SYNTAX CHECKER

The Syntax Checker's backend uses the SyntaxChecker.py file to determine if any syntax related error occurs on parse. If any specific error is formed due to syntax related issues, the specific error code is packaged and sent back to the user. If the syntax is valid, a 'Valid Syntax' message is sent to the user to be displayed on the frontend.

TRUTH TABLE GENERATOR

The truth table generator's backend uses the SyntaxChecker.py to extract and generate a truth table. The endpoint begins by extracting all valid variables from the provided expression. Next, a Cartesian product is applied to each of these variable's true/false values to create a list of permutations, each representing the state of True/False values for each of the variables. Next, the function "traverse_verify" is used from syntaxChecker.py. This function takes a logical expression and a mapping of variables to their True/False values.

The backend finds the truth statement of the given expression for all the permutations of true/false values of the variables.

This is then packaged and sent to the front end. The front-end uses Javascript to build a table element with the truth-table information.

PARSE TREE

The Parse tree's backend exposes an endpoint called "generate_tree". This function uses the syntaxChecker.py 's "traverse_pathing" function. The "traverse_pathing" function traverses through the tree, using a Depth-first search. Each element detected is given a unique identifier and added list-like structure. This structure is then sent to the front end.

The front-end of the parse tree uses the JavaScript library, Dagre-D3, to render the Acyclc Directed Graph (parse tree). When the user sends an expression to the backend, the backend creates the list-like structure and sends it to the frontend. This resulting data is parsed and fed into the Dagre-D3 graphing utility and an SVG is created with the visual representation of the parse tree.

TEST YOURSELF

The Test-Yourself backend uses a JSON file, "questions.json", to extract all of its questions. The backend exposes 2 endpoints, "get_questions" and "score_questions". The "get_questions" endpoint reads each saved question from the file, appends a unique ID onto each question/answer, removes all traces of which answer is correct, shuffles all questions/answers, and packages that into a readable format to be sent to the front-end. The "score_questions" endpoint receives a list of key-value pairs in the form of {"question_id":"answer_id"}. The endpoint compares the responses from the frontend against its list of questions, and determines which questions are right/wrong. The endpoint returns to the front-end any instance of an incorrect question along with the reason that question was incorrect.

The front end uses Javascript to initially call the "get_questions" endpoint to populate the questions on the page. The form submission is captured by Javascript which calls the "score_questions" endpoint.