Program7: Implement unification in first order logic

lgorithm:	
Sport of the state	C box
First Order Logic:	Pseudocode:
1 John to a human	Function Translate to sol (sentence):
- human (John)	Entere = surface.strip().lower()
	il "is both" in sentence and "and" in sentence:
2) Army human is a world -	return backelor_ and_ married (scortnee)
Deveny human is a mortal (n)	(prof) trasport " Forth trade &
-	if "is the mother of in sentence:
3 John loves Mary	The state of (scalesce)
-> loves (John, Many)	return mother of (sontence)
wes (sold), many	i) "are both students" in gentence:
A There is conveniently by	return both shidents (continue)
There is someone who love Mary.	return both students (tollow
- I n(love (n, May))	with the second
2 M 4	If "if" in centence and "then" in sentence:
All dogs are animals	return if then (sentence)
y a (dogs(n) → arimal (n))	the barrier of the soulest of
	if "there is a person" in centence:
Done done are brown	return knows-everyone (sentence)
3 x (dog (n) 1 brown (n)).	
2 6 9	if "taller than themselvee" in sentence
100	return nobody taller (sentence)
. There is no person who is both a backelor -	
and married:	return "Translation not available"
∀x(married (n) -> n bachelor(n))	
N 1 1/	del mother of (centence): match re match (r [a-ZA-Z]+) is
	match = re, match (r (a-2A-2)+) is
1 1 1 1 1 1 1 1	the mother of ([a·z A·z)+), sentone)
7 17 Y 3 -	il magtab :
	subject = match. group() Olj = match. group(2) return y "Mohurg (F. subject 3, fobj 2) return "Invalid sentence".
	obj a match. group(2)
	return of Mothers (7 subject 5, 706, 5)
	Origination of expression:
	The same of the sa
	def unity (expr1, expr2, subs):
	def unity (expr1, expr2, anbs):
	ZWDS - 1 (
	if expr 1 == expr 2
	rehim enbe
Dule Page	ij is-var(expr1): return unity var(expr1, expr2, su li s. var(expr2): return unity var(expr2, expr1, sut
4-3	return unity - var (exprt, exprt, su
	ij is var (expr 2):
	return unity var (expr2, expr1, sub
Output:	is comp (expr1) and ic comp (expr2)
	if expritali= expr2[a] or len (expr1 != len (expr2):
Many is the mother of John , Mother Of (Many, John)	1= len (expr2):
> Mether Of (Mary, John)	raise error
and the second of the second o	return unity lists (exp. 1, expr 2)
1	unity (expresto), expresto), sub
D John and Many are both strokents.	raise error
-> student (John) ^ student (Havy)	- Sini world
representation of the standard and the standard	de unity var (var, expr, subs):
⇒ If it raining the ground is wet. ⇒ Paining → wet (ground)	def unity var (var, expr., cubs):
> Raining > wet (ground)	return unity (subs (var), expr, sub
TO THE REPORT OF THE PROPERTY OF THE PARTY O	elif becur check (var, expr, cubs):
(4) Nobody is faller than themedian	raise error
1) Nobody is feller than themselves - Vx Taller (2,2).	elici
- V1 1000 (1/x).	subs(ear) = expr
I The sentence and their to become	return subs
(1) Oras mult (1) noder	
	de unity - list (11,12, subs)
Tunker in Comme of mile	for expl, expr2 in 21p (L1, L2),
Canton Vangence - Swans weller	Sub-unity (exprt, expr2, Sub-
V	return subs

	K see
	del is-var(term)
	def is - var (term) vetum is instance (term, str) and term is lslower.
	1
	de 12_comp(term): return 16_instance (term, hist, tuple)). (un(term)>0.
	100000
	def occur-check (var, expr, subs).
0.4	yer == expr:
- PS	
	return any (occur-chect (var. subs
	ellif expr in subs
2.	return occurs check (wood var & m/2 (exce
	return false. Subs
	The language was been a fining 19
	Output: molecular liver married
	A Strain and the Market Strain Commencer of the Commencer
	Expression $f(x,g(b))$ & $f(a,g(b))$ Expert: ('f'', 'x', ('g', 'y')) Substitution: ('x': 'a', 'y': b'?
	(') '' ('a' ('a' b'))
	substitution: ['x': a', 'y': b'3.
	Commence of the property
1.	St. Law 1 Law 1 plans Less Sal
	1970 1990
	Krie 1
5	Sam eats this and is still alive
FOL /ON	F: Consumes (Sam, idli) A - Harmed By (Sam, idli)
0.	gill and everything can note
FOL:	y x (consumes (som, x) → consumes (Bill_x)
INF:	(Billy)
041	-consumes (sam, x) V consumes (Bill, x)
	Good: Ravi lite Idli; Enjoys (Ravi, idli)
	The Proof tree:
	Enjoy (Ravi, tali)
	Food (Idi)
	Food (Pizza) consume (sam, Idli) - Harmed By (son)
	Lange Conference
	to have a second and provide as
	50
1	Limit by
	190. 900 800
	1 1813 1316 11 1406

	EINSEMACE (3) Date
	0000
	Prove using forward chaining: technique
	Facti:
	Raw enjoy a wide variety of foods:
	on Food (>1) → Enjoys (Ravi, 2)
CNF	- Food (2) V Enjoys (Ravi, 2)
ે રે.	Bananas are food.
POL	Food (Banana)
CN	r. Food (Banana)
-3.	Pizza is food
FOL	CNF = Food (Pizza)
4	a hand is anothing that anyone consumes
1.	A food is anything that anyone consumes and len't harned by.
	x+x(3y (consumes (y, x) ∧ [haranedby (y, x))
	(Or)
CNF:	- 3y (consumed (y, 2) 1 t Harmed by (y, x)) V Food (x)
	Subst Start y with c(x)
	- consumed (C(2), x) V Harmed By (CGy), x) V tood (x)

```
Code:
def unify terms(term a, term b, subs=None):
  if subs is None:
    subs = \{\}
  if term a == term b:
    return subs
  if is variable(term a):
    return unify with var(term a, term b, subs)
  if is variable(term b):
    return unify with var(term b, term a, subs)
  if is compound(term a) and is compound(term b):
    if term a[0] := term b[0] or len(term a[1]) := len(term b[1]):
       return None
    for subterm a, subterm b in zip(term a[1], term b[1]):
       subs = unify terms(subterm a, subterm b, subs)
       if subs is None:
         return None
    return subs
  if isinstance(term a, list) and isinstance(term b, list):
    if len(term a) != len(term b):
       return None
    for element a, element b in zip(term a, term b):
       subs = unify terms(element a, element b, subs)
       if subs is None:
         return None
    return subs
  return None
def unify with var(var, expr, subs):
  if var in subs:
    return unify terms(subs[var], expr, subs)
  if expr in subs:
    return unify terms(var, subs[expr], subs)
  if occurs check(var, expr, subs):
    return None # Cyclic substitution check failed
  subs[var] = expr
  return subs
def occurs check(var, expr, subs):
  if var == expr:
```

```
return True
  if is compound(expr):
    return any(occurs check(var, arg, subs) for arg in expr[1])
  if isinstance(expr, list):
    return any(occurs check(var, item, subs) for item in expr)
  if expr in subs:
    return occurs check(var, subs[expr], subs)
  return False
def is variable(item):
  return isinstance(item, str) and item.startswith('?')
def is compound(item):
  return isinstance(item, tuple) and len(item) == 2 and isinstance(item[1], list)
if name == " main ":
  print("Enter expressions in the following format:")
  print("Compound terms: ('f', ['a', 'b'])")
  print("Variables: '?x', '?y'")
  print("Lists: ['a', 'b']")
  print("Constants: 'a', 'b', etc.\n")
  term 1 = \text{eval(input("Enter the first expression } (\Psi_1): "))}
  term 2 = \text{eval(input("Enter the second expression } (\Psi_2): "))}
  substitution result = unify terms(term 1, term 2)
  if substitution result is None:
    print("Unification failed!")
  else:
    print("Unification successful!")
    print("Substitution Set:", substitution result)
Output Snapshot:
Enter expressions in the following format:
Compound terms: ('f', ['a', 'b'])
Variables: '?x', '?y'
Lists: ['a', 'b']
Constants: 'a', 'b', etc.
Enter the first expression (Ψ1): ('Studies',['Abubakar','?x'])
Enter the second expression (\Pu_2): ('Studies',['?y','AI'])
Unification successful!
Substitution Set: {'?y': 'Abubakar', '?x': 'AI'}
```