**Lab Assignment 3: Implementation of Constraint Satisfaction Problem (CSP)**

**1. Objective:**

The goal of this lab assignment is to implement the **Constraint Satisfaction Problem (CSP)** framework to solve problems where a solution must satisfy a set of constraints. Examples of CSPs include problems like Sudoku, the 8-queens problem, and graph coloring. The assignment will focus on understanding how constraints can be used to limit the search space and improve solution efficiency.

**2. Problem Statement:**

Given a set of variables, domains, and constraints, implement a solution to a Constraint Satisfaction Problem using backtracking. This includes formulating a problem (e.g., n-queens or Sudoku), defining the constraints, and solving the problem by systematically assigning values to variables while respecting constraints.

**3. Theory:**

**3.1. Constraint Satisfaction Problems (CSP):**

A **Constraint Satisfaction Problem (CSP)** is a mathematical problem where the objective is to find an assignment to a set of variables that satisfies a given set of constraints. Formally, a CSP consists of:

* **Variables**: A set of variables, X={X1,X2,...,Xn}X = \{X\_1, X\_2, ..., X\_n\}X={X1​,X2​,...,Xn​}, each with a domain of possible values.
* **Domains**: For each variable, XiX\_iXi​, there is a corresponding domain DiD\_iDi​, which defines the possible values that the variable can take.
* **Constraints**: A set of rules or conditions that limit the values the variables can take. A constraint is typically a relation that must hold between two or more variables.

Examples of CSPs:

* **Sudoku**: Variables represent the cells, domains are the numbers 1–9, and constraints enforce that each row, column, and subgrid must contain distinct numbers.
* **N-Queens Problem**: Variables represent the positions of queens on a chessboard, domains are the rows and columns, and constraints prevent queens from attacking each other.

**3.2. Backtracking for CSP:**

Backtracking is a depth-first search method used to solve CSPs. It systematically assigns values to variables and checks constraints after each assignment. If a constraint is violated, the algorithm backtracks to the previous variable and tries a different value.

**3.3. Key Techniques in CSP:**

* **Backtracking Search**: A basic approach that assigns values to variables incrementally and undoes assignments when a constraint is violated.
* **Forward Checking**: Prevents assigning values to variables that would immediately violate a constraint by reducing the domains of unassigned variables.
* **Arc Consistency (AC-3)**: Ensures that every variable assignment is consistent with every constraint before making further assignments, thereby pruning the search space.

**4. Algorithm Design:**

**4.1. Variables:**

For any CSP, the variables are the elements that need to be assigned values. For example:

* In the 8-queens problem, variables represent the column positions of queens on an 8x8 board.
* In Sudoku, variables represent the cells on a 9x9 grid.

**4.2. Domains:**

Each variable has a domain of possible values. For example:

* In the 8-queens problem, the domain is the set of row numbers for each queen.
* In Sudoku, the domain for each cell is the set {1, 2, ..., 9}.

**4.3. Constraints:**

The constraints define the rules of the problem. For example:

* In the 8-queens problem, no two queens can share the same row, column, or diagonal.
* In Sudoku, no two cells in the same row, column, or 3x3 subgrid can contain the same number.

**5. Pseudocode:**

**5.1. Backtracking for CSP:**

sql

Copy code

CSP\_Backtracking(variables, domains, constraints):

if all variables are assigned:

return solution

select an unassigned variable

for each value in domain of the selected variable:

if the value is consistent with the constraints:

assign the value to the variable

result = CSP\_Backtracking(variables, domains, constraints)

if result is not failure:

return result

remove the assignment (backtrack)

return failure

**5.2. Forward Checking (Optional Extension):**

sql

Copy code

Forward\_Checking(variable, value, domains, constraints):

for each unassigned variable:

for each value in the domain of the unassigned variable:

if the assignment of the value violates a constraint:

remove the value from the domain of the unassigned variable

if any domain is empty, return failure

**6. Example: Solving the 8-Queens Problem:**

In the **8-queens problem**, the objective is to place 8 queens on a chessboard in such a way that no two queens can attack each other. The variables represent the positions of the queens in each column, and the constraints ensure that no two queens share the same row, column, or diagonal.

* **Variables**: X={Q1,Q2,...,Q8}X = \{Q\_1, Q\_2, ..., Q\_8\}X={Q1​,Q2​,...,Q8​}, where QiQ\_iQi​ represents the row position of the queen in column iii.
* **Domains**: For each QiQ\_iQi​, the domain is the set of possible row numbers: Di={1,2,...,8}D\_i = \{1, 2, ..., 8\}Di​={1,2,...,8}.
* **Constraints**: No two queens can be in the same row or diagonal.

**Pseudocode for the 8-Queens Problem:**

sql

Copy code

CSP\_Backtracking(queens, rows, constraints):

if all queens are placed:

return solution

select a column for the next queen

for each row in 1 to 8:

if placing the queen in this row does not violate constraints:

assign the queen to this row

result = CSP\_Backtracking(queens, rows, constraints)

if result is not failure:

return result

remove the queen (backtrack)

return failure

**7. Expected Output:**

For problems like the 8-queens or Sudoku, the program should output a valid configuration of the board or grid that satisfies all the constraints. For example, in the 8-queens problem, the output will be the positions of the queens on the board such that no two queens attack each other.

**8. Conclusion:**

This lab assignment explores the implementation of CSP using backtracking. Through the 8-queens problem (or any other chosen CSP), we see how constraint satisfaction reduces the complexity of search problems by limiting the possible assignments of variables to only those that satisfy the given constraints.

**9. References:**

* Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.
* Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

This lab format covers CSP theory, algorithm design, pseudocode, and examples to help you implement a constraint satisfaction solver using backtracking.

4o

**You said:**

Perform parsing of family tree using knowledge-base. same as above

**ChatGPT said:**

ChatGPT

**Lab Assignment: Parsing of a Family Tree Using Knowledge Base**

**1. Objective:**

The objective of this lab assignment is to implement a **Family Tree Parsing System** using a **Knowledge Base (KB)** and logical rules to infer relationships. The system will store family relations (like parent, sibling, child) in a knowledge base and use rules to parse and deduce new relationships, such as grandparent, cousin, and uncle/aunt.

**2. Problem Statement:**

Given a family tree and a set of relations stored in a knowledge base, implement a system that can infer additional family relationships by using logical rules and parsing techniques. The system will define family relationships (e.g., parent, child) and use inference rules to deduce indirect relationships like grandparent, cousin, or sibling.

**3. Theory:**

**3.1. Family Tree:**

A **family tree** is a diagram representing family relationships in a hierarchical structure. Each node represents a person, and the edges between nodes represent familial relationships such as parent, sibling, or spouse.

**3.2. Knowledge Base (KB):**

A **Knowledge Base (KB)** is a collection of facts and rules about a particular domain. For the family tree, the knowledge base will store facts about individuals and their relationships, such as:

* **Facts**: Direct relationships between family members (e.g., "John is the parent of Alice").
* **Rules**: Inference rules that allow the system to deduce indirect relationships (e.g., "If X is the parent of Y and Y is the parent of Z, then X is the grandparent of Z").

**3.3. Logic and Inference Rules:**

The family relationships can be expressed as logical predicates:

* **Parent(X, Y)**: X is the parent of Y.
* **Child(Y, X)**: Y is the child of X.
* **Sibling(X, Y)**: X and Y have the same parent(s).
* **Grandparent(X, Y)**: X is the parent of Y's parent.
* **Cousin(X, Y)**: X and Y are children of siblings.

The system will use these predicates to infer new relationships by applying logical rules.

**4. Algorithm Design:**

**4.1. Step 1: Defining the Knowledge Base (KB)**

The knowledge base consists of facts and rules that represent family relationships. You can define these in terms of predicates like Parent(X, Y), Sibling(X, Y), etc.

**Example Facts:**

* Parent(John, Alice).
* Parent(John, Bob).
* Parent(Alice, Charlie).
* Parent(Bob, David).

**Example Rules:**

* **Rule for Sibling**:
  + If Parent(P, X) and Parent(P, Y) and X ≠ Y, then Sibling(X, Y).
* **Rule for Grandparent**:
  + If Parent(X, Y) and Parent(Y, Z), then Grandparent(X, Z).

**4.2. Step 2: Parsing Relationships Using Rules**

The system will apply logical inference to deduce new relationships:

* For each fact in the knowledge base, apply the rules to infer additional relationships.
* Use backtracking to check all possible combinations of relationships.

**4.3. Step 3: Querying the Knowledge Base**

Once the knowledge base is populated with facts and rules, the system should allow queries like:

* Who are the grandparents of Charlie?
* Is Alice a sibling of Bob?
* Who are the cousins of David?

**5. Pseudocode:**

**5.1. Knowledge Base Initialization:**

scss

Copy code

Initialize\_KnowledgeBase():

KB = set()

Add\_Fact(KB, Parent(John, Alice))

Add\_Fact(KB, Parent(John, Bob))

Add\_Fact(KB, Parent(Alice, Charlie))

Add\_Fact(KB, Parent(Bob, David))

return KB

**5.2. Inference Rules:**

scss

Copy code

Infer\_Sibling(KB):

for each X, Y, P in KB:

if Parent(P, X) and Parent(P, Y) and X ≠ Y:

Add\_Fact(KB, Sibling(X, Y))

Infer\_Grandparent(KB):

for each X, Y, Z in KB:

if Parent(X, Y) and Parent(Y, Z):

Add\_Fact(KB, Grandparent(X, Z))

**5.3. Querying the Knowledge Base:**

scss

Copy code

Query\_Grandparent(KB, Z):

for each fact in KB:

if Grandparent(X, Z):

print(X, "is the grandparent of", Z)

Query\_Sibling(KB, X, Y):

if Sibling(X, Y) in KB:

print(X, "is a sibling of", Y)

else:

print(X, "is not a sibling of", Y)

**6. Example: Parsing a Family Tree**

**Step 1: Define Facts in the Knowledge Base**

scss

Copy code

KB = {

Parent(John, Alice),

Parent(John, Bob),

Parent(Alice, Charlie),

Parent(Bob, David)

}

**Step 2: Apply Inference Rules**

* Infer Sibling: From Parent(John, Alice) and Parent(John, Bob), we infer that Sibling(Alice, Bob).
* Infer Grandparent: From Parent(Alice, Charlie), we know that Grandparent(John, Charlie).

**Step 3: Querying the System**

* **Query**: Who is the grandparent of Charlie?
  + **Output**: "John is the grandparent of Charlie."
* **Query**: Is Alice a sibling of Bob?
  + **Output**: "Yes, Alice is a sibling of Bob."
* **Query**: Who is the cousin of Charlie?
  + **Output**: "David is the cousin of Charlie."

**7. Expected Output:**

For a given family tree, the system should:

* Parse relationships and infer new relationships using the provided rules.
* Correctly respond to queries about family relationships (e.g., finding grandparents, cousins, or siblings).

Example Output for the family tree:

scss

Copy code

KB = {

Parent(John, Alice),

Parent(John, Bob),

Parent(Alice, Charlie),

Parent(Bob, David),

Sibling(Alice, Bob),

Grandparent(John, Charlie),

Cousin(Charlie, David)

}

Query\_Grandparent(KB, Charlie) => John is the grandparent of Charlie.

Query\_Sibling(KB, Alice, Bob) => Alice is a sibling of Bob.

Query\_Cousin(KB, Charlie, David) => Charlie is a cousin of David.

**8. Conclusion:**

This lab assignment demonstrates how a family tree can be parsed using a knowledge base and logical inference. By storing basic family relationships and applying rules, the system can infer complex relationships like grandparents, siblings, and cousins. This approach can be extended to larger and more complex family trees, demonstrating the power of knowledge-based systems in reasoning.

**9. References:**

* Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
* Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.