Progressive Education Society's
Modern College of Engineering, Shivajinagar, Pune-05.

Department of Computer Engineering

Course File

Name of the Course: Lab Practice II

Academic Year: 2024-2025 Term: I

Course: Artificial Intelligence
Course Code: CSE10502

Class: FYMTECH **Division: A/ B**

PES's Modern College of Engineering
(An Autonomous Institute Affiliated to Savitribai Phule Pune University)
Level 6.5: First Year M. Tech. (2024 Pattern)

Course Code: CSE10502	**Course Name: Laboratory Practice II**
Semester: I	

Teaching Scheme	**Credit**	**Examination Scheme**
Practical: 04 Hrs./ week	**Practical: 02**	**Term work (TW): 50 Marks**

Course Objectives:

- To inculcate different tools and technologies to solve real life problems.
- To understand applications development using techniques of artificial intelligence, data mining and network computing.
- To understand the fundamental concepts and objectives of research.
- To develop the skills necessary for conducting effective literature reviews and synthesizing information ethically.
- To demonstrate proficiency in developing research proposals and understand the principles of research design.

Course Outcomes:

On completion of the course, the student will be able to,

CO506.1: Use different tools and technologies to solve real life problems.

CO506.2: Develop applications using techniques of artificial intelligence, data mining and network computing.

CO506.3: Demonstrate comprehension of research proposal components.

CO506.4: Develop proficiency in summarizing technical papers.

Guidelines for Laboratory Conduction

A minimum of six experiments should be performed under Lab Practice – II i.e. 3 from MEC-I and 3 from RM subjects. A list of experiments that may be performed under various subjects of semester - I is given below as a guideline.

Sr. No.	**List of Laboratory Assignments**	**CO Mapping**
	Elective -I- Artificial Intelligence (CSE10503A)	
1.	Implement A star Algorithm for any game search problem.	**CO506.1**
2.	Implement any one of the following expert systems in prolog. a) Medical Diagnosis b) Financial Investment Advisor	**CO506.1**
3.	Define the operators for controlling domestic robot; use these operators to plan an activity to be executed by the robot. For example, transferring two/three objects one over the other from one place to another. Prepare a plan and implement the solution in prolog.	**CO506.2**
4.	Implementation of Unification algorithm.	**CO506.2**
	Elective -I- Data Mining & Business Intelligence (CSE10503B)	
1.	Consider a suitable dataset. For clustering of data instances in different groups, apply different clustering algorithms. Visualize the clusters using suitable tool.	**CO506.1**
2.	Implement the Apriori and FPGrowth algorithm. Build your own association task. Design the task for generating association rules based on minsup and minconf. Compare and analyse the performance of Apriori and FPGrowth	**CO506.1**

	algorithms.	
3.	Implement an application for the share marketing sector which will help customers to suggest whether to buy or sell the shares for a particular company/organization. Apply classification algorithm to Share purchase dataset using any suitable analytical tool such as KNIME, WEKA, R	**CO506.2**
4.	A supermarket has number of items for sale. Build a required Database to develop an application using BI tool for considering one aspect of growth to the business Such as organization of products based on demand and patterns use R Programming or other equivalent latest tools.	**CO506.2**
Elective -I- Network Computing (CSE10503C)		
1.	Create a network design using a specified tool and simulate network traffic. Tools to be used: Cisco Packet Tracer, NetSim, or GNS3.	**CO506.1**
2.	Set up a network that uses feedback mechanisms (e.g., TCP congestion control) to manage QoS. Tools to be used: Wireshark, netcat.	**CO506.1**
3.	Capture and analyse IP packets using a network analyser tool. Tools to be used: Wireshark.	**CO506.2**
4.	Set up a network to stream video and audio, implementing protocols such as RTP (Real-time Transport Protocol) and RTSP (Real-Time Streaming Protocol). Tools to be used: VLC Media Player, Wireshark, Cisco Packet Tracer or GNS3.	**CO506.2**
Research Methodology (RMT06504)		
1.	Study of various research proposals.	**CO506.3**
2.	Summarize technical papers.	**CO506.4**
3.	Design and analyse a sample survey.	**CO506.4**
4.	Analyse data and testing of hypotheses to derive meaningful conclusions.	**CO506.4**
5.	Explore report writing process using Latex.	**CO506.3**
6.	Elaborating Intellectual Property process	**CO506.4**

Learning Resources

Text Books:

1. Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye, "Probability and Statistics for Engineers and Scientists", 9th Edition Prentice Hall, ISBN: 9781292161365.
2. Han, Jiawei Kamber, Micheline Pei and Jian, "Data Mining: Concepts and Techniques", Elsevier Publishers, ISBN:9780123814791, 9780123814807.
3. Natalia Olifer, Victor Olifer, "Computer Networks, Principles, Technologies and Protocols for network design", Wiley India ISBN: - 978-0470869826.
4. C. R. Kothari and Gaurav Garg, "Research Methodology Methods and Techniques", 4th Edition, New Age International Publishers, 2019
5. Ranjit Kumar, "Research Methodology": A Step-by-Step Guide for Beginners, 3rd Edition, Sage Publications, 2011
6. T. Ramappa, "Intellectual Property Rights under WTO: Tasks before India", Wheeler Publications, 2010.
7. Debora J. Halbert, "Resisting Intellectual Property", Routledge, Taylor & Francis Group, 2005

MOOC Courses (Web Links):

1. NPTEL, IIT Patna, Dr. Rajiv Misra, "Big Data Computing", https://nptel.ac.in/courses/106104189
2. Introduction to Research, By Prof. Prathap Haridoss, IIT Madras, https://onlinecourses.nptel.ac.in/noc21_ge03/preview

1. Implement A star Algorithm for any game search problem

Title

Implementation of A* Algorithm for a Game Search Problem

Aim

To implement and analyze the A* algorithm for solving a game search problem by finding the optimal path to reach the goal state in a grid-based environment.

Objective

1. Understand the working of the A* algorithm in informed search.
2. Apply heuristic functions to improve search efficiency.
3. Evaluate the algorithm's ability to find the shortest path in a game scenario.

Algorithm

*A Algorithm**

1. **Initialize**:
    - Define the start node and goal node.
    - Initialize the open list with the start node and an empty closed list.
2. **Loop Until Goal is Reached or Open List is Empty**:
   a. Select the node from the open list with the lowest , where:
    - : Cost from the start node to the current node.
    - Heuristic estimate of the cost from the current node to the goal.

b. If the current node is the goal node:

- Terminate the search and reconstruct the path.

c. Otherwise:

- Remove the current node from the open list and add it to the closed list.

d. Expand the current node by generating its neighbors.

- For each neighbor:
  i. If it is in the closed list, skip it.
  ii. If it is not in the open list or has a lower update its , and parent pointer, then add it to the open list.

3. **Terminate**:
    - If the open list is empty, the goal is unreachable.

Implementation Steps

1. **Define the Environment**:
    - Represent the game search problem as a grid or graph.
    - Define obstacles, start, and goal states.
2. **Heuristic Function**:
    - Use an appropriate heuristic, such as Manhattan Distance or Euclidean Distance, depending on the problem.
3. *Implement the A Algorithm**:
    - Use data structures like priority queues for the open list to efficiently retrieve nodes with the lowest
4. **Run the Algorithm**:
    - Execute the algorithm and visualize the path found.

Conclusion

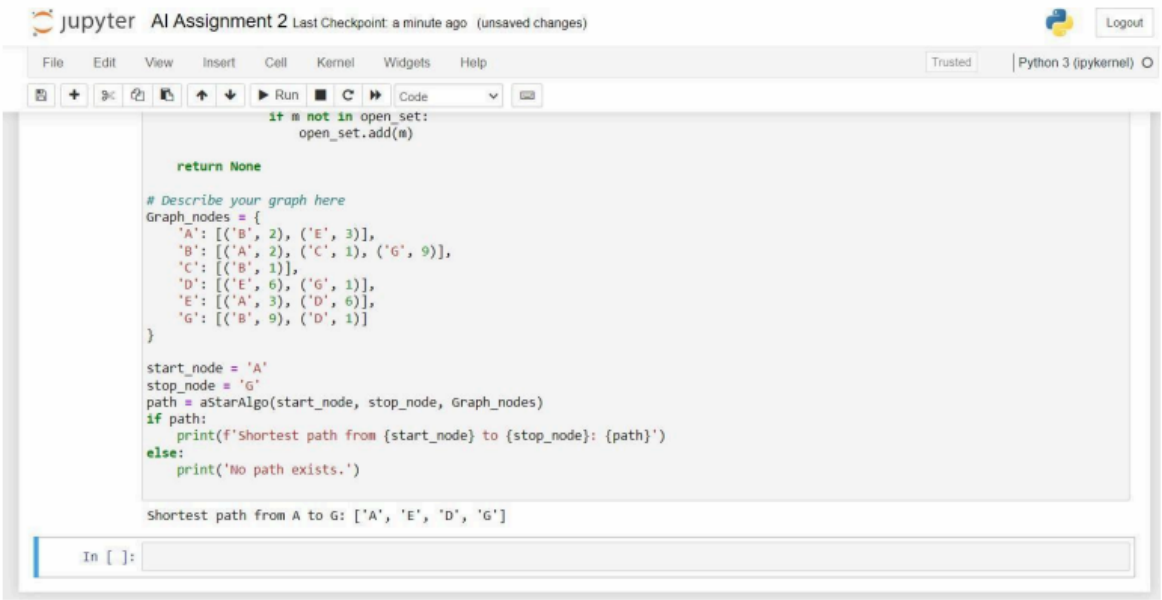
The A* algorithm efficiently finds the shortest path in a game search problem by combining cost-so-far with a heuristic estimate Its use of informed search ensures an optimal and computationally feasible solution compared to uninformed methods like breadth-first or depth-first search. By implementing the algorithm, the student gains hands-on experience in heuristic-based pathfinding and observes its application in real-world scenarios such as games and robotics.

Code:

```
'''Implement A star (A*) Algorithm for any game search problem.'''
def aStarAlgo(start_node, stop_node, graph):
open_set = set([start_node])
closed_set = set()
g = {node: float('inf') for node in graph}
g[start_node] = 0
parents = {node: None for node in graph}
while open_set:
n = min(open_set, key=lambda node: g[node] + heuristic(node))
if n == stop_node:
path = []
while n is not None:
path.append(n) n
= parents[n]
path.reverse()
return path
open_set.remove(n)
closed_set.add(n)
for (m, weight) in graph[n]:
if m in closed_set:
continue
tentative_g = g[n] + weight
if tentative_g < g[m]:
parents[m] = n
g[m] = tentative_g
if m not in open_set:
open_set.add(m)
return None
```

```
# Describe your graph here
Graph_nodes = {
'A': [('B', 2), ('E', 3)],
'B': [('A', 2), ('C', 1), ('G', 9)],
'C': [('B', 1)],
'D': [('E', 6), ('G', 1)],
'E': [('A', 3), ('D', 6)],
'G': [('B', 9), ('D', 1)]
}
start_node = 'A'
stop_node = 'G'
path = aStarAlgo(start_node, stop_node, Graph_nodes)
if path:
print(f'Shortest path from {start_node} to {stop_node}: {path}')
else:
print('No path exists.')
```

Output:



2. **Implement any one of the following expert systems in prolog.**
   a) Medical Diagnosis
   b) Financial Investment Advisor

a) Medical Diagnosis

Title

Implementation of a Medical Diagnosis Expert System in Prolog

Aim

To design and implement a rule-based expert system in Prolog for diagnosing medical conditions based on user-provided symptoms.

Objective

1. Understand the principles of expert systems and their role in decision-making.
2. Explore Prolog as a tool for representing knowledge and inference.
3. Develop a simple expert system that interacts with users and provides diagnostic advice.

Algorithm

Step 1: Define Knowledge Base

- Encode medical knowledge in the form of **rules** and **facts**.
    - Facts represent known information (e.g., symptoms).
    - Rules derive conclusions (e.g., diseases) based on the facts.

Step 2: Input User Symptoms

- Prompt the user to input observed symptoms.

Step 3: Match Symptoms Against Rules

- Use Prolog's inference engine to find rules that match the given symptoms.
- Apply backward or forward chaining to derive conclusions.

Step 4: Provide Diagnosis

- Display the likely medical condition based on the matched rules.

Step 5: Handle Uncertainty (Optional)

- Incorporate confidence levels or probabilities for more realistic diagnoses.

Conclusion

This lab practice demonstrates the implementation of a simple medical diagnosis expert system in Prolog. The system uses a knowledge base of symptoms and rules to infer potential diseases based on user inputs. Prolog's declarative nature and built-in inference engine make it a suitable tool for building rule-based expert systems. The exercise highlights the practical application of AI in domains like healthcare, showcasing how expert systems assist in decision-making under structured rules and knowledge.

Code:

```
% Facts: Symptoms of diseases
symptom(cold, fever).
symptom(cold, runny_nose).
symptom(cold, sore_throat).
symptom(flu, fever).
symptom(flu, headache).
symptom(flu, fatigue).
symptom(malaria, fever).
symptom(malaria, chills).
symptom(malaria, sweating).

% Rules: Diagnosis based on symptoms
diagnose(Disease) :-
    symptom(Disease, fever),
    symptom(Disease, SpecificSymptom),
    ask(SpecificSymptom).

% User interaction
ask(Symptom) :-
    format('Do you have ~w? (yes/no): ', [Symptom]),
    read(Response),
    Response = yes.
```

To diagnose a condition, the user runs:
Prolog

```
?- diagnose(Disease).
```

b) Financial Investment Advisor

Title

Implementation of a Financial Investment Advisor Expert System in Prolog

Aim

To develop an expert system in Prolog to provide investment advice based on the user's financial goals, risk tolerance, and preferences.

Objective

1. Learn the basics of expert system design using Prolog.
2. Represent financial knowledge and rules in Prolog's declarative language.
3. Build an interactive system that recommends suitable investment options based on user input.

Algorithm

Step 1: Define Knowledge Base

- Represent knowledge as **facts** (e.g., investment types, risk levels) and **rules** (e.g., conditions for recommending an investment).
- Encode investment options such as stocks, bonds, mutual funds, and real estate with their associated risk levels and expected returns.

Step 2: Input User Preferences

- Prompt the user for input on financial goals (e.g., retirement, savings), risk tolerance (low, medium, high), and investment horizon (short-term, long-term).

Step 3: Match Preferences Against Rules

- Use Prolog's inference mechanism to match user preferences with predefined rules for recommending investments.

Step 4: Provide Recommendations

- Output the recommended investment options tailored to the user's input.

Example 1

This prolog will advise you whether you should invest your money in savings account, stock market or divide in both (combination). It will decide based on following information: No of dependents, Yearly income, Money saved,

Rules: You should have atleast $5000 saved for each dependent. You should have steady income of $15000/yearly. You should be earning $4000 for each dependent.

If you have more money then mentioned limit you will be advised to invest in stock market, else in savings account

To run this prolog program: First you need to download and install win-prolog and prolog IDE and compile for windows x86-64 platforms available at this link: https://drive.google.com/drive/folders/1l-tvqG2WrtkvankGtjoPyROP4YvJzlii. If you have already installed prolog IDE then you can ignore this. Now download the financial_advisor.pl files in anywhere in your pc. Open the financial_advisor.pl file in notpad or any text editor (Last 3 lines in file will be used as input). Input: e.g. amount_saved(10000)., numDependents(1)., earnings(15000,steady). These parameters will decide the outcome. Save the financial_advisor.pl file. Now open prolog IDE, Click->Load->financial_advisor.pl file. It will just simply show one message in prolog IDE that file has been loaded. Now, its time that you can question from this program, it will show following symbols means its ready for questions. : ?- You can end your question with "." dot/ full stop mark. You can ask investment(X). : ?- investment(X). It will output, savings, stock, combo. Savings means you need to put money only in savings account. Stock means you should invest your money in stock market. Combo means you need to diversfy your investment options and invest in both in stock and savings account.

```
min_savings(Dependents, Amount) :- Amount is 5000 * Dependents.
min_income(Dependents, Amount) :- Amount is (4000 * Dependents) + 15000.
investment(savings) :- savings(inadequate).
investment(stocks) :- savings(adequate), income(adequate).
investment(combo) :- savings(adequate), income(inadequate).
savings(adequate) :-
amount_saved(Amount),numDependents(Dependents),min_savings(Dependents,SavingsMin
),Amount > SavingsMin.
savings(adequate) :- numDependents(Dependents),Dependents == 0.
savings(inadequate) :-
amount_saved(Amount),numDependents(Dependents),min_savings(Dependents,
SavingsMin), Amount =< SavingsMin.
income(adequate) :- earnings(AmountEarned, steady),
numDependents(Dependents),min_income(Dependents, IncomeMin), AmountEarned >
IncomeMin.
income(inadequate) :- earnings(AmountEarned,
steady),numDependents(Dependents),min_income(Dependents, IncomeMin), AmountEarned
=< IncomeMin.
income(adequate) :- numDependents(Dependents), Dependents == 0.
income(inadequate) :- earnings(AmountEarned, unsteady).
income(inadequate) :- earnings(AmountEarned, unsteady).
```

amount_saved(4000).
numDependents(2).
earnings(1000,steady).

Example 2

Code:

Implementation (Example)

1. Knowledge Base in Prolog

```
% Facts: Investment options with risk levels
investment(stocks, high, long_term).
investment(bonds, low, long_term).
investment(mutual_funds, medium, medium_term).
investment(real_estate, medium, long_term).
investment(savings_account, low, short_term).
investment(certificates_of_deposit, low, medium_term).

% Rules: Recommendations based on user preferences
recommend(Investment) :-
    goal(Goal),
    risk_tolerance(Risk),
    time_horizon(Horizon),
    investment(Investment, Risk, Horizon),
    goal_suitable(Investment, Goal).

% Goal suitability
goal_suitable(stocks, growth).
goal_suitable(mutual_funds, growth).
goal_suitable(bonds, stability).
goal_suitable(real_estate, growth).
goal_suitable(savings_account, stability).
```

```
goal_suitable(certificates_of_deposit, stability).


% User inputs
ask_user :-
   write('What is your financial goal? (growth/stability): '), read(Goal), assert(goal(Goal)),
        write('What is your risk tolerance? (low/medium/high): '), read(Risk), assert(risk_tolerance(Risk)),
      write('What is your investment horizon? (short_term/medium_term/long_term): '), read(Horizon), assert(time_horizon(Horizon)).
```

2. **Query Example**

To get investment advice, the user runs:

```
?- ask_user, recommend(Investment).
```

Conclusion

This lab practice demonstrates the implementation of a Financial Investment Advisor expert system in Prolog. The system effectively recommends investment options based on user-defined goals, risk tolerance, and time horizon. By representing financial knowledge and rules declaratively, Prolog simplifies the design and inference processes. This practical exercise highlights how expert systems can assist users in making informed financial decisions.

3 Define the operators for controlling domestic robot; use these operators to plan an activity to be executed by the robot. For example, transferring two/three objects one over the other from one place to another. Prepare a plan and implement the solution in prolog.

Title

Defining Operators and Planning Activities for a Domestic Robot Using Prolog

Aim

To define operators for controlling a domestic robot and use these operators to create a plan for executing an activity, such as transferring objects from one place to another, in Prolog.

Objective

1. Understand the use of operators to model robot actions.
2. Develop a planning system in Prolog to automate a sequence of tasks.
3. Apply AI planning techniques to solve real-world robotics problems.

Algorithm

Step 1: Define Problem Domain

- Identify the objects, locations, and states (e.g., on table, on another object, held by the robot).
- Represent initial and goal states.

Step 2: Define Operators

- Define actions (e.g., pick, place, stack) as operators with preconditions and effects.

Step 3: Define Planning Rules

- Use Prolog's inference mechanism to determine the sequence of actions needed to transition from the initial state to the goal state.

Step 4: Execute Plan

- Simulate the actions based on the derived plan and update the world state accordingly.

Conclusion

This lab demonstrates the use of Prolog to define operators and plan activities for a domestic robot. By modeling actions and their effects, the system can generate a sequence of actions to achieve a specified goal state. This practical exercise highlights the application of AI planning techniques in robotics, showcasing Prolog's utility in reasoning and decision-making for automated systems.

Code:

Plan Outline

1. Define Operators: Define predicates representing actions that the robot can perform, such as
`pick`, `place`, `stack`, and `unstack`.
2. Initial State Representation: Represent the initial state of the environment using facts.
3. Goal State Representation: Define the goal state the robot needs to achieve.
4. Plan Execution: Write Prolog rules to generate a sequence of actions for transforming the initial
state into the goal state.

Implementation

```
% File: robot_planner.pl
% Define the initial state
% Represent objects and their positions
on_table(a).
on_table(b).
on_table(c).
clear(a).
clear(b).
clear(c).
holding(none).
% Define the goal state
goal([on(a, b), on(b, c)]).
% Actions
% Picking up an object
action(pick(X),
    [on_table(X), clear(X), holding(none)],
    [holding(X), clear(X)],
    [on_table(X), clear(X), holding(none)]).
% Placing an object on the table
```



```
action(place(X),
    [holding(X)],
    [on_table(X), clear(X), holding(none)],
    [holding(X)]).
% Stacking an object on another
action(stack(X, Y),
    [holding(X), clear(Y)],
    [on(X, Y), clear(X), holding(none)],
    [holding(X), clear(Y)]).
% Unstacking an object from another
action(unstack(X, Y),
```

```
    [on(X, Y), clear(X), holding(none)],
    [holding(X), clear(Y)],
    [on(X, Y), clear(X), holding(none)]).
% Plan generation
plan(State, Goal, _, []) :-
  subset(Goal, State).
plan(State, Goal, Visited, [Action|Actions]) :
  action(Action, Preconditions, AddList, DeleteList),
  subset(Preconditions, State),
  not(member(State, Visited)),
  subtract(State, DeleteList, TempState),
  union(TempState, AddList, NewState),
  plan(NewState, Goal, [State|Visited], Actions).
% Solve the problem
solve_plan :
  findall(GoalFact, goal(GoalFacts), Goals),
  initial_state(InitialState),
  plan(InitialState, Goals, [], Plan),
  write('Plan: '), writeln(Plan).
% Helper functions
subset([], _).
subset([H|T], List) :-
  member(H, List),
  subset(T, List).
subtract([], _, []).
subtract([H|T], List, Result) :-
  member(H, List),
  subtract(T, List, Result).
subtract([H|T], List, [H|Result]) :-
  not(member(H, List)),
  subtract(T, List, Result).
union([], L, L).
union([H|T], L, Result) :-
  member(H, L),
  union(T, L, Result).
union([H|T], L, [H|Result]) :-
  not(member(H, L)),
  union(T, L, Result)
```

To execute the plan:

1. Load the Prolog file.
2. Query solve_plan. to generate and display the action plan.

4. Implementation of Unification algorithm.

Title

Implementation of the Unification Algorithm in Prolog

Aim

To implement and analyze the unification algorithm, which determines the compatibility of terms by finding a substitution that makes them identical.

Objective

1. Understand the principles of unification in Prolog and logic programming.
2. Implement the unification algorithm to find substitutions for matching terms.
3. Analyze the role of unification in enabling logical inference and query resolution.

Algorithm

Step 1: Define Terms and Variables

- Represent terms as variables, constants, or compound terms.

Step 2: Check Compatibility

- If both terms are identical, unification succeeds with no substitution.
- If one term is a variable, unify it with the other term unless it causes a cyclic reference.

Step 3: Apply Substitutions

- Replace variables in the terms with their corresponding values from the substitution list.

Step 4: Recursively Unify Compound Terms

- Decompose compound terms into their arguments and recursively apply the unification process.

Step 5: Return Substitution

- If unification succeeds, return the substitution list. Otherwise, indicate failure.

Conclusion

This lab exercise demonstrates the implementation of the unification algorithm in Prolog. The algorithm successfully determines the substitutions required to unify two terms, highlighting its role in enabling logical inference and query resolution in Prolog. This practical exercise provides insights into how Prolog efficiently resolves queries through its built-in unification mechanism, an essential aspect of logic programming**.**

Code

```
# File: unification_algorithm.py


def is_variable(term):
    """Check if a term is a variable."""
    return isinstance(term, str) and term.islower()


def unify(x, y, subst):
    """
    Unification algorithm: Attempts to make two terms equal.
    :param x: First term
    :param y: Second term
    :param subst: Current substitution
    :return: Updated substitution or failure
    """
    if subst is None:
        return None
    elif x == y:
        return subst
    elif is_variable(x):
        return unify_var(x, y, subst)
    elif is_variable(y):
        return unify_var(y, x, subst)
    elif isinstance(x, tuple) and isinstance(y, tuple) and len(x) == len(y):
```

```
        for xi, yi in zip(x, y):
            subst = unify(xi, yi, subst)
            if subst is None:
                return None
        return subst
    else:
        return None


def unify_var(var, x, subst):
    """
    Unify a variable with a term.
    :param var: Variable
    :param x: Term
    :param subst: Current substitution
    :return: Updated substitution
    """
    if var in subst:
        return unify(subst[var], x, subst)
    elif x in subst:
        return unify(var, subst[x], subst)
    elif occurs_check(var, x):
        return None
    else:
        subst[var] = x
        return subst


def occurs_check(var, x):
```

```
    """
    Check for cyclic dependencies (occurs check).
    :param var: Variable
    :param x: Term
    :return: True if var occurs in x, False otherwise
    """
    if var == x:
        return True
    elif isinstance(x, tuple):
        return any(occurs_check(var, xi) for xi in x)
    return False


# Test cases
if __name__ == "__main__":
    print("Unification Algorithm Test Cases:")
    tests = [
        (("x", "y"), ("a", "b")),  # Simple variables and constants
        (("x", "y"), ("a", "a")),  # Unifiable terms
        (("x", "y"), ("x", "b")),  # Contradiction
        (("f", ("x", "y")), ("f", ("a", "b"))),  # Nested terms
        (("f", "x"), ("f", ("a", "b"))),  # Structure mismatch
        (("x", "f", ("y",)), ("f", "f", ("z",))),  # Mixed variables and functions
    ]

    for i, (x, y) in enumerate(tests, start=1):
        subst = {}
        result = unify(x, y, subst)
        print(f"Test {i}: {x} = {y} ->", "Success:" if result else "Failure:", result)
```