TRIBHUVAN UNIVERSITY

Institute of Science and Technology (IoST)

Samriddhi College

A Lab Report on Artificial Intelligence

LAB 8: Propositional and Predicate Logic with Resolution Algorithm

Submitted To:

Bikram Acharya

Submitted By:

Kanchan Joshi (Roll no.10)

Date: October 04, 2025

1. Introduction

Logical reasoning serves as the cornerstone of artificial intelligence and computational reasoning frameworks. This experimental investigation examines two core logic systems: Propositional Logic and First-Order Predicate Logic (FOPL), focusing particularly on the resolution method as a technique for mechanized theorem validation. The source code implementations and complete materials for the logical systems examined in this laboratory work are maintained in an accessible repository for future reference.

Propositional logic operates with declarative statements that possess definite truth values, employing logical operators including conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and conditional statements ($\rightarrow$). Despite its utility across numerous domains, propositional logic encounters constraints when addressing object relationships and quantified expressions.

First-Order Predicate Logic builds upon propositional foundations by incorporating predicates, functions, constants, variables, and quantification operators. This extension enables richer knowledge representation and supports reasoning about entities and their interconnections.

1.1 Resolution Method Overview

The resolution method, developed by J.A. Robinson in 1965, represents a core inference mechanism employed in computational theorem proving. The method operates through:

1. Transforming expressions into Conjunctive Normal Form (CNF)
2. Incorporating the negated query into the knowledge set
3. Iteratively executing resolution operations to generate additional clauses
4. Concluding when either a null clause (indicating contradiction) emerges or no additional clauses can be produced

Resolution's strength stems from its completeness property - when a statement follows logically from a knowledge foundation, resolution will ultimately establish its validity.

1.2 Unification Process in First-Order Logic

First-order logic resolution necessitates unification - the procedure of discovering substitutions that render two expressions equivalent. The unification process encompasses:

- Variable Binding: Substituting variables with terms to achieve correspondence
- Occurrence Verification: Preventing infinite constructions by ensuring variables don't appear within terms they're unified with
- Most General Unifier (MGU): Identifying the most comprehensive substitution that unifies two expressions

2. Propositional Logic Implementation

Propositional logic establishes the groundwork for comprehending logical inference. Our implementation showcases the resolution method through a concrete example involving meteorological conditions and their implications.

2.1 Problem Specification

Examine the following knowledge foundation:

1. When precipitation occurs, the surface becomes wet: $R_p \rightarrow W_s$
2. When the surface is wet, conditions become slippery: $W_s \rightarrow S_c$
3. Precipitation is occurring: R_p

Query: Are conditions slippery? S_c

2.2 Propositional Resolution Framework

```python
class PropositionalClause:
    """Represents a logical clause in propositional logic (
        disjunction of literals)"""
    def __init__(self, literals: List[str]):
        self.literals = set(literals)
    def __str__(self):
        if not self.literals:
            return "EMPTY" # Empty clause (contradiction)
        return " OR ".join(sorted(self.literals))
    def is_empty(self):
        return len(self.literals) == 0
    def resolve_with(self, other):
        """Apply resolution between this clause and another clause"""
        new_literals = set()
        resolved = False
```

```
        for lit1 in self.literals:
            for lit2 in other.literals:
                # Check if literals are complementary
                if lit1.startswith('NOT_ ') and lit2 == lit1[5:]:
                    # Resolve NOT_P_v and P_v
                    new_literals = (self.literals - {lit1}) | (other.
                        literals - {lit2})
                    resolved = True
                    break
                elif lit2.startswith('NOT_ ') and lit1 == lit2[5:]:
                    # Resolve P_v and NOT_P_v
                    new_literals = (self.literals - {lit1}) | (other.
                        literals - {lit2})
                    resolved = True
                    break
            if resolved:
                break
        if resolved:
            return PropositionalClause(list(new_literals))
        return None
```

Code 2.1: Propositional Logic Clause Representation

2.3 Resolution Method Implementation

```
def propositional_resolution(clauses: List[PropositionalClause]) ->
    Tuple[bool, List[str]]:
    """Execute resolution algorithm on propositional clauses"""
    clauses_set = set(clauses)
    steps = []
    steps.append("Starting clauses:")
    for i, clause in enumerate(clauses):
        steps.append(f"{i+1}. {clause}")
    iteration = 1
    while True:
        steps.append(f"\nIteration {iteration}:")
        iteration_new = set()
        # Attempt to resolve each pair of clauses
        clause_list = list(clauses_set)
        for i in range(len(clause_list)):
            for j in range(i + 1, len(clause_list)):
```

```
            resolvent = clause_list[i].resolve_with(clause_list[j])
            if resolvent is not None:
                steps.append(f"Combining '{clause_list[i]}' and '{
                    clause_list[j]}' -> '{resolvent}'")
                if resolvent.is_empty():
                    steps.append("SUCCESS: Empty clause derived!
                        Contradiction established.")
                    return True, steps
                iteration_new.add(resolvent)
    # Check if no new clauses were generated
    if iteration_new.issubset(clauses_set):
        steps.append("No new clauses generated. Resolution
            terminates.")
        return False, steps
    clauses_set.update(iteration_new)
    iteration += 1
```

Code 2.2: Propositional Resolution Process

2.4 Propositional Logic Demonstration Results

Knowledge Base Transformation to CNF:

1. $R_p \rightarrow W_s \equiv \neg R_p \vee W_s$
2. $W_s \rightarrow S_c \equiv \neg W_s \vee S_c$
3. R_p
4. $\neg S_c$ (query negation)

Resolution Execution:
Starting clauses:
1. $W_s \vee \neg R_p$
2. $S_c \vee \neg W_s$
3. R_p
4. $\neg S_c$

Iteration 1:

- Combining R_p with $W_s \vee \neg R_p$ yields W_s
- Combining $\neg S_c$ with $S_c \vee \neg W_s$ yields $\neg W_s$

- Combining $S_c \vee \neg W_s$ with $W_s \vee \neg R_p$ yields $S_c \vee \neg R_p$

Iteration 2:

- Combining W_s with $\neg W_s$ yields $\square$ (Empty clause)

Empty clause derived! Contradiction established.

Result: Query is PROVABLE

Consequently, based on the established knowledge foundation, we can definitively demonstrate that "conditions are slippery" when "precipitation occurs."

3. First-Order Predicate Logic Framework

First-Order Predicate Logic (FOPL) enhances propositional logic through predicates, functions, constants, variables, and quantifiers, facilitating more comprehensive knowledge representation.

3.1 Predicate Resolution Framework

The implementation for predicate logic extends the propositional one by incorporating unification to handle variables and predicates.

```
def unify(lit1, lit2):
   """Simple unification function for literals"""
   if lit1 == lit2:
      return {}
   if is_variable(lit1):
      if lit1 in lit2 or occurs_check(lit1, lit2):
         return None
      return {lit1: lit2}
   if is_variable(lit2):
      return unify(lit2, lit1)
   if is_compound(lit1) and is_compound(lit2):
      if lit1.functor != lit2.functor or len(lit1.args) != len(lit2.
         args):
         return None
      sub = {}
      for a1, a2 in zip(lit1.args, lit2.args):
         s = unify(a1, a2)
         if s is None:
            return None
         compose_sub(sub, s)
      return sub
   return None
```

Code 3.1: Unification Function

```python
class PredicateClause:
    """Represents a clause in predicate logic"""
    def __init__(self, literals):
        self.literals = literals # List of literals

    def resolve_with(self, other):
        """Resolve two clauses using unification"""
        for l1 in self.literals:
            for l2 in other.literals:
                if is_negation(l1, l2):
                    sub = unify(l1.atom, l2.atom)
                    if sub:
                        # Apply substitution and combine remaining
                            literals
                        new_lits = apply_sub(self.literals + other.
                            literals, sub)
                        new_lits.remove(l1)
                        new_lits.remove(l2)
                        return PredicateClause(new_lits)
        return None
```

Code 3.2: Predicate Clause Representation

3.2 Past Exam Question 1 (2080): Resolution as Rule of Inference and Proof

How resolution algorithm is used as a rule of inference in predicate logic? Convert the following sentences into FOPL. All over smart persons are stupid. Children of all stupid persons are naughty. Roney is child of Harry. Harry is over smart. Prove that "Roney is naughty".

The resolution algorithm serves as a complete inference rule in predicate logic by systematically deriving new clauses from existing ones until a contradiction is reached or no further derivations are possible. In predicate logic, it involves:

1. Converting sentences to clausal form (CNF with Skolemization for universals and dropping existentials).

2. Using unification to find substitutions that make literals complementary.

3. Resolving complementary literals to produce resolvents.
4. Adding the negated goal and repeating until the empty clause is derived, proving the goal.

FOPL Representation:

1. $\forall x$ (OverSmart(x) $\rightarrow$ Stupid(x))
2. $\forall x$ (Stupid(x) $\rightarrow \forall y$ (Child(y, x) $\rightarrow$ Naughty(y)))
3. Child(Roney, Harry)
4. OverSmart(Harry)

Goal: Naughty(Roney)
Clausal Form (CNF):
1. {$\neg$OverSmart(x), Stupid(x)}
2. {$\neg$Stupid(x), $\neg$Child(y, x), Naughty(y)}
3. {Child(Roney, Harry)}
4. {OverSmart(Harry)}
Negated Goal: {$\neg$Naughty(Roney)}
Resolution Steps:

- Resolve 4 and 1 (unify x=Harry): {Stupid(Harry)}
- Resolve {Stupid(Harry)} and 2 (unify x=Harry): {$\neg$Child(y, Harry), Naughty(y)}
- Resolve above and 3 (unify y=Roney): {Naughty(Roney)}
- Resolve {Naughty(Roney)} and negated goal: Empty clause

Empty clause derived, proving the goal.

3.3 Past Exam Question 2 (2080): CNF Conversion Rules

Write the rules to convert statements in predicate logic into CNF form. Convert the following sentences into FOPL. All students of BSC CSIT are intelligent person.

Rules for Conversion to CNF:

1. Eliminate implications: $P \rightarrow Q \equiv \neg P \vee Q$
2. Move negations inward: $\neg \forall x P \equiv \exists x \neg P$, $\neg \exists x P \equiv \forall x \neg P$

3. Standardize variables apart.
4. Skolemize: Replace $\forall x \exists y P(x, y)$ with $P(x, f(x))$ where f is a Skolem function.
5. Drop existential quantifiers.
6. Convert to prenex form if needed.
7. Distribute disjunctions over conjunctions to get clausal form.

FOPL: $\forall s$ (Student(s) $\wedge$ Of(s, BSC_CSIT) $\rightarrow$ Intelligent(s))
Equivalent: $\forall s$ ($\neg$Student(s) $\vee \neg$Of(s, BSC_CSIT) $\vee$ Intelligent(s))
CNF: {$\neg$Student(s), $\neg$Of(s, BSC_CSIT), Intelligent(s)}

3.4 Past Exam Question 3 (Model 2081): Proof Using Resolution for Likes

Consider the following statements: Rabin likes only easy courses. Science courses are hard. All courses in the CSIT are easy. CSC 101 is a CSIT course. Translate the sentences into predicate logic and convert into clausal normal form (CNF). Prove that Rabin likes CSC 101.

FOPL Representation:

1. $\forall x$ (Likes(Rabin, x) $\rightarrow$ Easy(x))
2. $\forall x$ (Science(x) $\rightarrow$ Hard(x))
3. $\forall x$ (CSIT(x) $\rightarrow$ Easy(x))
4. CSIT(CSC101)

Goal: Likes(Rabin, CSC101)
Clausal Form (CNF):
1. {$\neg$Likes(Rabin, x), Easy(x)}
2. {$\neg$Science(x), Hard(x)}
3. {$\neg$CSIT(x), Easy(x)}
4. {CSIT(CSC101)}
Negated Goal: {$\neg$Likes(Rabin, CSC101)}
Resolution Steps:

- Resolve 4 and 3 (unify x=CSC101): {Easy(CSC101)}
- Resolve {Easy(CSC101)} and 1 (unify x=CSC101): {Likes(Rabin, CSC101)}

- Resolve {Likes(Rabin, CSC101)} and negated goal: Empty clause

Empty clause derived, proving the goal.

3.5 Past Exam Question 4 (Model 2081): Differentiation and Clausal Form

Differentiate propositional and predicate logic. What is clausal form? How is it useful? Define a well-formed formula (wff).

Propositional Logic vs Predicate Logic:

Propositional Logic deals with atomic propositions that are either true or false, using connectives like $\wedge, \vee, \neg$. It cannot express relationships between objects.

Predicate Logic extends it with predicates, variables, quantifiers ($\forall, \exists$), allowing expressions like $\forall x P(x)$.

Clausal Form: A conjunction of disjunctions of literals (CNF for resolution). Useful for automated theorem proving as it simplifies resolution.

Well-Formed Formula (wff): A syntactically correct logical expression, e.g., $(P \vee Q) \wedge \neg R$.

3.6 Past Exam Question 5 (Model 2081): Unification in Resolution

Explain the unification process in first-order logic resolution with an example.

Unification finds substitutions to make two expressions identical. It involves variable binding, occurrence test, and finding MGU.

Example: Unify Knows(John, x) and Knows(y, Mary): MGU $\{x$/Mary, y/John$\}$.

This ensures complementary literals match for resolution.

4. Discussion

The resolution algorithm provides a sound and complete method for automated reasoning in both propositional and predicate logic. In propositional logic, it efficiently handles finite domains but struggles with scalability due to exponential clause growth. Predicate logic extends this capability through unification, enabling generalization over objects, but introduces complexity in handling quantifiers and substitutions.

The examples demonstrate resolution's refutation completeness: deriving the empty clause proves entailment. However, practical limitations include the need for efficient unification algorithms and strategies like set-of-support to prune search spaces. Compared to forward chaining, resolution excels in backward reasoning for goal-directed proofs.

In AI applications, such as expert systems, resolution underpins theorem provers like Prover9. Future enhancements could integrate with machine learning for heuristic guidance in clause selection.

5. Conclusion

This laboratory demonstrates the application of resolution in both propositional and predicate logic, showcasing its role in automated theorem proving. The propositional example illustrates basic inference, while the predicate examples highlight unification and handling of quantifiers. These techniques form the basis for advanced AI reasoning systems.

Future work could explore extensions like higher-order logic or integration with modern solvers.