Name: Shabbar Adamjee
Roll No.: PB57
PRN: 1032221508

AIES ASSIGNMENT 4

Unification Algorithm

Code

```python
from dataclasses import dataclass
from typing import Dict, List, Optional, Union


@dataclass
class Constant:
    value: str


@dataclass
class Variable:
    value: str


@dataclass
class Function:
    name: str
    args: List["Term"]


Term = Union[Constant, Variable, Function]


@dataclass
class Relation:
    predicate: str
    args: List[Term]

    def __str__(self):
        args_str = ", ".join(str(arg) for arg in self.args)
        return f"{self.predicate}({args_str})"


def occurs_check(var: Variable, term: Term, substitution: Dict[str, Term]) ->
bool:
    if isinstance(term, Variable):
```

```
        if var.value == term.value:
            return True
        elif term.value in substitution:
            return occurs_check(var, substitution[term.value], substitution)
    elif isinstance(term, Constant):
        return False
    elif isinstance(term, Function):
        return any(occurs_check(var, arg, substitution) for arg in term.args)
    return False


def apply_substitution(term: Term, substitution: Dict[str, Term]) -> Term:
    if isinstance(term, Variable) and term.value in substitution:
        return apply_substitution(substitution[term.value], substitution)
    elif isinstance(term, Function):
        return Function(
            term.name, [apply_substitution(arg, substitution) for arg in
term.args]
        )
    return term


def unify_terms(
    term1: Term, term2: Term, substitution: Dict[str, Term]
) -> Optional[Dict[str, Term]]:
    term1 = apply_substitution(term1, substitution)
    term2 = apply_substitution(term2, substitution)

    if term1 == term2:
        return substitution
    elif isinstance(term1, Variable):
        if occurs_check(term1, term2, substitution):
            return None
        substitution[term1.value] = term2
        return substitution
    elif isinstance(term2, Variable):
        return unify_terms(term2, term1, substitution)
    elif isinstance(term1, Function) and isinstance(term2, Function):
        if term1.name != term2.name or len(term1.args) != len(term2.args):
            return None
        for arg1, arg2 in zip(term1.args, term2.args):
            substitution = unify_terms(arg1, arg2, substitution)
            if substitution is None:
                return None
        return substitution
    elif isinstance(term1, Constant) or isinstance(term2, Constant):
        return None
    else:
        return None
```

```python
def unifier(r1: Relation, r2: Relation) -> Optional[Dict[str, Term]]:
    if r1.predicate != r2.predicate or len(r1.args) != len(r2.args):
        return None

    substitution = {}
    for arg1, arg2 in zip(r1.args, r2.args):
        result = unify_terms(arg1, arg2, substitution)
        if result is None:
            return None
        substitution = result

    return substitution


def main():
    # Example usage
    relation1 = Relation("Knows", [Constant("Raj"), Variable("X")])
    relation2 = Relation("Knows", [Variable("Y"), Constant("Seeta")])

    result = unifier(relation1, relation2)

    if result:
        print("Unification successful:")
        for var, term in result.items():
            print(f"{var} = {term}")
    else:
        print("Unification failed")

    # Additional test cases
    relation3 = Relation("Likes", [Variable("X"), Variable("Y")])
    relation4 = Relation("Likes", [Constant("John"), Constant("Pizza")])

    result2 = unifier(relation3, relation4)
    if result2:
        print("\nUnification successful:")
        for var, term in result2.items():
            print(f"{var} = {term}")
    else:
        print("\nUnification failed")

    # Test case with nested functions
    relation5 = Relation(
        "Father", [Function("Parent", [Variable("X")]), Constant("John")]
    )
    relation6 = Relation(
        "Father", [Function("Parent", [Constant("Mary")]), Variable("Y")]
    )
```

```
    result3 = unifier(relation5, relation6)
    if result3:
        print("\nUnification successful:")
        for var, term in result3.items():
            print(f"{var} = {term}")
    else:
        print("\nUnification failed")


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

Output

```
AIES python .\unification2.py
Unification successful:
Y = Constant(value='Raj')
X = Constant(value='Seeta')

Unification successful:
X = Constant(value='John')
Y = Constant(value='Pizza')

Unification successful:
X = Constant(value='Mary')
Y = Constant(value='John')
```

Page No.
Date :

Assignment No. 4

Title

Implementation of Unification Algorithm

FAQs

1. Why resolution is required?

Resolution is required as a proof procedure in automated theorem proving and logical reasoning. It is used to derive contradictions from a set of clauses, ultimately proving that a certain statement logically follows from the premises.

In first-order logic, resolution helps in refuting the negation of the goal, thereby confirming the validity of the original goal. It is essential for deriving conclusions in automated reasoning systems.

2. What are the prerequisites for applying the unification algorithm?

- Both expressions (literals or terms) must be well-formed and in the same formal language (typically first-order logic).

- The two literals or terms should be of the same arity (i.e. the same number of arguments) for the algorithm to attempt unification.

- Free variables in one literal must not conflict with bound variables in the other, ensuring that they can be substituted consistently.

PJ 18/10/24

3. What are the applications of the unification algorithm?

- Automated theorem proving
- Logic programming (eg. Prolog)
- Type inference
- Artificial intelligence