broader more (white visited while) if broader more (white visited while) if broader more (white visited while) if broad in [0] 27: for more in [0] 28: f	mai ted wholer append (sor)	It made a = 18 is the same
for more in branch : If else more some limbs - I visited whited There is a some later of the stated: The state when the stated is a state when the stat	Company of the second second	[Suns (Henry) Lyng (L) gyl) = Pay (Hank)
if she (more some some -) visited whited resource true resource true resource true resource true resource true resource true or some (she in initial - states): de = 17 if to not in logged: de = 17 if to not in logged: de = 17 if to not in logged: resource true		The second secon
return trace return long fronthe more while visited white if broathe more while visited white if broathe more while visited white if broat in large and visited white for more and	The Mark of Market	The same of the sa
round finder for finance control writed whether: for finance control writed whether: for finance control writed whether: for a start when (1) de = 17 if b most in [0] x? for most in [0] x? f	A The most feeling from F - 1 ways where I	1200 12 00 - Frank 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
for more in 1020 1 for mo	ACHIAN TAKE	. And (plank-1) Ent (plank) - zob say
for more in [0,3]: Sor was a start depth in some (start in record depth): Let a start when (-1) Let a start whe	volume follow	
= state wider (-1) = state = (1)		worm some
= state wider (-) = 1	del brogable-marce (shalf, visited-shake):	It rooks (see single depth)
if b not in [0,2]: if b not in [0,2]: if b not in [0,2]: if b not in [0,3]: if b not in [0,3]: if b not in [0,3]: for man in [0,3]:	b = state vides (-1)	Sux & in ronge (Spoth):
5 not in [0,12]: 1	8=17	CI= ANDONA - SCHA = 1)
	il 10 mod ii 10127:	St. Ser. Accord it visibilities
1 5 not in [238] 1 6 not in [238] 1 7 not in [238] 1 8 not in [
1 - 2 0 0 0 0 0 0 0 0 0	1 + = U - C	
	if b not in 14,785:	rehing talke
	d+-2'.	
	1 6 mole in Q587	Output:
\$ 2000 in (036) \$ 2000 = 1 2 3 4 5 7 6 7 9 \$ 2000 = 1 1 2 3 4 5 7 8 7 9 \$ 2000 = 1 1 2 3 4 5 7 8 7 9 \$ 2000 = 1 1 2 3 4 5 7 8 7 9 \$ 2000 = 1 1 2 3 4 5 7 8 7 9 \$ 2000 = 1 1 2 3 4 5 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7 8 7	0 - 1-1	1 4 5 5 5
Soc man in d:		
Box mary in d:	1 6 mg in (036)	famile - 112, 34 5, 7 6,78)
for man = 1) (of deft (on tempt deft)	2+ = '\+'	C DIAN 1/=1 1/1/2
for more in di		0 1/2 (0 0 1/2 0 1/2)
for mores append Ign (state more)	105-Maris = 1	ig als (The feeling orbits)
for mores append Ign (state more)	tor man in l:	falx.
the state of the s	Les march am en l'an later a somme de	
	the way the way was so	

4. 8 Puzzle A* Search Algorithm



(2) (2) (3) (4) (4) (4) (4) (4) (4) (5) (4) (4	Ret supp (sext most)	lef france Carl stone on a com
from the control of t		stort = made (se a contrata):
from the control of t	for i in root	chart And = coll (() () () ()
the first fill = pollift on fartiff 12. Set of and for sold fill and fartiff 12. Set of and for sold fill and fartiff 12. Set of and for sold fill on fartiff 12. Set of and for sold fill and fartiff 12. Set of and for sold fill on fartiff 12. Set of and for for sold fill on fartiff 12. Set of and for for sold fill on fartiff 12. Set of and for for sold fill on fartiff 12. Set of and for for sold fill on fartiff 12. Set of and for for for sold fill on fartiff 12. Set of and for for for for sold fill on fartiff 12. Set of and for for for for sold fill on fartiff 12. Set of and for	12= 4	cell ones a later and detail
to grand or some of an (and stad) and and (and stad) and an an grand (and stad) and an	is in i with	Drive (11) Hours (18 arg)
Come stage Provided by a single (o con (con) star) Provided by a single (o con) star) Provided by a singl	e approach (i)	white To
Set & (all that bod)	Ans. World	
equilland for a single (a and explain) by in rank (a and explain) chow found chow found		Curs self open 101
So the lead for a so and and seal of the s		
So I is rough (o an (af leta)): There is a real to an (af leta): The after an	0.0641 0 0.	
Chan Party (1 mile (1 mile) 2 mile (1 mile)	at the court fine x	for I in i.
Chan Party (1 mile (1 mile) 2 mile (1 mile)	on I'm rende (o fen (en (en f. the));	print (i end =" ")
Chan Party (1 mile (1 mile) 2 mile (1 mile)	for in renge (a can (my late));	201/4+ (M)
Chen Engle Chen Engle Chen Engle Act - inite (Act engl) Act open = [] Act open = [] Act of chen sort (ang - land or x and x an	if puzilli] == x .	11 541 1 5 00
Class Projec () of the control of t	return in	A 30 4 (can xand shap - 30 0) == 0;
Let b (coll start date date) Set b (coll start date date) Let b (coll start date) Let b (coll		
20 - 1 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2	class Russian	for 1 in agr. generate - chill ():
20 doch = [] 20	Contraction of the contraction o	1. fred = ser. f (1 and - data)
20 doch = [] 20	yel -wit - (ref rix):	Cry and all the second of the
20 doch = [] 20	self. n = size	Toky Shirt
20 dox d = [] 20 dox	SAL OPEN = []	
Set & (set) dear good) = start and = [r'11 2 2 3] [r' 2 2 3] Sect & (set) dear good) = produce (real good) = program (real good) =	S(V, d) = []	del sey open la
** ** ** ** ** ** ** * * * * * * * * *	- Wall	Ed. open sort (Her = land 0
** Charm soft is (short folk good) + Stat what = [[1] & 3] [1] & 3] [1] & 3] **Rect is (say short good): **Rect is (say short good		0 1000 8: 5:40
Ref & (sy dark god): Ref & (sy dark god): Rep = 0 Reg prouse (dark date god): Reg prouse (dark date) Reg prouse (dark dat	tel f leel, dert, good &	Cours = tales
Ret & (set start god): Ret & (set start god): Remp = 0 Response (o set so): Response	return solile (about 00 as 0)+	Star- state = 16,1, 5, 3,1 1, 1, 1, 1, 1, 1, 2, 8,]
Ret & (self start god): Remp = 0 Remp = 0 Response (start start) Response		ma sak [[1 23] [4 2 27 [4 4 1]
Amp = 0 And in range (0, My n): - 46 -	start lend	bus = Birde(a)
for 1 in range (0, M/n): for 1 in range (0, M/n): Acart [1] = graf[1] 1 ml earl[1] 1 2 3 7 8 -	det h (self work god):	18 1813
for in range (0 mg n): - 46	time = 0	Just bearing (yearst plan , but thatis)
1 23 78 - 1-6 return there 1 23 1 25 1 25 1 25	for in very last of	Plas
**************************************	in who (o' ref. is)	-46 122
1 23 78 - 1-6 return there 1 23 1 25 1 25 1 25	tox in range (0 MI m):	758
758 123 456	11 should 61621 - 06162 0	
758 123 456	and with any descripting	123 78_
123	AMP T - 1	1-6
1 23 H 56	return town	758
H 56		
H 56	is all oply	1 2 2
		7-8

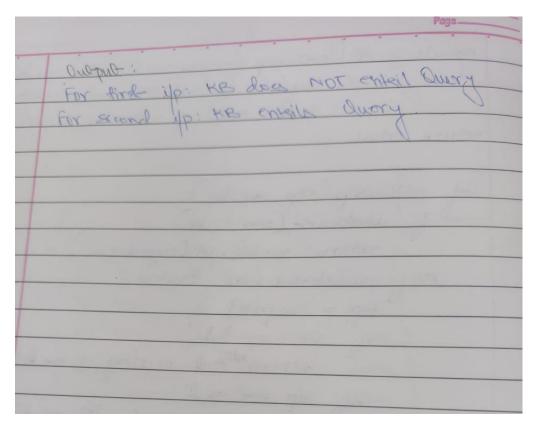
5. Vacuum Cleaner

	Date/_Pags	Dote
		1 = - 2 mo - water y
15/1/1c (ber) 21	Yacuum Claner Agril	Comp & & " Long
	ADCINING O	print (" moning Richet De
		6.0
	yearum marid ():	(xist + = 1
	10 'B' 10 : 10 - 10 - 10 - 10 - 10 - 10 - 10 -	print ("Cod for narry Abort"
(xx)	0 0	+ -9x (002)
	was an input = infut ("Enterloc of vec")	16 = [12] Hole - Bage
4(11)	- a - inch = in out - (" Every wholes of a Dorollion is)	cost + - 1
mi	come injust comp = injust (" enter school of other)	print (" conf. for suck " - str(cost)
1-1-3	(the long start " no tilbred poilered Portine") there	print ("local - 2 been closed")
+ Country a= A F	SIMP - MAST MORENT CONTROL - S. HEAVILLE	
plat (V pecel	most "Tours a location Condition" + ATY (modesty)	e/ge:
	MINTER A WATER	mint ("no asher" strowth)
	gral Alate ('A') = b'	a (corte)
	cost +=1	prive ("& Myerdy dean")
	print (" coop the change A" + cho (worl)	burn Co when he
	print (" A is clamed ")	
	i 1 == cross - have - consts = = 1':	ela:
		mit (" Vacuum is placed in 2")
	print "nowing sight so B"	"y status - ingro- == 1"
	term current afor	print (" & a Surky")
	coop += 1	grad = 1804 (B) = '0"
	print (" case for monity right")	May - Kert (ID):
	ST (cod)	ext+ = 1
	and state (BT=)	must ("cost tox decring + Ar (with)
	000	print (" & has been deaned
	(1)	" = = Con dans = Co fin - who say
	goal - state (6) = 6	A VESTILL - A Disa contrata
	print (16 has been desmed)	brint (H & ally)
		print ("many 1884 to 8
	[[[]]	cost + = 1
	print ("no act on! Ar (cost)	
il	Atom - 198 print (" & atready chan')	print are for muly ter
1	====	Sen Ords
+		god - startia'] = " ""
	print (" A akealy chan")	Deg - Mart H) - a
		wek + 3 = 1
1		Lords and Lot a lat

Date_____ point (" A has been chanted) mint (lost for moning let " str (cot vile (10 Jack Note ['A'] = 60 must ("coop for suck "+ styled print (" A how been chanced print (" SOAL STATE: ") mint (god-state) print (" forthermone mea surrement " + streat vecum- mored () Fater Los & vac. A Enter elabore a other vistom: 1 This I be consider I'm' 'o' '8' i'o' Vacuum is plant in LOCA Lor A is dixty cost for cleaning A: 1 bor a how been cleaned LOC & 'a firty MOVIN RIGHT ROB cool for moving RIBHT 2 Cost for Suck : 3 LOC & has been cleaned to I perfor maner

6. Knowledge Base Entailment

Data	Dors
where the many preparational larger or made	Man we all one can form on the ser in the
The first of market - I was to save I	when well and cap (recover miles)
of most of folke of True of	wind winds (cap in)
ments & Fall & Fall & Fall &	ely involved (or frote): or = cyplo1 if or = not':
months I rue & True 'Y' false I	
(wat Theil = Truc	return end-cap (raph) model) and:
for model in models: madel? = madel? = model? = model? = model? = model? = model?	return (not and exp (apt) model) and
contails = false	return and - cop (april) moles or
return onthis to	tiret - premise (and (or ' p' 'q') (' or ' (not 'r'))
Ary real-record (premove conclu):	result first = condit first Presence first
for Bin Time False	Treath - that: Print ("for Left itp: he entail away)
for 2 in Frac, Folke] for = in [Trac Folke]	print " for 1st it is los our asil"
	D. 12 10 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	brint (" for ong it is we year whe who
	The state of the state of the



7. Knowledge Base Resolution

7	Date/_
	Page
	8 Resolution
	Circle KB weight PL 3 prove dury by Resolution
COL	E import se
	- let main (rules goal):
	rules = rules splik (")
	print (In step le 1 course let De vivodi on le)
	print (1 - 1 30)
	i = \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
	for stop in stops .
	print (+ 1/2) 121 9 stop 1 12 1 8 stop stop]
	Hiteland
THE RESERVE	7 1+= 1= (3/19/29/2
	Let north (trom)
	return +' ~ Sterms' if term[o]! = 9~
	else formali)
	M. N. CHAR = 1
	del rerosa (clause).
	if lon (ckuge) >2:
100	t = split- ferms (dauge)
Ellegris	return & & o[1] \ \ 2 x[0] \'
	return "
100	ENDE WENTER B
10000	ly split- serma (sule):
10	ap = '(~ a [Pars])'
20032	December 2 10 cm (10 kg)
26	derma = re. findul (ap rule)
4	strain terms
	Splite terms (' NPVR')
	(SP , 8)

eg controliction (god chuse):

controliction = [It Squal IV

snepst (god) VSpol IV

reham dage in controlictions or

renere (days) in controlictions Source + - ly Shulp II doc short (2) (2) deg resolve (onles good):

thup = soles copy ()

thup + = [neptr (ghel)]

styps = liell Los Pilland & bordons" + = C" paper. for our in thing estipolitule] = Cover conducion while i < lon (stop) Tehen exps elif ton (gen) - = 1: /gen (0) }) j = and 1/2 n doux = 1) Il double to bag & Bitwall J Arms: = spile for my (long li) V frm 2 6 331). temp append (2) Home 1/0) V

compa (1) = 1" legolard

Shumpli) I and shumpli] I

be soop [-1] I for c in trans1: if negati (O) in throw 2. the = It has him promote to 1 = negate (5) action vietes

the clause in clauses clause not in large and j=9-1/2-1 return steps NEWS GEN BY rules = IRVP RYND main (rules, smalls OUTPUT : Derivolis Claud given RYNP 80 KA HRVP MRVB N NK Liver Resolved A contradiction found when NR is arrival as tone · R'is trut

8. Unification

	re Week & -> longicalian Poss	
		1117
	ed genrente (expression)	
	capression = capression : splice (" (") [1:]	
	capieses in = capieses in [-1]	
	expression = 00.8000 ("(1 <1)()	
	abreason = expression [:-1] where a = expression [:-1] where a = expression [:-1]	
	rehion expression	
	7 2 0 7 8	
	af st Init (ap):	
-	schish edb : 2011 6- (a Ca) (0)	
	Por Ser De Co	
	Ser is constant (char):	
	school cher aupper () and low (chen) == 1	
0	el in a real .	
	ef is variable (char):	
	1==(reph) nod lone (rephis rephis no moder	
Q	A COLOR OF THE COL	
	of replace Attribute (exp old new):	
	attributes = get Attributes (eap)	
	for indea val in enumerate (apprilants);	
	If val == ord:	
	attributes linder To new	
	predicate = get his Dredicate (exp)	
	rehan predicate + "(9 + 4 a join	
16 84	(approximately) 1	
	I'm Di day on 11 House	
l de	and la lava and Arlandi in):	
	apply (rap substitution):	-
	for sut in substitution.	
	200 018 = 810 com	
	exp = replace Attributs (exp old new)	
	rehinn eap	

Popu	Date
and Downs (var Mp)	4 at count 1 mil long (app)
at an Ind (var) -	1 0 Gmb 1 = = 1:
ord Oceans (var (Ap)) Cy tool (con)	report in t
relien Time	
	I all = apply (soil) in (Kent)
sery gating fort (ap)	
at = get Alt land	ran - unity (dell) fails
refur r 20 [a]	if not seen (daily sails)
	when folse
al we R (a)	Colora sell laboration a
by minit (ent)	I enter in puts
The straight of the straight o	
2000 F 2 = 2 + (" + " 10 in (2+ 1: 1 +)"	Oute of t
rehin books	0.12
William miner	[(,21', +(D)) (20/0 L, A)]
(40) 400)	predicates do not match another with a
The mile to the	busylay up nex mency country per muli, or
by ming (ap) cops).	
(mentar	Sub!
	Folk
if is Content (eaps): reharn (eaps, eaps)]	
return ((eapz, eap1))	Sub-
	('s (Contact) ('mother () 'x')
	1 Claus 2
if in lar (sup 2): if check (sup 2, (np))	
I check (exp2, (API)	SW.
	False
if get line ? (exp1) != get line ? (exp2).	The last the second that I have been a second
De al Circulation of the contract of the contr	4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
but are (" no wet ())	The second secon
reform false	West Carling and a second
and the same of th	
A Part of the second second	

9 FOL to CNF

FOL to CNF	Date / /
	white " My" in shapparent .
week?	
FIR DO CHE	M-Hamon Life ("N")
COLE	Must [+1]
ed and (exist)	
method - no tradall (en ming)	white '-I' in statement:
when I'm de min as (matched of my delen)	i - State mail 1 1 1
School III also	S - Clase (Make to C)
	5- (14 (14 mod) = 4, 4(10) 21 -
M mer small ([A-7 a-2]+))	3= 1,5/(12),00
of more (etting): 10-2-2-11-1 Trium re-findal (og estring)	for a in real ments
July se till and the	statement = Alternate xplay (c
	Marine replace (2
string - string replace (~1)	(a) - 1 (1) + (1) + (1) - 2 (2)
Thing - town (AM) wobill	1442 1911 1.27 + 11
String - String. Abbed (~1	4.0
	100
for pred in gh Pred Cothing)	for MS in Ademente
tox pied in gl- Pred (coting). shing - string replace (pred i'r spred)	. 2 lete ment - extrement replace (3 semprens)
3-list (string)	when stop man
leh fol-to-on (tol):	
ely fol-to-con (od):	
distribution of the state of th	statement = stement replan (101/101)
while '- ' in stylement	Shotment - Adment replace ("N(E") 1-E")
i - Natement indea ('-')	
Mon- statement - [statement (6:i)	Il fake hould
1 '>' statement [i' 1:]1	a kac and
128 (1 + Hokmont Pi+1:2+	
AMMAN IN C. J.	
1 + (1:) Swatches + 1 <-	

Page_ Dugeno

10. Forward reasoning

		111
	Neck 10 Page	
250	pure such and proof escanish	
1	and phony washing	///
	CODE:	41
1	limpara re	1
1	de isvariable (2)	
	Chan lower = = (and x inches	
	and a is alpha ()	
		1)
	at = 1/(1 1) 1+1)	1
	ca = 1/(1 x) 2+1),	/
	nother = rc. findall (ex string)	
	return metalies	
	1 Commence of the control of the con	
	Let get Predicates (chin)	
	Let get Predicates (string). Ex = '([q-zn]+) ([12]]+)'	
	26 pro 200 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	retion re-findall (ex string)	
	, , ,	
	chus Fact:	
	1 in thatise	
	1 Split expression	3 10
	Set getreenly (set):	
	remin ed result	
	Let get constants (set):	
	return [Mone if ignor(c) else c for	
	11 222 67	1
	cin sell besome	
	det getvar (self).	
	reveruly if istar (v) else none for	
	Service State Long Mr	
	y in self-paramed	
		TAX DESCRIPTION

	Inalication	Dote 1 Page
)	che Impleading Colle copression	
J	/ Minalize	Onthat
¥	Pet reducte (set forts)	All took :
	Pet valuale (sec)	
		1 Newpor (MO)
	o o sel vos	3. compand (hert)
	1 rd product = text products (constants [v] = text get Constants	S. Supplie (voca)
	constants [v] = tack get Constanting	5. Every (Nono Anerica)
	The state of the s	6 angian (next)
	predicate attributes the expression (1)	T. MAN (MIND) LAND . F
	get Adiales (self The eggregation) [0]	4. Wissile (M)
	art got Attributes (if . rha. expr) (0])	2027 (CW)
ch	K KB.	
	/ julk all of	
	a parties and	
	Par Arricalias.	
	are vacisales.	
	det tell (soff e): if = in e: self implications add (typ (e))	
-	self implications and (Imp (e))	
-		
	elge.	
	sof tack add (fack(e))	
- Fi	- (12 . 200 C - 22(C))	
	0	
	get mary rout c):	
	det query (solf c):	
	Constitution of the second	
	Act Or Carry	
	Set liep (self):	
	1 mint All facts	

Course Outcome

CO1	Apply knowledge of agent architecture, searching and reasoning techniques for different applications.
CO2	Analyse Searching and Inferencing Techniques.
CO3	Design a reasoning system for a given requirement.
CO4	Conduct practical experiments for demonstrating agents, searching and inferencing.

1. Implement Tic -Tac -Toe Game.

```
import math
import copy
X = "X"
O = "O"
EMPTY = None
def initial_state():
  return [[EMPTY, EMPTY, EMPTY],
      [EMPTY, EMPTY, EMPTY],
      [EMPTY, EMPTY, EMPTY]]
def player(board):
  countO = 0
  countX = 0
  for y in [0, 1, 2]:
    for x in board[y]:
      if x == "O":
         countO = countO + 1
      elif x == "X":
         countX = countX + 1
  if countO >= countX:
    return X
  elif countX > countO:
    return O
```

def actions(board):

```
freeboxes = set()
  for i in [0, 1, 2]:
    for j in [0, 1, 2]:
      if board[i][j] == EMPTY:
        freeboxes.add((i, j))
  return freeboxes
def result(board, action):
  i = action[0]
 j = action[1]
  if type(action) == list:
    action = (i, j)
  if action in actions(board):
    if player(board) == X:
      board[i][j] = X
    elif player(board) == O:
      board[i][j] = O
  return board
def winner(board):
  board[1][2] == X \text{ or } board[2][0] == board[2][1] == board[2][2] == X):
    return X
  board[1][2] == O \text{ or } board[2][0] == board[2][1] == board[2][2] == O):
    return O
  for i in [0, 1, 2]:
    s2 = []
    for j in [0, 1, 2]:
```

```
s2.append(board[j][i])
     if (s2[0] == s2[1] == s2[2]):
       return s2[0]
  strikeD = []
  for i in [0, 1, 2]:
     strikeD.append(board[i][i])
  if (strikeD[0] == strikeD[1] == strikeD[2]):
     return strikeD[0]
  if (board[0][2] == board[1][1] == board[2][0]):
     return board[0][2]
  return None
def terminal(board):
  Full = True
  for i in [0, 1, 2]:
     for j in board[i]:
       if j is None:
          Full = False
  if Full:
     return True
  if (winner(board) is not None):
     return True
  return False
def utility(board):
  if (winner(board) == X):
     return 1
  elif winner(board) == 0:
```

```
return -1
  else:
    return 0
def minimax helper(board):
  isMaxTurn = True if player(board) == X else False
  if terminal(board):
    return utility(board)
  scores = []
  for move in actions(board):
    result(board, move)
    scores.append(minimax helper(board))
    board[move[0]][move[1]] = EMPTY
  return max(scores) if isMaxTurn else min(scores)
def minimax(board):
  isMaxTurn = True if player(board) == X else False
  bestMove = None
  if isMaxTurn:
    bestScore = -math.inf
    for move in actions(board):
       result(board, move)
       score = minimax helper(board)
       board[move[0]][move[1]] = EMPTY
       if (score > bestScore):
         bestScore = score
         bestMove = move
```

```
return bestMove
  else:
     bestScore = +math.inf
     for move in actions(board):
       result(board, move)
       score = minimax helper(board)
       board[move[0]][move[1]] = EMPTY
       if (score < bestScore):</pre>
         bestScore = score
         bestMove = move
     return bestMove
def print board(board):
  for row in board:
     print(row)
# Example usage:
game_board = initial_state()
print("Initial Board:")
print_board(game_board)
while not terminal(game board):
  if player(game_board) == X:
     user input = input("\nEnter your move (row, column): ")
    row, col = map(int, user_input.split(','))
    result(game board, (row, col))
  else:
     print("\nAI is making a move...")
```

```
move = minimax(copy.deepcopy(game_board))
result(game_board, move)

print("\nCurrent Board:")
print_board(game_board)

# Determine the winner
if winner(game_board) is not None:
    print(f"\nThe winner is: {winner(game_board)}")
else:
    print("\nIt's a tie!")
```

OUTPUT:

```
Initial Board:
[None, None, None]
[None, None, None]
[None, None, None]
Enter your move (row, column): 1,2
Current Board:
[None, None, None]
[None, None, 'X']
[None, None, None]
AI is making a move...
Current Board:
[None, None, None]
[None, 'O', 'X']
[None, None, None]
Enter your move (row, column): 0,0
Current Board:
['X', None, None]
[None, 'O', 'X']
[None, None, None]
AI is making a move...
Current Board:
['X', '0', None]
[None, '0', 'X']
[None, None, None]
Enter your move (row, column): 2,1
```

```
Current Board:
['X', '0', None]
[None, '0', 'X']
[None, 'X', None]

AI is making a move...

Current Board:
['X', '0', None]
[None, '0', 'X']
['0', 'X', None]

Enter your move (row, column): 1,0

Current Board:
['X', '0', None]
['X', '0', 'X']
['0', 'X', None]

AI is making a move...

Current Board:
['X', '0', '0']
['X', '0', '0']
['X', '0', 'X']
['0', 'X', None]

The winner is: 0
```

2. Solve 8 puzzle problems.

```
def bfs(src,target):
  queue = []
  queue.append(src)
  exp = []
  while len(queue) > 0:
    source = queue.pop(0)
    exp.append(source)
    print(source)
    if source==target:
       print("Success")
       return
    poss_moves_to_do = []
    poss_moves_to_do = possible_moves(source,exp)
    for move in poss_moves_to_do:
       if move not in exp and move not in queue:
         queue.append(move)
def possible moves(state, visited states):
  #index of empty spot
  b = state.index(0)
  #directions array
```

```
d = []
  #Add all the possible directions
  if b not in [0,1,2]:
    d.append('u')
  if b not in [6,7,8]:
    d.append('d')
  if b not in [0,3,6]:
    d.append('l')
  if b not in [2,5,8]:
     d.append('r')
  # If direction is possible then add state to move
  pos moves it can = []
  # for all possible directions find the state if that move is played
  ### Jump to gen function to generate all possible moves in the given directions
  for i in d:
    pos moves it can.append(gen(state,i,b))
  return [move_it_can for move_it_can in pos_moves_it_can if move_it_can not in
visited states]
def gen(state, m, b):
  temp = state.copy()
  if m=='d':
    temp[b+3],temp[b] = temp[b],temp[b+3]
  if m=='u':
```

```
temp[b-3],temp[b] = temp[b],temp[b-3]
  if m=='l':
     temp[b-1],temp[b] = temp[b],temp[b-1]
  if m=='r':
     temp[b+1],temp[b] = temp[b],temp[b+1]
  # return new state with tested move to later check if "src == target"
  return temp
print("Example 1")
src= [2,0,3,1,8,4,7,6,5]
target=[1,2,3,8,0,4,7,6,5]
print("Source: " , src)
print("Goal State: " , target)
bfs(src, target)
print("\nExample 2")
src = [1,2,3,0,4,5,6,7,8]
target = [1,2,3,4,5,0,6,7,8]
print("Source: ", src)
print("Goal State: " , target)
bfs(src, target)
```

OUTPUT:

```
Example 1
Source: [2, 0, 3, 1, 8, 4, 7, 6, 5]
Goal State: [1, 2, 3, 8, 0, 4, 7, 6, 5]
[2, 0, 3, 1, 8, 4, 7, 6, 5]
[2, 8, 3, 1, 0, 4, 7, 6, 5]
[0, 2, 3, 1, 8, 4, 7, 6, 5]
[2, 3, 0, 1, 8, 4, 7, 6, 5]
[2, 8, 3, 1, 6, 4, 7, 0, 5]
[2, 8, 3, 0, 1, 4, 7, 6, 5]
[2, 8, 3, 1, 4, 0, 7, 6, 5]
[1, 2, 3, 0, 8, 4, 7, 6, 5]
[2, 3, 4, 1, 8, 0, 7, 6, 5]
[2, 8, 3, 1, 6, 4, 0, 7, 5]
[2, 8, 3, 1, 6, 4, 7, 5, 0]
[0, 8, 3, 2, 1, 4, 7, 6, 5]
[2, 8, 3, 7, 1, 4, 0, 6, 5]
[2, 8, 0, 1, 4, 3, 7, 6, 5]
[2, 8, 3, 1, 4, 5, 7, 6, 0]
[1, 2, 3, 7, 8, 4, 0, 6, 5]
[1, 2, 3, 8, 0, 4, 7, 6, 5]
Success
Example 2
Source:
         [1, 2, 3, 0, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 4, 5, 0, 6, 7, 8]
[1, 2, 3, 0, 4, 5, 6, 7, 8]
[0, 2, 3, 1, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 4, 5, 0, 7, 8]
[1, 2, 3, 4, 0, 5, 6, 7, 8]
[2, 0, 3, 1, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 4, 5, 7, 0, 8]
[1, 0, 3, 4, 2, 5, 6, 7, 8]
[1, 2, 3, 4, 7, 5, 6, 0, 8]
[1, 2, 3, 4, 5, 0, 6, 7, 8]
Success
```

3. Implement Iterative deepening search algorithm.

```
def iterative_deepening_search(src, target):
  depth limit = 0
  while True:
     result = depth limited search(src, target, depth limit, [])
     if result is not None:
       print("Success")
       return
     depth limit += 1
     if depth limit > 30: # Set a reasonable depth limit to avoid an infinite loop
       print("Solution not found within depth limit.")
       return
def depth limited search(src, target, depth limit, visited states):
  if src == target:
    print state(src)
     return src
  if depth \lim_{t\to 0}:
     return None
  visited_states.append(src)
  poss_moves_to_do = possible_moves(src, visited states)
  for move in poss moves to do:
     if move not in visited states:
       print state(move)
       result = depth limited search(move, target, depth limit - 1, visited states)
       if result is not None:
```

return result

```
return None
def possible_moves(state, visited_states):
  b = state.index(0)
  d = []
  if b not in [0, 1, 2]:
     d.append('u')
  if b not in [6, 7, 8]:
     d.append('d')
  if b not in [0, 3, 6]:
     d.append('l')
  if b not in [2, 5, 8]:
     d.append('r')
  pos_moves_it_can = []
  for i in d:
     pos moves it can.append(gen(state, i, b))
  return [move_it_can for move_it_can in pos_moves_it_can if move_it_can not in
visited states]
def gen(state, m, b):
  temp = state.copy()
  if m == 'd':
     temp[b+3], temp[b] = temp[b], temp[b+3]
  elif m == 'u':
```

```
temp[b - 3], temp[b] = temp[b], temp[b - 3]
elif m == 'l':
    temp[b - 1], temp[b] = temp[b], temp[b - 1]
elif m == 'r':
    temp[b + 1], temp[b] = temp[b], temp[b + 1]

return temp

def print_state(state):
    print(f"{state[0]} {state[1]} {state[2]}\n{state[3]} {state[4]} {state[5]}\n{state[6]} {state[7]} {state[8]}\n")

print("Example 1")

src = [1,2,3,0,4,5,6,7,8]
target = [1,2,3,4,5,0,6,7,8]
print("Source: ", src)
print("Goal State: ", target)
iterative_deepening_search(src, target)
```

```
Example 1
Source: [1, 2, 3, 0, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 4, 5, 0, 6, 7, 8]
0 2 3
1 4 5
6 7 8
1 2 3
6 4 5
0 7 8
1 2 3
4 0 5
6 7 8
0 2 3
1 4 5
6 7 8
2 0 3
1 4 5
6 7 8
1 2 3
6 4 5
0 7 8
1 2 3
6 4 5
7 0 8
1 2 3
4 0 5
6 7 8
```

```
1 0 3
4 2 5
6 7 8

1 2 3
4 7 5
6 0 8

1 2 3
4 5 0
6 7 8

1 2 3
4 5 0
6 7 8

Success
```

4. Implement A* search algorithm.

```
def print_grid(src):
  state = src.copy()
  state[state.index(-1)] = ' '
  print(
     f'''''
{state[0]} {state[1]} {state[2]}
{state[3]} {state[4]} {state[5]}
{state[6]} {state[7]} {state[8]}
  )
def h(state, target):
  #Manhattan distance
  dist = 0
  for i in state:
     d1, d2 = state.index(i), target.index(i)
     x1, y1 = d1 \% 3, d1 // 3
     x2, y2 = d2 \% 3, d2 // 3
     dist += abs(x1-x2) + abs(y1-y2)
  return dist
def astar(src, target):
  states = [src]
  g = 0
  visited_states = set()
  while len(states):
     moves = []
     for state in states:
```

```
visited_states.add(tuple(state))
       print_grid(state)
       if state == target:
          print("Success")
          return
       moves += [move for move in possible moves(state, visited states) if move not in
moves]
     costs = [g + h(move, target) for move in moves]
     states = [moves[i] for i in range(len(moves)) if costs[i] == min(costs)]
     g += 1
  print("Fail")
def possible moves(state, visited states):
  b = state.index(-1)
  d = []
  if 9 > b - 3 >= 0:
     d += 'u'
  if 9 > b + 3 >= 0:
     d += 'd'
  if b not in [2,5,8]:
     d += 'r'
  if b not in [0,3,6]:
     d += '1'
  pos moves = []
  for move in d:
     pos moves.append(gen(state,move,b))
  return [move for move in pos moves if tuple(move) not in visited states]
def gen(state, direction, b):
  temp = state.copy()
  if direction == 'u':
```

```
temp[b-3], temp[b] = temp[b], temp[b-3]
  if direction == 'd':
     temp[b+3], temp[b] = temp[b], temp[b+3]
  if direction == 'r':
     temp[b+1], temp[b] = temp[b], temp[b+1]
  if direction == 'l':
     temp[b-1], temp[b] = temp[b], temp[b-1]
  return temp
#Test 1
print("Example 1")
src = [1,2,3,-1,4,5,6,7,8]
target = [1,2,3,4,5,-1,6,7,8]
print("Source: " , src)
print("Goal State: " , target)
astar(src, target)
# Test 2
print("Example 2")
src = [1,2,3,-1,4,5,6,7,8]
target=[1,2,3,6,4,5,-1,7,8]
print("Source: ", src)
print("Goal State: " , target)
astar(src, target)
# Test 3
print("Example 3")
src = [1,2,3,7,4,5,6,-1,8]
```

```
target=[1,2,3,6,4,5,-1,7,8]
print("Source: ", src)
print("Goal State: ", target)
astar(src, target)
```

```
Example 1
Source: [1, 2, 3, -1, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 4, 5, -1, 6, 7, 8]
1 2 3
 4 5
6 7 8
1 2 3
4 5
6 7 8
1 2 3
4 5
6 7 8
Success
Example 2
Source: [1, 2, 3, -1, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 6, 4, 5, -1, 7, 8]
1 2 3
  4 5
6 7 8
1 2 3
6 4 5
  7 8
Success
```

```
1 2 3
Example 3
                                                                              6 5
Source: [1, 2, 3, 7, 4, 5, 6, -1, 8]
Goal State: [1, 2, 3, 6, 4, 5, -1, 7, 8]
                                                                            4 7 8
1 2 3
                                                                            1 2 3
7 4 5
6 8
                                                                            4 7 8
1 2 3
                                                                            1 2 3
7 4 5
                                                                            6 7 5
  6 8
                                                                            4 8
1 2 3
                                                                            1 2 3
 4 5
                                                                            6 7 5
7 6 8
  2 3
                                                                            1 2 3
1 4 5
                                                                             7 5
7 6 8
                                                                            6 4 8
1 2 3
                                                                            2 3
1 7 5
4 5
7 6 8
                                                                            6 4 8
1 2 3
                                                                            1 2 3
4 6 5
                                                                            7 5
6 4 8
7 8
```

```
7 1 3
4 6 5
  2 8
7 1 3
4 6 5
2 8
7 1 3
4 5
2 6 8
7 1 3
4 6 5
2 8
7 1 3
4 5
2 6 8
7 1 3
2 4 5
  6 8
Fail
```

5. Implement vacuum cleaner agent.

```
def clean(floor, row, col):
  i, j, m, n = row, col, len(floor), len(floor[0])
  goRight = goDown = True
  cleaned = [not any(f) for f in floor]
  while not all(cleaned):
     while any(floor[i]):
       print_floor(floor, i, j)
       if floor[i][j]:
          floor[i][j] = 0
          print_floor(floor, i, j)
       if not any(floor[i]):
          cleaned[i] = True
          break
       if j == n - 1:
          j -= 1
          goRight = False
       elif j == 0:
          j += 1
          goRight = True
       else:
          j += 1 if goRight else -1
     if all(cleaned):
       break
     if i == m - 1:
       i = 1
       goDown = False
     elif i == 0:
       i += 1
```

```
goDown = True
     else:
       i += 1 if goDown else -1
     if cleaned[i]:
       print_floor(floor, i, j)
def print_floor(floor, row, col): # row, col represent the current vacuum cleaner position
  for r in range(len(floor)):
     for c in range(len(floor[r])):
       if r == row and c == col:
          print(f'' > \{floor[r][c]\} < ", end = ")
       else:
          print(f'' \{floor[r][c]\} ", end = ")
     print(end = '\n')
  print(end = '\n')
#Test 1
floor = [[1, 0, 0, 0],
     [0, 1, 0, 1],
     [1, 0, 1, 1]]
print("Room Condition: ")
for row in floor:
  print(row)
print("\n")
clean(floor, 1, 2)
```

```
Room Condition:
                                                         1
                                                              0
                                                                    0
                                                                         0
[1, 0, 0, 0]
                                                         0
                                                              0
                                                                    0
                                                                         0
[0, 1, 0, 1]
                                                        >1<
                                                              0
                                                                    1
                                                                         1
[1, 0, 1, 1]
                                                        1
                                                              0
                                                                   0
                                                                         0
                                                         0
                                                              0
                                                                   0
                                                                         0
                                                        >0<
                                                              0
  1
       0
             0
                   0
  0
        1
            >0<
                   1
                                                              0
                                                                   0
                                                                         0
        0
                                                         0
                                                              0
                                                                    0
                                                                         0
                                                         0
                                                             >0<
                                                                   1
                                                                         1
  1
        0
             0
                   0
        1
             0
                  >1<
  0
                                                         1
                                                              0
                                                                   0
                                                                         0
        0
  1
             1
                   1
                                                         0
                                                              0
                                                                   0
                                                                         0
                                                         0
                                                              0
                                                                  >1<
                                                                         1
  1
        0
             0
                   0
                                                         1
                                                              0
                                                                   0
                                                                         0
  0
        1
             0
                  >0<
  1
        0
             1
                                                         0
                                                              0
                                                                   0
                                                                         0
                                                         0
                                                              0
                                                                  >0<
                                                                         1
  1
       0
             0
                   0
                                                                   0
                                                                         0
                                                         1
                                                              0
        1
                   0
  0
            >0<
                                                         0
                                                              0
                                                                   0
                                                                         0
        0
                   1
  1
             1
                                                         0
                                                              0
                                                                    0
                                                                        >1<
  1
       0
                   0
             0
                                                         1
                                                              0
                                                                    0
                                                                         0
  0
       >1<
             0
                   0
                                                         0
                                                              0
                                                                    0
                                                                         0
       0
  1
             1
                   1
                                                         0
                                                              0
                                                                    0
                                                                        >0<
  1
       0
             0
                   0
                                                         1
                                                              0
                                                                    0
                                                                         0
                   0
                                                         0
                                                              0
                                                                    0
                                                                        >0<
  0
       >0<
             0
                                                         0
                                                              0
                                                                    0
                                                                         0
  1
       0
             1
                   1
                                                         1
                                                              0
                                                                    0
                                                                        >0<
  1
       0
             0
                   0
                                                         0
                                                              0
                                                                    0
                                                                         0
  0
        0
              0
                   0
                                                         0
                                                              0
                                                                    0
                                                                         0
  1
       >0<
             1
                   1
```

```
1
       0
            >0<
                    0
       0
             0
 0
                    0
 0
       0
             0
                    0
 1
      >0<
             0
                    0
 0
       0
             0
                    0
 0
       0
             0
                    0
>1<
       0
             0
                    0
0
       0
             0
                    0
0
       0
             0
                    0
>0<
       0
             0
                    0
0
       0
             0
                    0
 0
       0
             0
                    0
```

6. Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not.

```
def evaluate expression(p, q, r):
  expression result = (p \text{ or } q) and (not r \text{ or } p)
  return expression result
def generate_truth_table():
  print(" p | q | r | Expression (KB) | Query (p^r)")
  print("---|---|---")
  for p in [True, False]:
     for q in [True, False]:
       for r in [True, False]:
          expression result = evaluate expression(p, q, r)
          query result = p and r
          print(f" {p} | {q} | {r} | {expression_result}
                                                        | {query result}")
def query entails knowledge():
  for p in [True, False]:
     for q in [True, False]:
       for r in [True, False]:
          expression result = evaluate expression(p, q, r)
          query result = p and r
          if expression result and not query result:
            return False
  return True
```

```
def main():
    generate_truth_table()

if query_entails_knowledge():
    print("\nQuery entails the knowledge.")

else:
    print("\nQuery does not entail the knowledge.")

if __name__ == "__main__":
    main()
```

```
KB: (p or q) and (not r or p)
            Expression (KB)
                             Query (p^r)
              True True
                                          True
              False
                     True
                                           False
       False
True | False
               False | True
                                            False
 False
       True
               True | False
 False
        True | False | True
 False
        False | True |
                      False
                                             False
 False | False | False | False
Query does not entail the knowledge.
```

7. Create a knowledge base using prepositional logic and prove the given query using resolution

import re

def main(rules, goal): rules = rules.split(' ') steps = resolve(rules, goal) print('\nStep\t|Clause\t|Derivation\t') print('-' * 30) i = 1for step in steps: print(f' {i}.\t| {step}\t| {steps[step]}\t') i += 1def negate(term): return $f' \sim \{\text{term}\}' \text{ if } \text{term}[0] != '\sim' \text{ else } \text{term}[1]$ def reverse(clause): if len(clause) > 2: t = split_terms(clause) return $f'\{t[1]\}v\{t[0]\}'$ return " def split terms(rule): $exp = '(\sim *[PQRS])'$ terms = re.findall(exp, rule) return terms split_terms('~PvR') def contradiction(goal, clause): contradictions = $[f\{goal\}v\{negate(goal)\}', f\{negate(goal)\}v\{goal\}']$ return clause in contradictions or reverse(clause) in contradictions def resolve(rules, goal):

```
temp = rules.copy()
  temp += [negate(goal)]
  steps = dict()
  for rule in temp:
     steps[rule] = 'Given.'
  steps[negate(goal)] = 'Negated conclusion.'
  i = 0
  while i < len(temp):
     n = len(temp)
     j = (i + 1) \% n
     clauses = []
     while i != i:
        terms1 = split_terms(temp[i])
        terms2 = split terms(temp[i])
        for c in terms1:
           if negate(c) in terms2:
             t1 = [t \text{ for } t \text{ in terms } 1 \text{ if } t != c]
             t2 = [t \text{ for } t \text{ in terms 2 if } t != negate(c)]
             gen = t1 + t2
             if len(gen) == 2:
                if gen[0] != negate(gen[1]):
                   clauses += [f'\{gen[0]\}v\{gen[1]\}']
                else:
                   if contradiction(goal,f'{gen[0]}v{gen[1]}'):
                      temp.append(f'\{gen[0]\}v\{gen[1]\}')
                     steps["] = f"Resolved \{temp[i]\} and \{temp[j]\} to \{temp[-1]\}, which is in
turn null. \
                     \nA contradiction is found when {negate(goal)} is assumed as true.
Hence, {goal} is true."
                     return steps
             elif len(gen) == 1:
```

```
clauses += [f'{gen[0]}']
             else:
                if contradiction(goal,f'{terms1[0]}v{terms2[0]}'):
                  temp.append(f'{terms1[0]}v{terms2[0]}')
                  steps["] = f"Resolved \{temp[i]\} and \{temp[j]\} to \{temp[-1]\}, which is in
turn null. \
                  \nA contradiction is found when {negate(goal)} is assumed as true. Hence,
{goal} is true."
                  return steps
        for clause in clauses:
          if clause not in temp and clause != reverse(clause) and reverse(clause) not in temp:
             temp.append(clause)
             steps[clause] = f'Resolved from {temp[i]} and {temp[i]}.'
       j = (j + 1) \% n
     i += 1
  return steps
rules = 'Rv\sim P Rv\sim Q \sim RvP \sim RvQ' \#(P^{\wedge}Q) \le R : (Rv\sim P)v(Rv\sim Q)^{\wedge}(\sim RvP)^{\wedge}(\sim RvQ)
goal = 'R'
print('Rules: ',rules)
print("Goal: ",goal)
main(rules, goal)
rules = PvQ \sim PvR \sim QvR' \#P=vQ, P=>Q : \sim PvQ, Q=>R, \sim QvR
goal = 'R'
print('Rules: ',rules)
print("Goal: ",goal)
main(rules, goal)
rules = 'PvQ PvR \simPvR RvS Rv\simQ \simSv\simQ' # (P=>Q)=>Q, (P=>P)=>R, (R=>S)=>\sim(S=>Q)
goal = 'R'
print('Rules: ',rules)
```

```
print("Goal: ",goal)
main(rules, goal)
```

```
Example 1
Rules: Rv~P Rv~Q ~RvP ~RvQ
Goal: R
        |Clause |Derivation
Step
         Rv~P
                  Given.
1.
         Rv~Q
                  Given.
3.
         ~RvP
                  Given.
4.
         ~RvQ
                  Given.
                  Negated conclusion.
5.
         ~R
                 Resolved Rv~P and ~RvP to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
Example 2
Rules: PvQ ~PvR ~QvR
Goal: R
        |Clause |Derivation
Step
1.
         PvQ
                 Given.
                  Given.
         ~PvR
                  Given.
3.
         ~QvR
         ~R
                  Negated conclusion.
4.
                 Resolved from PvQ and ~PvR.
5.
         QvR
         PvR
                 Resolved from PvQ and ~QvR.
6.
         ~P
                  Resolved from ~PvR and ~R.
 7.
                  Resolved from ~QvR and ~R.
8.
         ~Q
                  Resolved from ~R and QvR.
9.
         Q
         Р
                 Resolved from ~R and PvR.
10.
11.
         R
                 Resolved from QvR and ~Q.
                 Resolved R and ~R to Rv~R, which is in turn null.
12.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

```
Example 3
Rules: PvQ PvR ~PvR RvS Rv~Q ~Sv~Q
Goal: R
Step
       |Clause |Derivation
 1.
          PvQ
                  Given.
          PvR
                  Given.
 2.
                  Given.
         ~PvR
 3.
         RvS
                  Given.
 4.
                  Given.
 5.
          Rv~Q
 6.
          ~Sv~Q
                  Given.
         ~R
                  Negated conclusion.
 7.
 8.
         QvR
                  Resolved from PvQ and ~PvR.
                  Resolved from PvQ and ~Sv~Q.
 9.
         Pv~S
        l P
 10.
                  Resolved from PvR and ~R.
 11.
          ~P
                  Resolved from ~PvR and ~R.
 12.
         Rv~S
                  Resolved from ~PvR and Pv~S.
 13.
         R
                  Resolved from ~PvR and P.
                  Resolved from RvS and ~R.
 14.
          S
                  Resolved from Rv~Q and ~R.
 15.
          ~Q
 16.
                  Resolved from ~R and QvR.
          Q
                  Resolved from ~R and Rv~S.
 17.
          ~S
18.
                 Resolved ~R and R to ~RvR, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

8. Implement unification in first order logic

```
import re
def getAttributes(expression):
  expression = expression.split("(")[1:]
  expression = "(".join(expression)
  expression = expression[:-1]
  expression = re.split("(?<!\(.),(?!.\))", expression)
  return expression
def getInitialPredicate(expression):
  return expression.split("(")[0]
def isConstant(char):
  return char.isupper() and len(char) == 1
def isVariable(char):
  return char.islower() and len(char) == 1
def replaceAttributes(exp, old, new):
  attributes = getAttributes(exp)
  for index, val in enumerate(attributes):
     if val == old:
       attributes[index] = new
  predicate = getInitialPredicate(exp)
  return predicate + "(" + ",".join(attributes) + ")"
def apply(exp, substitutions):
  for substitution in substitutions:
```

```
new, old = substitution
     exp = replaceAttributes(exp, old, new)
  return exp
def checkOccurs(var, exp):
  if exp.find(var) == -1:
     return False
  return True
def getFirstPart(expression):
  attributes = getAttributes(expression)
  return attributes[0]
def getRemainingPart(expression):
  predicate = getInitialPredicate(expression)
  attributes = getAttributes(expression)
  newExpression = predicate + "(" + ",".join(attributes[1:]) + ")"
  return newExpression
def unify(exp1, exp2):
  if exp1 == exp2:
     return []
  if isConstant(exp1) and isConstant(exp2):
     if exp1 != exp2:
       return False
  if isConstant(exp1):
```

```
return [(exp1, exp2)]
if isConstant(exp2):
  return [(exp2, exp1)]
if is Variable(exp1):
  if checkOccurs(exp1, exp2):
     return False
  else:
     return [(exp2, exp1)]
if is Variable(exp2):
  if checkOccurs(exp2, exp1):
     return False
  else:
     return [(exp1, exp2)]
if getInitialPredicate(exp1) != getInitialPredicate(exp2):
  print("Predicates do not match. Cannot be unified")
  return False
attributeCount1 = len(getAttributes(exp1))
attributeCount2 = len(getAttributes(exp2))
if attributeCount1 != attributeCount2:
  return False
head1 = getFirstPart(exp1)
head2 = getFirstPart(exp2)
initialSubstitution = unify(head1, head2)
if not initial Substitution:
```

```
return False
  if attributeCount1 == 1:
     return initialSubstitution
  tail1 = getRemainingPart(exp1)
  tail2 = getRemainingPart(exp2)
  if initialSubstitution != []:
     tail1 = apply(tail1, initialSubstitution)
     tail2 = apply(tail2, initialSubstitution)
  remainingSubstitution = unify(tail1, tail2)
  if not remaining Substitution:
     return False
  initialSubstitution.extend(remainingSubstitution)
  return initialSubstitution
print("\nExample 1")
exp1 = "knows(f(x),y)"
exp2 = "knows(J,John)"
print("Expression 1: ",exp1)
print("Expression 2: ",exp2)
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
print("\nExample 2")
exp1 = "knows(John,x)"
```

```
exp2 = "knows(y,mother(y))"
print("Expression 1: ",exp1)
print("Expression 2: ",exp2)

substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)

print("\nExample 3")
exp1 = "Student(x)"
exp2 = "Teacher(Rose)"
print("Expression 1: ",exp1)
print("Expression 2: ",exp2)

substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```

```
Example 1
Expression 1: knows(f(x),y)
Expression 2: knows(J,John)
Substitutions:
[('J', 'f(x)'), ('John', 'y')]

Example 2
Expression 1: knows(John,x)
Expression 2: knows(y,mother(y))
Substitutions:
[('John', 'y'), ('mother(y)', 'x')]

Example 3
Expression 1: Student(x)
Expression 2: Teacher(Rose)
Predicates do not match. Cannot be unified
Substitutions:
False
```

9. Convert a given first order logic statement into Conjunctive Normal Form (CNF).

```
def getAttributes(string):
  expr = ' ([^{\wedge})] + )'
  matches = re.findall(expr, string)
  return [m for m in str(matches) if m.isalpha()]
def getPredicates(string):
  expr = '[a-z\sim]+\backslash([A-Za-z,]+\backslash)'
  return re.findall(expr, string)
def Skolemization(statement):
  SKOLEM CONSTANTS = [f(chr(c))') for c in range(ord('A'), ord('Z')+1)]
  matches = re.findall('[\exists].', statement)
  for match in matches[::-1]:
     statement = statement.replace(match, ")
     for predicate in getPredicates(statement):
        attributes = getAttributes(predicate)
        if ".join(attributes).islower():
          statement = statement.replace(match[1],SKOLEM CONSTANTS.pop(0))
  return statement
import re
def fol to cnf(fol):
  statement = fol.replace("=>", "-")
  expr = ' ([ ^]] + ) '
  statements = re.findall(expr, statement)
  for i, s in enumerate(statements):
     if '[' in s and ']' not in s:
        statements[i] += ']'
```

```
for s in statements:

statement = statement.replace(s, fol_to_cnf(s))

while '-' in statement:

i = statement.index('-')

br = statement.index('['']) if '[' in statement else 0

new_statement = '-' + statement[br:i] + '|' + statement[i+1:]

statement = statement[:br] + new_statement if br > 0 else new_statement

return Skolemization(statement)

print(fol_to_cnf("bird(x)=>~fly(x)"))

print(fol_to_cnf("dx[bird(x)=>~fly(x)]"))

print(Skolemization(fol_to_cnf("animal(y)<=>loves(x,y)")))

print(Skolemization(fol_to_cnf("∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]")))

print(fol_to_cnf("[american(x)&weapon(y)&sells(x,y,z)&hostile(z)]=>criminal(x)"))
```

```
Example 1
FOL: bird(x)=>~fly(x)
CNF: ~bird(x)|~fly(x)

Example 2
FOL: ∃x[bird(x)=>~fly(x)]
CNF: [~bird(A)|~fly(A)]

Example 3
FOL: animal(y)<=>loves(x,y)
CNF: ~animal(y)<|loves(x,y)

Example 4
FOL: ∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]
CNF: ∀x~[∀y[~animal(y)|loves(x,y)]]|[[loves(A,x)]]

Example 5
FOL: [american(x)&weapon(y)&sells(x,y,z)&hostile(z)]=>criminal(x)
CNF: ~[american(x)&weapon(y)&sells(x,y,z)&hostile(z)]|criminal(x)
```

10. Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

```
import re
def isVariable(x):
  return len(x) == 1 and x.islower() and x.isalpha()
def getAttributes(string):
  expr = ' ([^{\wedge})] + )'
  matches = re.findall(expr, string)
  return matches
def getPredicates(string):
  expr = '([a-z\sim]+)\backslash([^{\wedge}\&|]+\backslash)'
  return re.findall(expr, string)
class Fact:
  def __init__(self, expression):
     self.expression = expression
     predicate, params = self.splitExpression(expression)
     self.predicate = predicate
     self.params = params
     self.result = any(self.getConstants())
  def splitExpression(self, expression):
     predicate = getPredicates(expression)[0]
     params = getAttributes(expression)[0].strip('()').split(',')
     return [predicate, params]
  def getResult(self):
```

```
return self.result
  def getConstants(self):
     return [None if isVariable(c) else c for c in self.params]
  def getVariables(self):
     return [v if isVariable(v) else None for v in self.params]
  def substitute(self, constants):
     c = constants.copy()
     f = f''\{self.predicate\}(\{','.join([constants.pop(0) if isVariable(p) else p for p in \})\}
self.params])})"
     return Fact(f)
class Implication:
  def init (self, expression):
     self.expression = expression
     1 = expression.split('=>')
     self.lhs = [Fact(f) for f in 1[0].split('&')]
     self.rhs = Fact(1[1])
  def evaluate(self, facts):
     constants = \{\}
     new lhs = []
     for fact in facts:
        for val in self.lhs:
          if val.predicate == fact.predicate:
             for i, v in enumerate(val.getVariables()):
               if v:
                  constants[v] = fact.getConstants()[i]
             new_lhs.append(fact)
```

```
predicate, attributes = getPredicates(self.rhs.expression)[0],
str(getAttributes(self.rhs.expression)[0])
     for key in constants:
        if constants[key]:
          attributes = attributes.replace(key, constants[key])
     expr = f'{predicate} {attributes}'
     return Fact(expr) if len(new lhs) and all([f.getResult() for f in new lhs]) else None
class KB:
  def __init__(self):
     self.facts = set()
     self.implications = set()
  def tell(self, e):
     if '=>' in e:
        self.implications.add(Implication(e))
     else:
        self.facts.add(Fact(e))
     for i in self.implications:
        res = i.evaluate(self.facts)
        if res:
          self.facts.add(res)
  def query(self, e):
     facts = set([f.expression for f in self.facts])
     i = 1
     print(f'Querying {e}:')
     for f in facts:
        if Fact(f).predicate == Fact(e).predicate:
          print(f \setminus t\{i\}, \{f\}')
          i += 1
```

```
def display(self):
     print("All facts: ")
     for i, f in enumerate(set([f.expression for f in self.facts])):
       print(f'\setminus \{i+1\}, \{f\}')
kb = KB()
kb.tell('missile(x)=>weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x,America)=>hostile(x)')
kb.tell('american(West)')
kb.tell('enemy(Nono,America)')
kb.tell('owns(Nono,M1)')
kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
kb.query('criminal(x)')
kb.display()
kb = KB()
kb_tiell('king(x)\&greedy(x)=>evil(x)')
kb_.tell('king(John)')
kb_.tell('greedy(John)')
kb_.tell('king(Richard)')
kb .query('evil(x)')
```

```
Example 1
Querying criminal(x):
        1. criminal(West)
All facts:
        1. american(West)
        2. enemy(Nono,America)
        3. hostile(Nono)
        4. sells(West,M1,Nono)
        5. owns(Nono,M1)
        6. missile(M1)
        7. weapon(M1)
        8. criminal(West)
Example 2
Querying evil(x):
        1. evil(John)
```