Logical Expression

Shervin Emrani

February 2, 2024

Understanding the Code

This document provides an overview and explanation of the Python code utilizing the `sympy` library to manipulate propositional logic expressions.

1 Code Overview

The provided code includes functions for simplifying logical expressions, converting expressions to Conjunctive Normal Form (CNF), checking if an expression is well-formed, and converting expressions to and from LaTeX notation.

1.1 Functions

1.1.1 `simpify_logic`

This function simplifies logical expressions, providing options for converting them to CNF or DNF. It also handles the standardization of constants and variables.

1.1.2 `to_cnf_`

This function converts a logical expression to CNF, eliminating implications and distributing conjunctions over disjunctions if needed.

1.1.3 `new_expression`

This function converts LaTeX notation to Python-friendly logical operators, allowing users to input expressions using LaTeX symbols.

1.1.4 `show_latex_expression`

This function converts Python-friendly logical operators back to LaTeX notation for displaying expressions.

1.1.5 `is_well_formed`

This function checks whether a logical expression is well-formed by attempting to simplify it using the `simplify_logic` function.

1.1.6 `main`

The main part of the code takes user input in LaTeX format, checks if the expression is well-formed, and then converts it to CNF using the `to_cnf` function.

2 Details of Code Execution

Let's delve into the code execution process to better understand how each function contributes to the overall functionality.

2.1 `simpify_logic` Function

The `simpify_logic` function first checks if a specific form (CNF or DNF) is requested. If the expression is already in the correct form and contains only literals, it returns the expression as is. The function then attempts to simplify the expression by replacing relational variables with simplified versions. If the expression is a Boolean function, it proceeds with the simplification process.

The function then replaces relational variables with Dummy variables and ensures that constants are standardized to 1 or 0. It generates truth tables for the variables and constructs the final expression either in SOP (Sum of Products) form for DNF or POS (Product of Sums) form for CNF.

2.2 `to_cnf_` Function

The `to_cnf_` function converts a logical expression to CNF. It first checks if the expression is already in CNF and returns it unchanged if true. Next, it eliminates implications and distributes conjunctions over disjunctions. The resulting expression is then returned.

2.3 Expression Conversion Functions

The `new_expression` and `show_latex_expression` functions are utility functions for converting expressions between LaTeX notation and Python-friendly notation. They replace specific symbols to facilitate user-friendly input and output.

2.4 `is_well_formed` Function

The `is_well_formed` function checks if a logical expression is well-formed. It attempts to simplify the expression using the `simplify_logic` function and returns `True` if successful, indicating a well-formed expression.

2.5 `main` Section

The main part of the code takes user input in LaTeX format, checks if the expression is well-formed using `is_well_formed`, and then converts it to CNF using the `to_cnf` function. The final CNF expression is then printed.