ASHESI UNIVERSITY

# DISCRETE PROJECT WORK: SYSTEM SPECIFICATION


GROUP MEMBERS

1. MIRIAM GALE WEPIYA
2. JAMES ASIAMAH-YEBOAH
3. MARTHA NAA OKAILEY AFFUL


CS221: DISCRETE MATHEMATICS

LECTURER: DR. AYAWOA DAGBOVIE

29TH NOVEMBER 2025

# INTRODUCTION

Propositional logic is a branch of discrete mathematics and computer science that focuses on the study of declarative statements that are either true or false, and these statements are combined using logical connectives. In different fields like software engineering and system specification, it is important to translate natural language statements into symbolic forms to analyse their logical structure and truth values. Humans, however, sometimes find it difficult to convert complex natural language sentences into symbolic notation. This project aims to address this but translating English sentences containing logical connectives. This is important because automating the process will reduce human errors.

This report is based on a Python-based application that uses a Graphical User Interface (GUI) to accept natural language as input and produce its symbolic equivalent, along with a legend of defined variables.

# METHODS

The solution was implemented using the Python programming language. The logic relies on string manipulation and recursive function calls to parse natural language. The user interface was built using the standard Tkinter library.

## INITIALISATION

To manage the translation process, the system maintains two global variables to track atomic propositions. An atomic proposition is a statement that cannot be broken down further, that is, it contains no connectives. The `symbol_table` is a dictionary that stores the proposition statement and its assigned variable (e.g. p). The `symbol_counter` ensures that the other propositions that will be assigned letters will continue from ‘p’, then to ‘q’ and so on.

## TEXT CONVERSION

Before any conversion is done, the input must be standardised to ensure consistency. The `normalise` function handles this by converting the text to lowercase and removing the excess whitespace. This ensures that certain inputs like “It is Raining” and “it is raining” are treated identically.

## ASSIGNING SYMBOLS

The `assign_symbol` function is responsible for managing the `symbol_table`. When an atomic proposition is identified, this function checks if it has already been encountered. It is new, it assigns the next letter available and increments the counter. If it already exists, it returns the previously assigned letters.

## RECURSIVE TRANSLATION LOGIC

The application of the `translate` function. It uses a recursive approach to handle nested logical statements. The function first standardises the proposition using the normalise function and then checks for logical connectives in a specific order of precedence. From IF-THEN, to AND, OR and NOT. For example, the handling of implications checks for both standard syntax (IF-THEN) and comma syntax. Similarly, conjunctions and disjunctions split the sentences and recursively translate the surrounding parts, wrapping the results in parentheses.

Finally, if there are no binary connectives, it checks for negation. The current implementation looks for the specific substring, “not”. If found, it removes “not” to extract the positive form of the proposition, gets the symbol and substitutes it.

## GRAPHICAL USER INTERFACE (GUI)

The GUI allows users to easily interact with the translator app. It provides text areas for input, symbolic output and a legend showing variable assignments. There are buttons like “Translate” and “Clear All.” The PEFORM_TRANSLATION function bridges the GUI and the backend logic.

# RESULTS

The application successfully translates natural language sentences involving standard connectives into symbolic logic.

The system correctly handles simple compound statements. Figure 1 shows the translation of a sentence containing a conjunction and a negation. The recursive nature of the algorithm allows it to handle complex nested sentences, automatically applying parentheses to resolve any ambiguity based on the implemented precedence order. Figure 2 demonstrates an implication where the antecedent contains a disjunction.

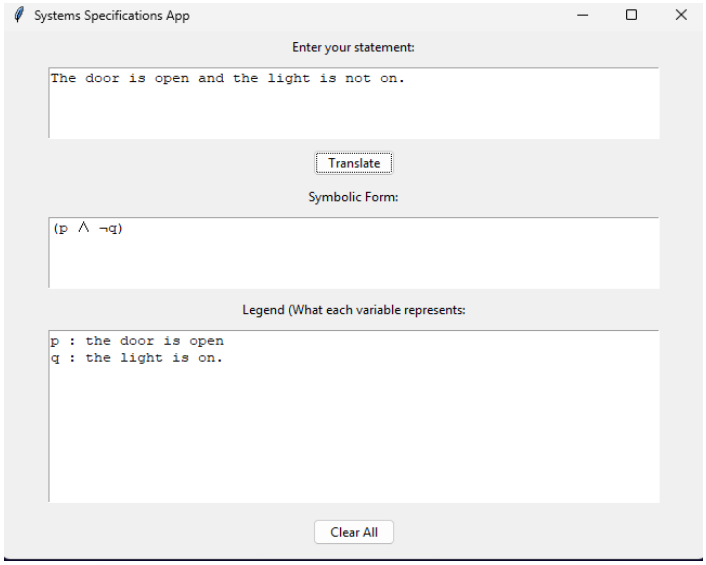


*Figure 1*

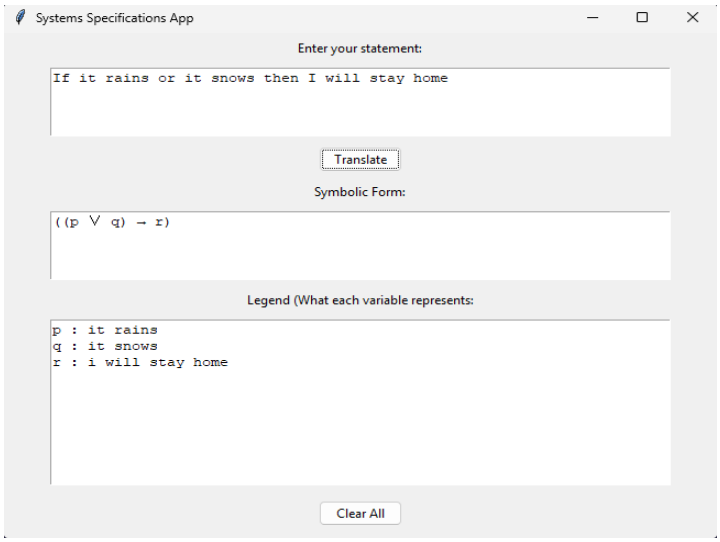


*Figure 2*

The clear all button successfully resets the application state, allowing the user to start a fresh translation without restarting the program. This ensures that the subsequent translations start labelling variables from ‘p’ again.

# DISCUSSIONS

The Python application successfully addresses the problem of automating the translation from natural language to symbolic logic for propositional statements. that results indicate that the recursive approach, along with the string manipulation, is an effective method for passing sentences built with standard connectives. A key insight gained from the development is the importance of establishing operator precedence within the logic. The order in which the TRANSLATE function checks for connectives is based on the structure of the statement. By checking for “IF-THEN before “AND” or “OR”, the system correctly interprets sentences like “If p then q and r” as a $p \rightarrow (q \wedge r)$, matching the logical conventions.

Despite our ability to convert statements into symbols, the system has limitations. The negation detection in the code currently detects the exact substring “not”. It would fail to correctly translate contractions like “isn’t” or prefixes like “un-happy”. Furthermore, the system is limited to propositional logic and cannot predicate logic involving quantifiers like “all”, “some” or “every”.

Possible upgrades could be the generation of a truth table and giving error feedback when the user types a wrong sentence structure.

# References

Rosen, K. H. (2012). Discrete Mathematics and its Applications (7th ed.). New York, NY: Mcgraw-Hill.

Link for the code: https://github.com/MaNOk06/System-s-Specification.git