TT4SPL

Ahmed Ghoor, Ahmed Khan, Mthokozeleni Sithole

Abstract

An educational tool that provides a user-interface that allows a user to type in valid sentences of propositional logic, and produces a complete, labelled truth table for any number of valid input sentences of propositional logic, allowing for the checking of logical properties such as truth-functional truth and truth-functional falsity. This project was implemented for multiple platforms, including a Web, Android, iOS, MacOS, Linux and Windows implementation for accessibility.

1. Introduction

The overall purpose of this project was to create an accessible and scalable educational tool to help users reason about structured information. This might be for philosophy students; students learning knowledge representation and reasoning or even to form the basis for a program that analyses natural language arguments. Reasoning about structured information is also useful for generating inferences in Artificial Intelligence.

Specifically, for this project, the purpose of this tool was to be able to produce a complete, labelled truth table for any input sentence of **propositional logic**, allowing for the checking of logical properties

"In propositional logic, symbols are used to represent statements about the world. For example, the English sentence "it is rainy or it is sunny" can be represented in propositional logic as "r V s" and the English sentence "if it is rainy then it is cold and windy" can be represented in propositional logic as "$r \rightarrow c \wedge w$". Sentences of propositional logic are defined recursively using the following five Boolean connectives: negation, conjunction, disjunction, material implication and material equivalence. Each sentence has a logical value: True or False. A truth table is used to record the truth value of a sentence on each truth value assignment of the literals in a sentence."

We, hence, applied all possible interpretations for each sentence of propositional logic by mapping all the statements in a sentence to different truth values and processing the result, until we exhausted all possible combinations. There was 2^n combinations for each sentence, where n is the number of statements in a sentence.

The project allows users to test the various applicable logical properties for each sentence or collection of sentences, such as truth-functional truth, consistency, equivalence, validity and entailment.

Lastly, since the aim of educational tools should always be accessibility and scalability, various design considerations were made with this in mind. Examples include coding the program for multiple platforms and taking in input that suits the needs of both beginner and advanced students.

2. Requirements Captured

2.1 Functional Requirements

- Input sentences of propositional logic - System must be able to read in and understand sentences of propositional logic entered by the user.
- Output truth table - Truth table must be outputted for the given sentence(s) of propositional logic
- Check for logical properties – System must allow the user to check truth tables for logical properties such as truth-functional truth, falsity, indeterminate, consistency, and equivalence, entailment, and validity.
- Help button – Users should be able to click the help button in order to be presented with a guide to assist them in using the system
- Graphical User interface – System must provide user with a GUI in order for them to present input and output

2.2 Non-Functional Requirements

- Accessibility: Since it is an educational tool, it should be, accessible to everyone, not matter their operating system or internet stability.
- Scalability: It should be easy to expand on the implementation should new requirement arise.
- Input buttons – System should provide user with a set of buttons to assist them with the task of entering input
- Input size – The system should be able to accept input for any number of logical sentences and still produce an accurate truth table
- Error Mitigation - System must strive to prevent errors from occurring.

2.3 Usability Requirements

- Simple and Easy GUI – GUI should be simple, attractive, and easy to understand for any users of the system.
- Learnability – System should be easy for new users to learn to use. Help menu should assist with any issues users may face
- Error Tolerance – System should be able to effectively handle all errors which may arise in a way that it will affect the user experience in the least possible way

2.4 Use Cases

2.4.1 Open Help Menu

Actors: User

A new User opening the program for the first time may not understand how to use it. If the User clicks on the “Help” button, the help menu will be displayed. The help menu consists of an exhaustive description detailing the usage of all functionality provided by the system.

When the User is ready to return to the app, they can click on the “back” button.

2.4.2 View Truth Table for sentence of propositional logic

Actors: User

When a User has added a valid sentence of propositional logic into the system, the User will have an opportunity to add another. The user will be able to assign meta data to each of these sentences, before adding them, to indicate the type of sentence it is.

When the User clicks the run button, the program will need to process the input to begin calculations to producing the output. The system will be able to recognize logic symbols such as V or Λ, as well as their English counterparts such as “or” or “and”.

The system will be able to calculate the truth table corresponding to the given input. This truth table will be labelled and displayed on the GUI for the user to review.

2.4.3 Check for logical properties

Actors: User

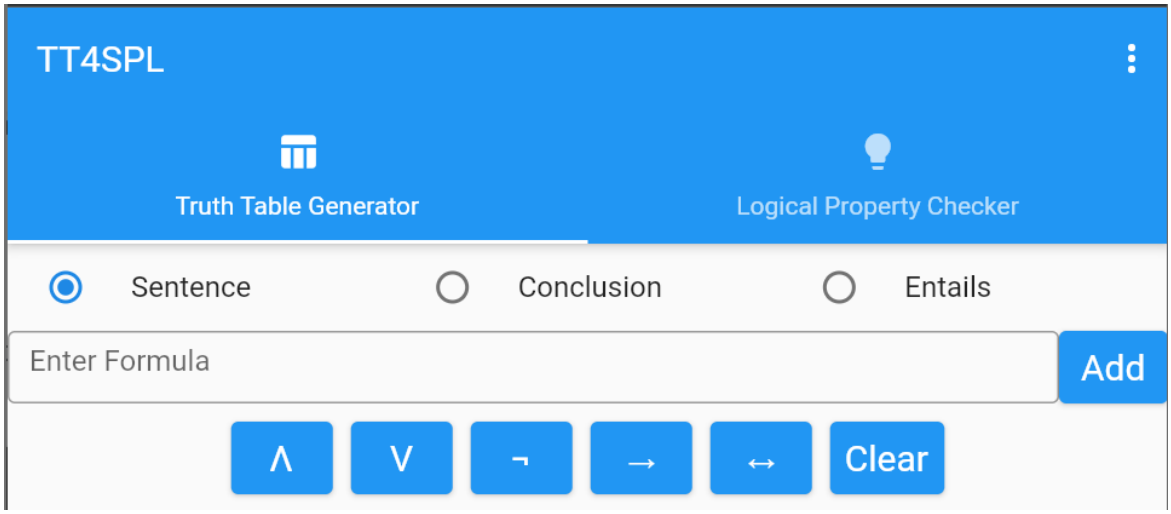
The system will allow them the option of checking for various logical properties such as truth – functional truth, truth functional falsity, truth functional consistency, or truth functional equivalence. The User will be able to enter a sentence or a set of sentences, and is then given the option to choose which logical check they would like to run, and the system will return an answer.

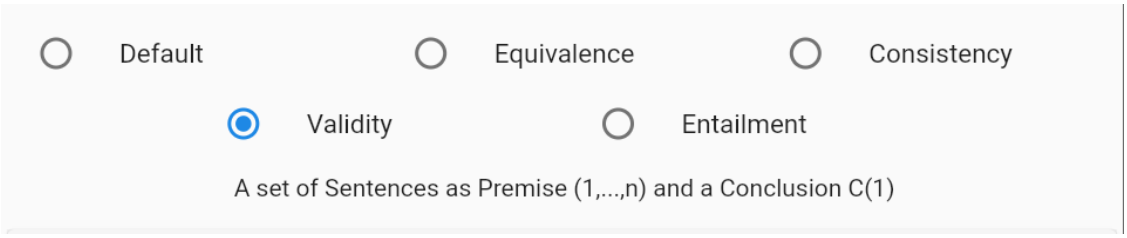
3. Design overview and Overall architecture

The user interface has 2 tabs; each tab is allocated to a main use case. The 1st tab which the user lands on is the Truth Table Generator tab. It is used for the 1st main functionality of producing truth tables. The 2nd tab is for checking logical properties which is the 2nd main use case of the program. There is also a help button on the top right corner for the user to read how the program is used.

On the top half of both the tabs there is a combination of components/ widgets (Radio buttons, a text field, an Add button, Connectives buttons and Clear button) that aid the user in inputting a sentence of propositional logic. On the 2nd tab (Logical Property Checker) extra components are added for the user to select which property they would like to check. Based on the property the user wishes to check feedback is provided on how the format of input should be.

At the bottom right hand, there is a floating run button. Once the users have inputted their set of sentences, and press the run button, a truth table is displayed on the Truth Table Tab, and the logical property checks are produced on the Logical Property Checker Tab.





4. Implementation

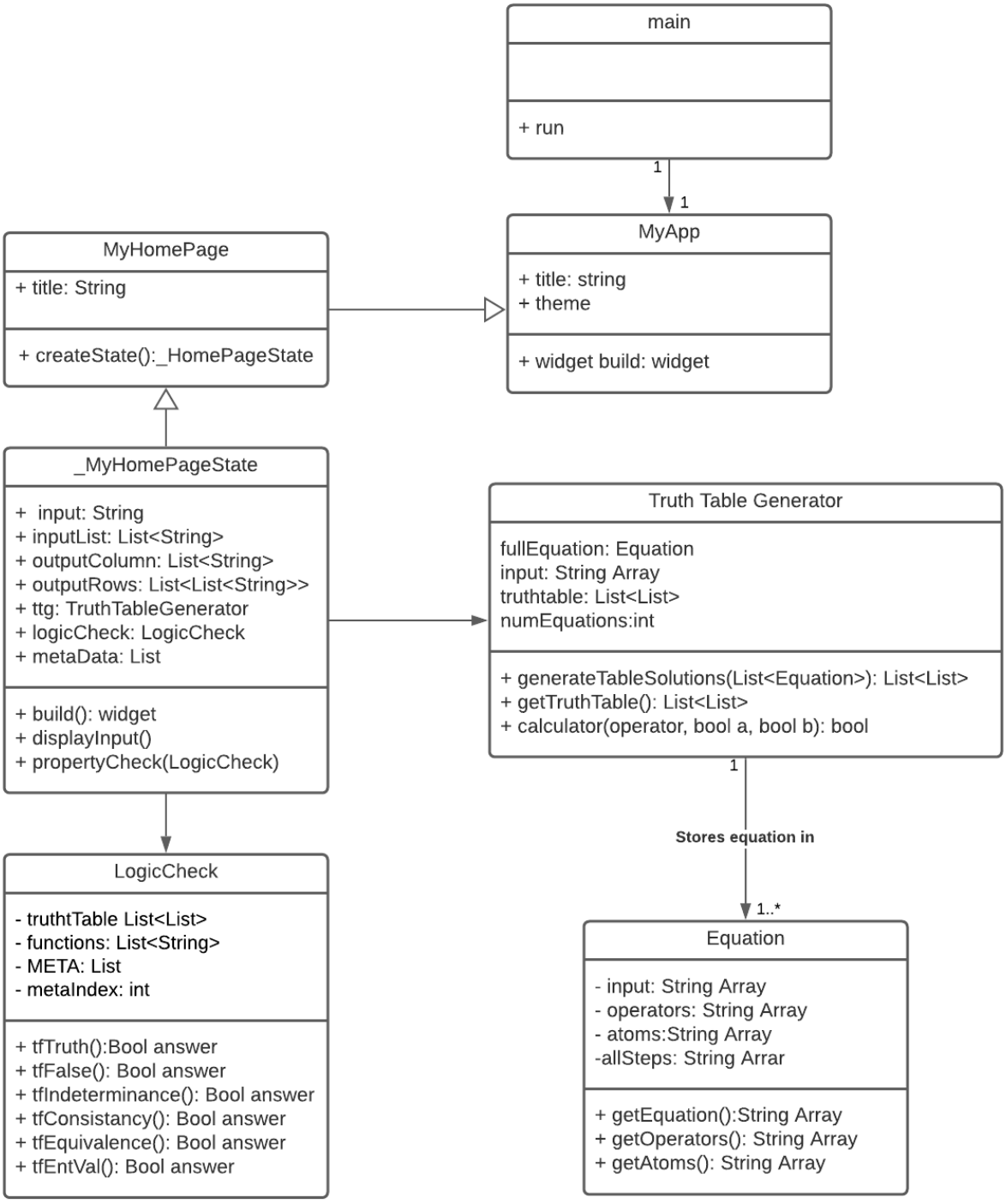
The program had 3 main sections:

The User Interface section which needed to take in user input, convert it and send it to the Truth Table Generator and Logical Checker before displaying the feedback.

The Truth Table Generator section which takes in a 1D input array from the User Interface Section and outputs a 2D Array (Table).

The Logical Property Checker section which takes in a table from the Truth Table Generator section and some meta data from the User Interface, and outputs the results.

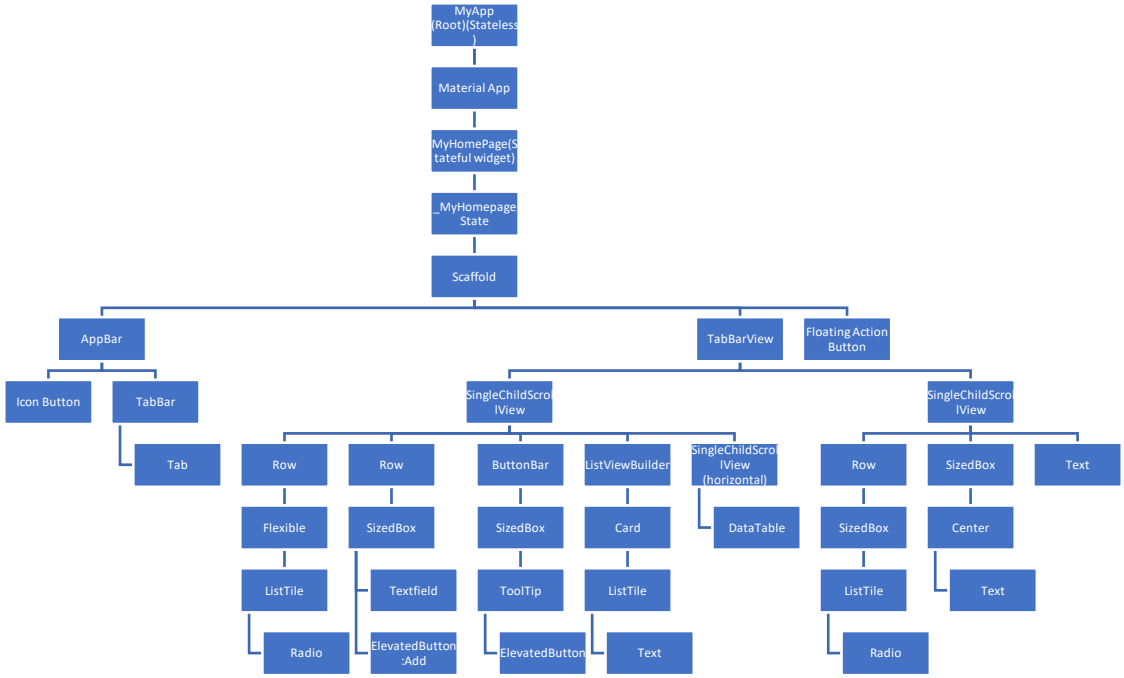
4.1 Class Diagram



4.2 User Interface implementation.

The user interface is implemented on the MyApp class. The MyApp class creates the components of the User Interface called widgets, defines how they should function and is responsible for the communication of the components within the user interface.

Flutter uses a concept of widgets; a widget is a basic building block of a flutter user interface. “Widgets describe what their view should look like given their current configuration and state”. The widgets form a hierarchy of parent and children based on how the programmer adds them. As a result, each widget is nested within a parent widget which gives it context, and the hierarchy is maintained up to the root widget and is called a widget tree.



This is the widget tree our user interface. The 2nd SingleChildScrollView contains all widget subtrees of the 1st Except DataTable with a horizontal SingleChildScrollView as a parent because user input features are consistent throughout the 2 tabs. The subtrees are not included in the diagram because repeating them would reduce readability.

At the lowest level of the hierarchy are widgets that are visible in the user interface and they are nested within parent widgets which are Containers. Containers control layout behaviour, like size, position and padding.

Data structures of UI class

- A List of String is used to store input from the textfield, I choose a list because it a simple data structure for traversing through elements to perform operations necessary to prepare the input to be used by the TruthTableGenerator class. A nested List of String is also used to store data from the TruthTableGenerator once the table is produced. The nested List was chosen because it stores data in form of rows and columns which is aligned to how it will be represented using a DataTable widget to produce the truth table.
- An enumeration is used to store states of a group of radio buttons which is important for implementing the functionality of widgets that depend on the state of radio buttons

Discuss the function of the most significant methods

- The most important method in the user interface (My App class) is the build() method. The widget build() method is called when a widget is added to the widget tree or when it's dependencies change. Every widget has a build method which can be called using the setState() method to update a widget and its subtree but method used to update the user interface is the build() method of the _MyHomepageState widget/class. This is because the _MyHomePageState class is responsible for state maintenance of the user interface and updating all other widgets under its subtree accordingly.

A description of the UI class relationship with other classes.

- The MyApp class accepts user input and passes it to the TruthTableGenerator class which generates a truth table then sends it back to My App which is then passed to the LogicCheck class which conducts the required property check the sends a string response which is displayed with the truth table.

4.3 Truth Table Generator

The TruthTableGenerator class performs the required calculations on the processed input, in order to return a truth table as a 2D Array. Upon receiving the input, the Truth Table Generator stores the input in an Equation Object which separates processes the input to make manipulation easier.

Some of the key methods in the Truth Table Generator Class are:

List<List> **generateTableSolutions**(List<Equation>)

/// Generates and returns a completed table with solutions for a list of equations.

List<List> **generateTable**(equation)

/// Generates and returns a table for a single equation.

List<bool> **generateSolution**(table, equation)

/// Generates the solution values for a given table

bool **calculator**(String operator, bool a, bool b)

/// Returns solution for and, or, implies, and equals gates

The approach taken was to try and have small methods that each do a small task, and then to have other methods simply calling those methods. The input is separated and stored as a list of Equations objects. It is then passed into the generateTableSolutions method which calls the generateTable() and generateSolution() methods for each equation, which themselves call other methods, before combining all solutions into a single table. generateSolution() calls the calculator() method several tims.

4.4 Equation

The Equation class is used to encapsulate the attributes of each sentence of propositional logic which gets entered into the system. An Equation object is used to calculate the truth table.

It seperates the atoms, operators and intermediate calculation steps and provides methods to call and edit these eg:

List<String> **getInput**()

/// returns input array

List<String> **getAtoms**()

/// returns all the atoms in the equation

List<String> **getOperators**()

/// returns list operators in input

List<String> **getAllSteps**()

/// returns array with all the steps to reach the solution

4.5 LogicCheck

The LogicCheck class was used to conduct the necessary logical checks required for this project. This class is instantiated by the MyApp class which feeds it a generated truth table stored in a List. The LogicCheck class then uses its methods to examine the given truth table in order to check for truth functional truth, falsity, indeterminacy, equivalence, consistency, and validity.

Data Structures used in LogicCheck class

- The predominant data structure used in this class is a dart List. Throughout the program, Lists are used to store the values of generated truth tables, as well as any metadata such as entailment or conclusion sentences which may be associated with any given truth table. This class receives these Lists as inputs from the MyApp class, and uses them to conduct the logical property checks

Important methods

The primary methods used by this class pertain to each of the respective logical checks. Each of these methods return boolean values indicating whether the test has passed or failed. Each of these methods are outlined below. Note that each of these methods are partnered with a respective toString() method which is used by the MyApp class to display the results on the GUI.

bool **tfTruth**(List<List> tt1, int num)

/// This method takes in a List containing truth table values, and checks it for truth-functional Truth. The integer “num” received by this method indicates the index of the target sentence, within the List, upon which this operation is to be conducted.

bool **tfFalse**(List<List> tt1, int num)

/// This method takes in a List containing truth table values, and checks it for truth-functional Falsity. The integer “num” received by this method indicates the index of the target sentence, within the List, upon which this operation is to be conducted.

bool **tfIndeterminacy**(List<List> tt1, int num)

/// This method uses the outputs of the tfTruth() and tfFalse() methods to determine if a sentence is truth-functionally indeterminant. The integer “num” received by this method indicates the index of the target sentence, within the List, upon which this operation is to be conducted.

bool **tfEquivalence**(List<List> tt1)

/// This method takes in a List containing truth table values, and compares the sentences within it for truth-functional equivalence.

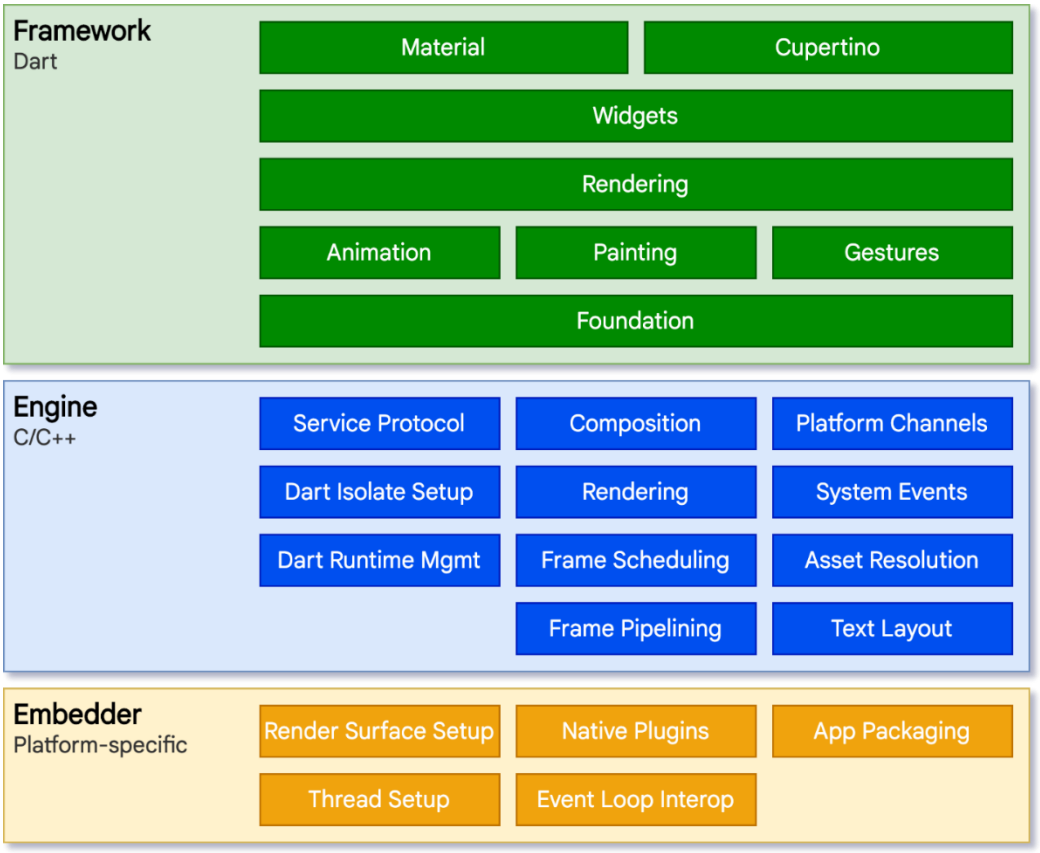
bool **tfConsistant**(List<List> tt1)

/// This method takes in a List containing truth table values, and compares the sentences within it for truth-functional consistency.

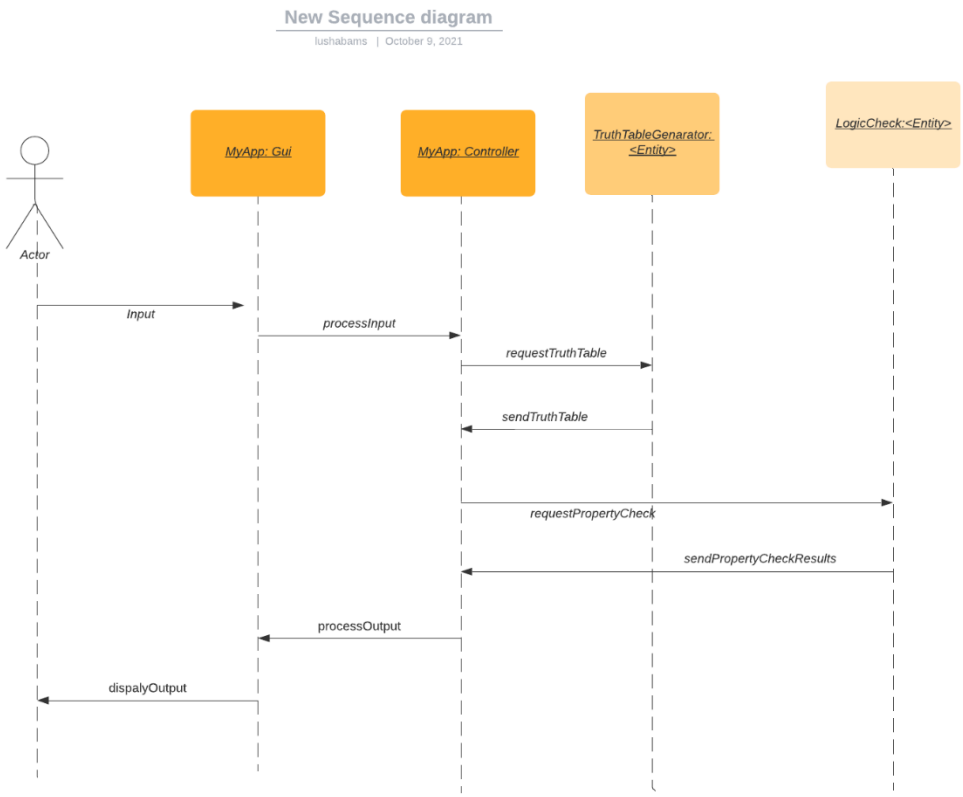
bool **tfEntVal**(List<List> tt)

/// This method takes in a List containing truth table values, and uses it to check for either truth-functional entailment, or truth-functional validity, depending on the context within which it is called.

4.6 Flutter Architectural layers



4.7 Sequence Diagram



5. Program Validation & Verification

5.1 Summary Testing Plan

Process	Technique
Class Testing: test methods and state behaviour of classes	All methods were tested individually, using White Box Unit Tests. [See earlier versions of code on GitLab before code clean-up]
Integration Testing: test methods and state behaviour of classes	Random and Behavioural Testing This program had 3 main sections that needed to integrate with each other: - The User Interface section which needed to take in user input, convert it and send it to the Truth Table Generator and Logical Checker before displaying the feedback. The functionality of each feature on the user interface was tested after implentation - The Truth Table Generator which took in a 1D input array from the User Interface Section and outputted a 2D Array (Table). - The Logical Checker which took in the table from the Truth Table generator and some meta data from the User Interface, and outputted the results. The behavior of each of these sections were first tested separately with a range of expected input and output from the other sections to ensure smooth integration. [See the main method in Truth Table Generator Class for some examples]
Validation Testing: test whether customer requirements are satisfied	Use-case based black box and Acceptance tests Had meetings with the client to get Acceptance on design and approach. Had random students try to use the program and give their feedback. Decided to used two tabs instead of one based on student feedback.
System Testing: test the behaviour of the system as part of a larger environment	Performance Stress tests were conducted to ensure that the program could handle very large and complex inputs.

5.2 Key Tests

Section	Test [Data example]	Result
Truth Table	Poduce a Truth Table [A or B]	Passed
Truth Table	Words instead of letters [Love or War implies Fair]	Passed
Truth Table	Order of operations [A or B and C]	Passed
Truth Table	Double negation [not not A]	Passed

Truth Table	Repeated atoms [A or B and A]	Passed
Truth Table	Brackets [A and (B or C)]	Passed
Truth Table	Nested brackets [(A or ((B and C) implies D)) equals E]	Passed
Truth Table	Multiple equations [A or B ; A and B]	Passed
Truth Table	Multiple equations of different atoms [A or B ; C and D]	Passed
Truth Table	Multiple equations of different length [A or B and C; D and C]	Passed
Logic Check	Truth Functional Truth/Falsity/Indeterminate [Any sentence]	Passed
Logic Check	Truth Functional Equivalence [Any two sentences]	Passed
Logic Check	Truth Functional Consistency [Any set of sentences]	Passed
Logic Check	Truth Functional Validity [Any set of premises with a conclusion]	Passed
Logic Check	Truth Functional Entailment [A knowledgebase with an entails]	Passed
GUI	User enters multiple sentences of various sizes	Passed
GUI	Display Truth Table of different sizes	Passed
GUI	Button functionality (behavior of buttons)	Passed
GUI	Input is consistent between tabs	Passed
GUI	RadioButton behavior	Passed

6. Discussion of results

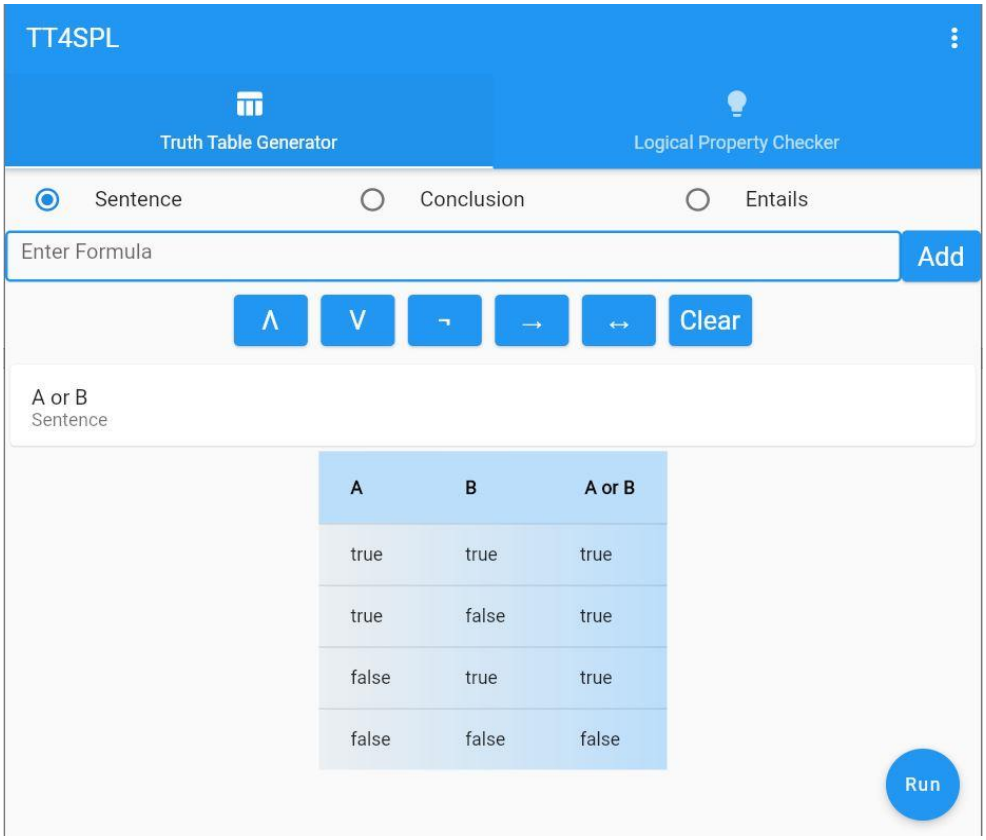
Below we will discuss the level to which our final program was able to meet the requirements which we had set out

6.1 Truth Tables

Users have the ability to input any number of valid sentences of propositional logic (logic formulas) and output a complete, accurate, and labelled truth table on all combinations of truth value assignments of the statements/atoms in a sentence

Users are able to type sentences into the input bar using keywords such as “and” or “or”, but on top of this they are also able to enter their input using a set of buttons which we have provided for them.

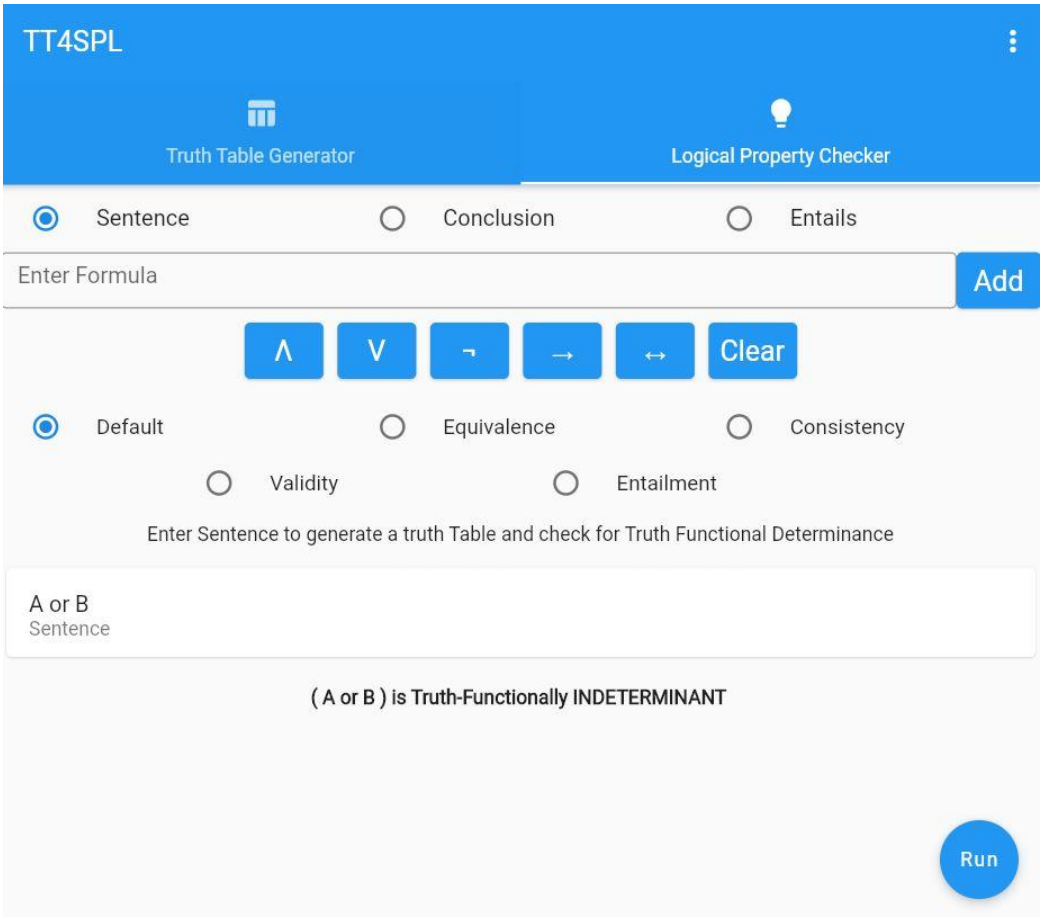
The program passed all the Truth Table test cases provided in the Demo Test Data Set. It all passed all 17 test cases that the developers created to test various levels of input complexity and boundary cases. [See main method of the TruthTableGenerator class]



6.2 Logical Properties

Users are able to input a set of sentences and check for any applicable logical properties. Users can check for truth-functional truth, truth-functional falsity, truth-functional indeterminate, truth-functional consistency, and and truth functional equivalence.

The program passed all, except one, of the Logic Checks provided in the Demo Test Data Set. There was nothing particularly unusual about the test case that failed that the program has not handled with other examples. Hence, it did not indicate an input complexity limit. Other than that particular example, the program should be able to process any logical check.

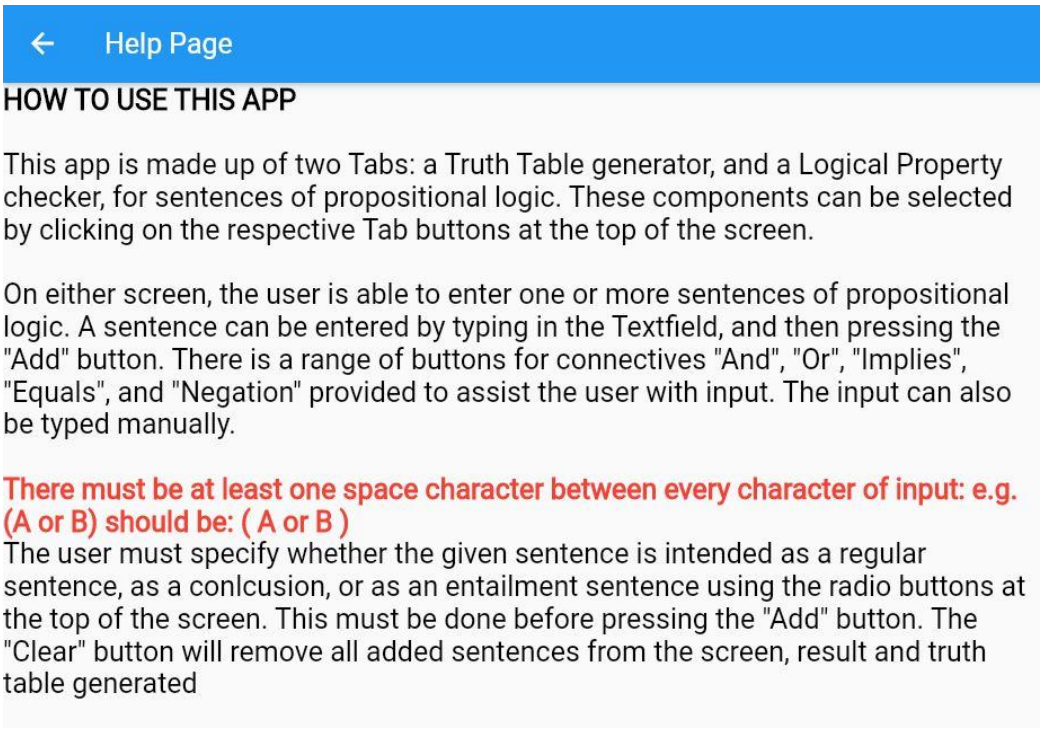


6.3 Help for New Users

We implemented a help menu which would provide the user with an exhaustive description on how to use each aspect of the application. However, since the goal of modern application design seeks to be intuitive enough to not need manuals, the program has a few other features.

The program provides text hints which appear when hovering the curser over any respective logical symbol. The allows students new to logic to understand what they mean without having to go to the help menu.

When the User clicks on a logical check, the program gives a short description of what type of sentences the User is required to input, if the user wants to run that particular check.



6.4 Simple user–friendly GUI to cater for all levels of students

“The Golden Rules of Design” were also used extensively on the front end:

The program provides a simple user interface that allows both beginner and advanced students to easily type in the valid sentences of propositional logic and generate their required output. It does this by having the program take in input such as “and” and “implies”, instead of symbols, and allowing users to use words such as “rain” as atoms instead of only letters.

This was done with the intention of making the input more intuitive for beginner users, while also providing buttons with symbols as a shortcut for more advanced students.

6.5 Error Mitigation

The program allows users to edit any sentence that they have already added to their set of sentences, if a mistake was made.

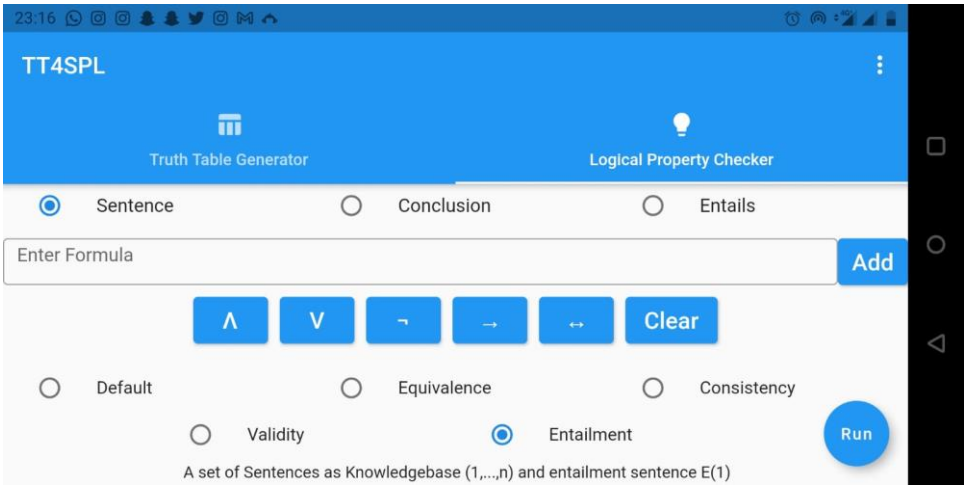
The program automatically removes unintentional extra white spaces in the input that could break the program.

The program provides hints based on the property being checked, as explained above under “Help for new Users”.

However, there is room to improve here. If the User still manages to input something invalid, the program does not provide any feedback.

6.6 Accessibility

The program has been implemented for multiple platforms so that no student is disadvantaged due to not having a specific operating system, access to the internet or a smart phone/pc. This was achieved by coding the program in Dart, using Flutter. Flutter allows a program to be built for multiple platforms, including Web, Android, iOS, Windows, Linux, and MacOS, using a single code base.



6.7 Scalability

Flutter provides an environment for easy scalability, as any updates required for all platforms only need to be made on one code base.

The program can take in words instead of letter for atoms, and take in words such as “and” and “implies” instead of symbols. This provides a good foundation to extend the program to take in natural language arguments, as the program can already read input that is very close to natural language.

7. References

The Logic Book 5th Edition Bergmann-Moor-Nelson
Author : flutter support team https://flutter.dev/docs
Author: Tutorialspoint date:N/A https://www.tutorialspoint.com/flutter/flutter_tutorial.pdf (October 2021)
Author:geeksforgeeks date: 23 Feb, 2021 https://www.geeksforgeeks.org/flutter-tutorial/(October 2021)

8. User Manual

8.1 How to run app

Standard:

1. Open program location: TT4SPL/TTG3/truth_table_generator_3
2. Run command "flutter run lib/main.dart"

Live Web Application (using github pages): URL: https://thaddeusowl.github.io./

Linux/Windows/MacOS Desktop Application:

1. Open program location: TT4SPL/TTG3/truth_table_generator_3
2. Run command "flutter run -d windows/linux/macos"

Android Application:

1. Copy APK file to Android device (/build/app/outputs/apk/release).
2. Open on device to install.

iOS: Source files stored in iOS folder. Just needs a MacOS device and Xcode to build for release

8.2 How to use app

This app is made up of two Tabs: The Truth Table generation tab, and a Logical Property checker tab, for sentences of propositional logic. These components can be selected by clicking on the respective Tab buttons at the top of the screen.

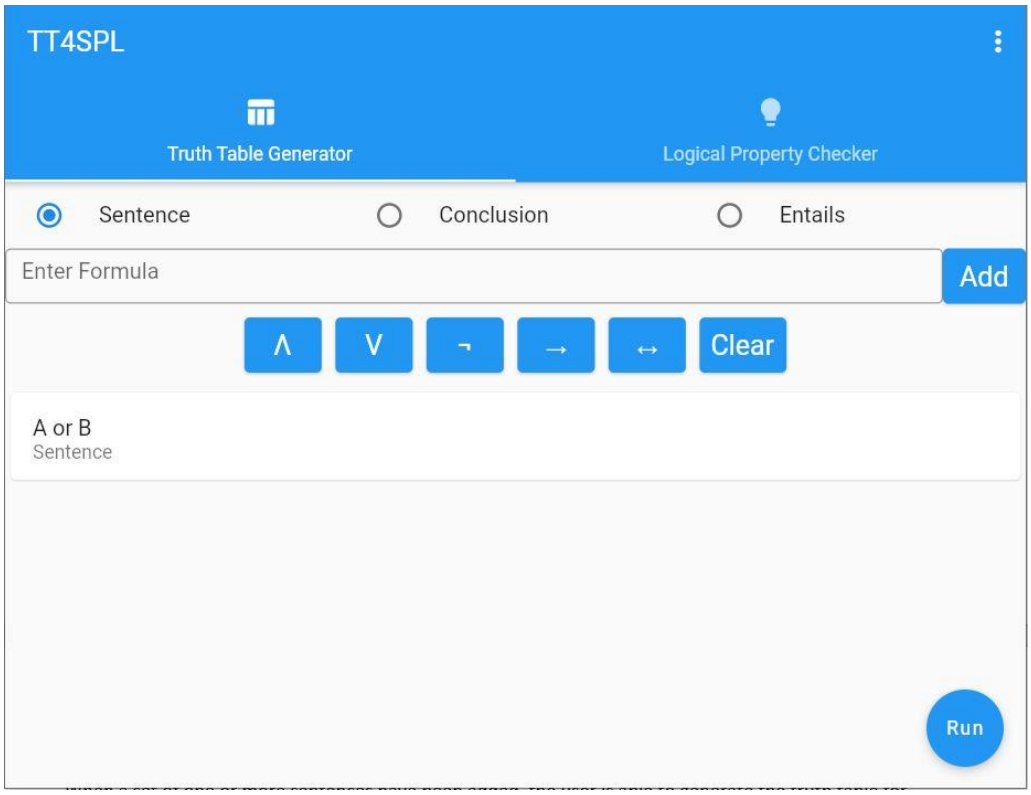
On either screen, the user is able to enter one or more sentences of propositional logic. A sentence can be entered by typing in the Text field, and then pressing the "Add" button. There is a range of buttons for connectives "Conjunction" (And), "Disjunction" (Or), "Implication" (Implies), "Equivalence" (Equals), and "Negation" (Not) provided to assist the user with input. The input can also be typed manually.

There must be at least one space character between every item of input: e.g. (A or B) should be: (A or B).

Atoms can be represented as either letters, or words: eg. (R or S), or (Rainy or Sunny).

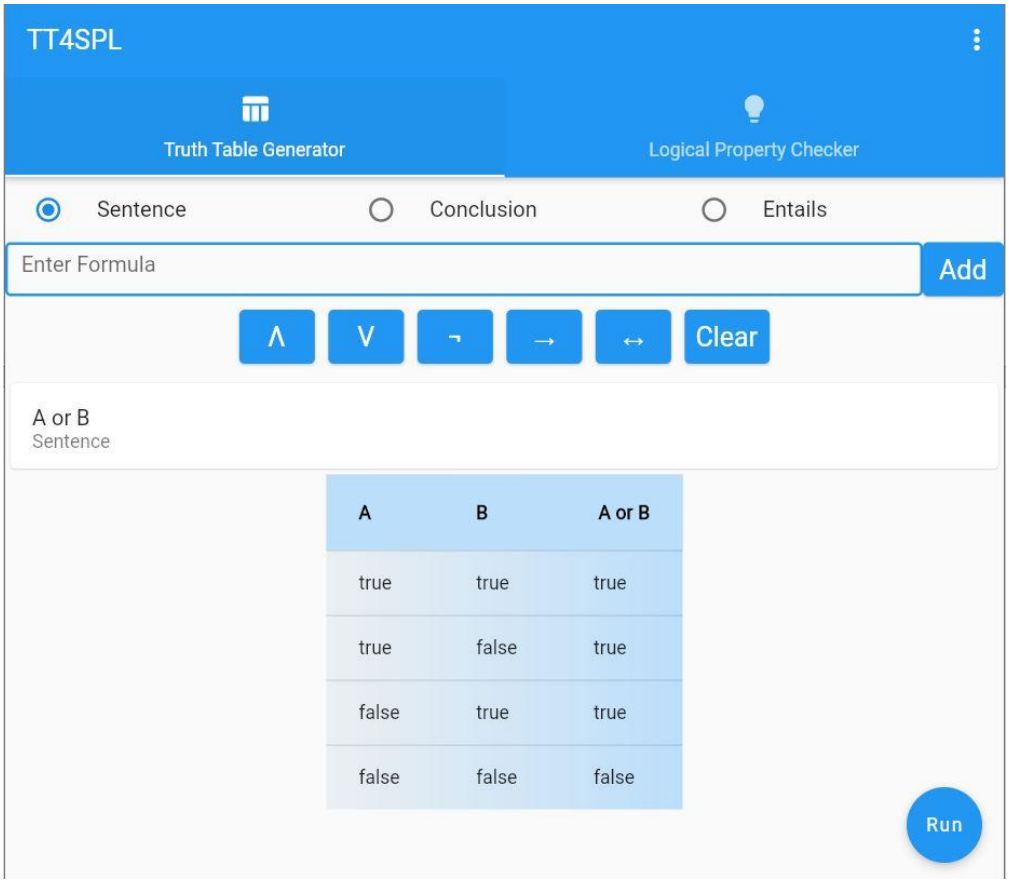
The user must specify whether the given sentence is intended as a regular sentence, as a conclusion, or as an entailment sentence using the radio buttons at the top of the screen. This must be done before pressing the "Add" button. The "Clear" button will remove all added sentences from the screen, result and truth table generated.

Once added, a sentence can be edited or removed by clicking on it.



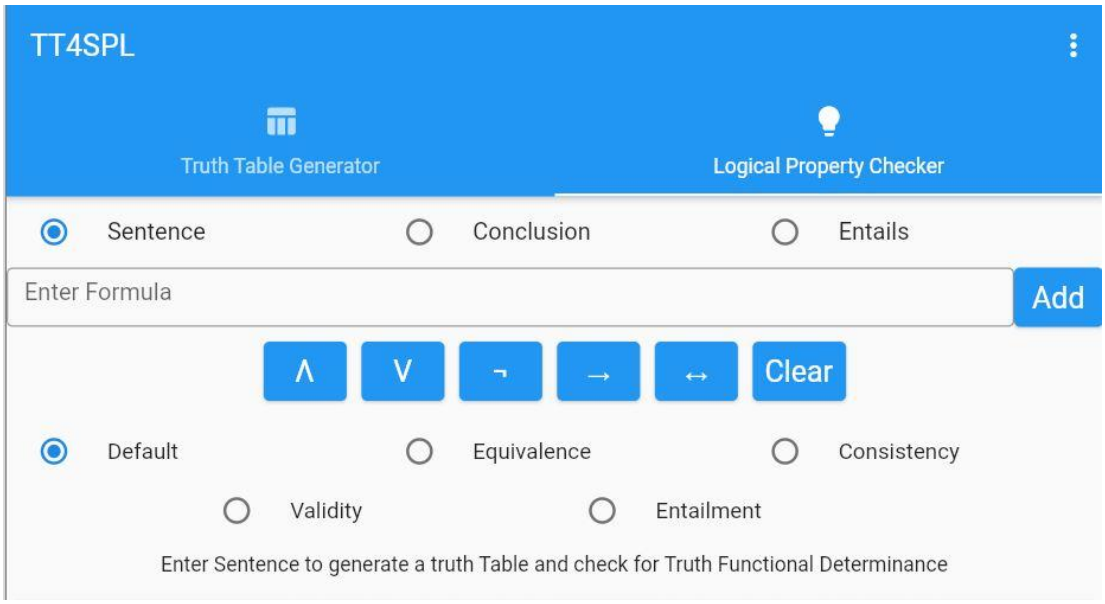
8.3 Truth Table Generator

When a set of one or more sentences have been added, the user is able to generate the truth table for this set by clicking the "Run" button at the bottom of the Truth Table Generator Screen. The generated Truth Table will then be displayed on this screen as shown below



8.4 Logical Property Checker

On the logical property check screen there are a further set of five radio buttons as shown below.



For each radio button the app will operate as follows:

- **Default** - With this radio button selected the app will test whether each of the given sentences are truth-functionally true, truth-functionally false, or truth-functionally indeterminant.

- **Equivalence** - With this radio button selected the app will test whether a set of 2 or more sentences of propositional logic are truth-functionally Equivalent to each other. This check requires at least 2 sentences be added.

- **Consistency** - With this radio button selected the app will test whether or not a given argument is truth-functionally Consistent. This check requires at least two sentences be added.

- **Validity** - With this radio button selected the app will test whether a given argument is truth-functionally valid. An argument is made up of one or more set of sentences that form a Premise and 1 sentence designated as the conclusion. This check will require that at least one premise as well as a conclusion.

- **Entailment** - With this radio button selected the app will test whether or not a set of sentences K is entailed by another sentence E. This check will require at least 1 sentence for the set of sentences K and an entailment sentence E.

The selected logical check can be run by clicking the "Run button at the bottom of the logical property check screen. Displayed below is an example of the results displayed for a “Default” logical check.