**BVRIT HYDERABAD**

**College of Engineering for Women**

**Department of CSE(AIML)**

CASE STUDY

**DISJUNCTIVE NORMAL FORM(DNF)**

**TEAM 1**

G. Revathi - 22WH1A6606

N. Divya Saahithya - 22WH1A6616

K. Vijaya Rajasree - 22WH1A6648

K. Sreeja - 22WH1A6657

P. Harshini - 22WH1A6660

**Under the guidance of**

**Ms A. Naga Kalyani (Asst.Professor)**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Content** | **Pg.No** |
| **1** | **Introduction** | **3** |
| **2** | **Problem Statement** | **4** |
| **3** | **Methodology** | **4** |
| **4** | **Implementation** | **6** |
| **5** | **Results** | **7** |
| **6** | **Discussion** | **8** |
| **7** | **Conclusion** | **8** |

**1. Introduction**

**1.1 Background**

Disjunctive Normal Form (DNF) is a standardized way of structuring logical expressions in propositional logic. DNF simplifies complex logical statements, making them easier to analyze and manipulate. In DNF, a logical expression is represented as a disjunction of one or more conjunctions of literals. Understanding and converting expressions to DNF is crucial in various fields such as computer science, artificial intelligence, and digital logic design.

**1.2 Objective**

The objective of this project is to write a program that converts any given propositional logic expression into Disjunctive Normal Form (DNF). This involves developing an algorithm that can systematically transform logical expressions to their equivalent DNF representation.

**1.3 Significance**

Converting logical expressions to DNF is valuable because it:

* **Simplifies Logical Expressions**: Makes complex logic easier to understand and work with.
* **Enhances Computational Efficiency**: DNF is often used in computational logic applications for more efficient processing.
* **Facilitates Automated Reasoning**: DNF representations are useful in automated reasoning and formal verification processes.

**2. Problem Statement**

The project addresses the problem of transforming arbitrary propositional logic expressions into their Disjunctive Normal Form (DNF). This transformation is essential for simplifying logical analysis and improving computational efficiency. The primary challenge lies in correctly applying logical equivalences and ensuring the resulting expression is in the correct DNF structure.

**3. Methodology**

**3.1 Approach**

The approach to solving this problem involves several key steps:

* **Parse the Expression**: Convert the input string into a structured format (e.g., parse tree).
* **Eliminate Implications**: Replace implications with their equivalent expressions using logical equivalences.
* **Move Negations Inward**: Apply De Morgan's laws to push negations to the literal level.
* **Distribute Disjunctions Over Conjunctions**: Use distributive properties to achieve the DNF structure.

**3.2 Tools and Techniques**

* **Programming Language**: Python
* **Libraries**:
  + ‘re’ for regular expressions to parse logical expressions.
  + Custom data structures for representing parse trees.
* **Techniques**: Recursive descent parsing, application of logical equivalences.

**4. Implementation**

**4.1 System Architecture**

The system consists of the following components:

* **Parser**: Converts the input expression into a parse tree.
* **Transformer**: Applies logical transformations to convert the expression to DNF.
* **Evaluator**: Validates the correctness of the transformation.

**4.2 Code Examples**

**Example Code Snippet**:

:- op(500, yfx, and).

:- op(500, yfx, or).

distribute(A and (B or C), (D1 or D2)) :-

distribute(A and B, D1),

distribute(A and C, D2).

distribute((A or B) and C, (D1 or D2)) :-

distribute(A and C, D1),

distribute(B and C, D2).

distribute(A and B, A1 and B1) :-

distribute(A, A1),

distribute(B, B1).

distribute(A or B, A1 or B1) :-

distribute(A, A1),

distribute(B, B1).

distribute(A, A) :- atom(A).

to\_dnf(Expr, DNF) :-

distribute(Expr, DNF).

example :-

Expr1 = ((a and b) or (a and c)),

to\_dnf(Expr1, DNF1),

write('DNF of '), write(Expr1), write(' is '), writeln(DNF1),

Expr2 = ((x or y) and z),

to\_dnf(Expr2, DNF2),

write('DNF of '), write(Expr2), write(' is '), writeln(DNF2),

Expr3 = ((p and q) or (r and s)),

to\_dnf(Expr3, DNF3),

write('DNF of '), write(Expr3), write(' is '), writeln(DNF3).

:- initialization(example).

**5. Results**

**5.1 Test Cases**

**Test Case 1:**

* **Input**: ((x or y) and z)
* **Expected output :** DNF of x or y and z is x and z or (y and z)
* **Actual output:** DNF of x or y and z is x and z or (y and z)

**Test Case 2**:

* **Input:** ((p and q) or (r and s))
* **Expected output:** DNF of p and q or (r and s) is p and q or (r and s)
* **Actual output** : DNF of p and q or (r and s) is p and q or (r and s)

**5.2 Analysis**

The results indicate that the implementation correctly transforms the given logical expressions into their DNF equivalents. The test cases validate both the correctness and performance of the algorithm. The implementation meets the objective of simplifying logical expressions and enhances computational efficiency.

**6. Discussion**

* 1. **Challenges**
* **Complexity:** Converting formulas to DNF can lead to an exponential blowup in the size of the formula, making computations resource-intensive.
* **Atomic Components:** Each variable, negation, conjunction, and disjunction must be represented distinctly while maintaining readability and manageability..

**6.2 Future Work**

* **Optimization**: Improve the efficiency of the transformation algorithm.
* **Extended Support**: Handle more complex logical operators and expressions.
* **User Interface**: Develop a graphical interface for easier input and visualization of logical expressions.

**7. Conclusion**

The project successfully developed a program to convert propositional logic expressions into Disjunctive Normal Form (DNF). The approach involved parsing the expression, eliminating implications, moving negations inward, and distributing disjunctions over conjunctions. The results demonstrated the correctness and efficiency of the implementation, highlighting the significance of DNF in simplifying logical expressions and enhancing computational tasks. Further improvements can enhance the algorithm's performance and extend its capabilities.