KI-Programmierung, Simple Scheduling

Roman Gerloff

WS 2022

1 Parsing

Tokens are used to parse the input, and the parser consists mainly of three classes. The WordParser then separates the input string into individual words. The Tokenizer ensures that the words correspond to the grammar and generates tokens. Finally, there is the TaskParser, which checks if a task can be generated. An object of the class Task is created and filled with the extracted names, durations, and a node tree representing the dependencies.

2 Code description

2.1 WordParser

The WordParser class reads words from a file one at a time. The next_word() method reads characters from the file until it finds a complete word, then returns it. Brakets are threated like words for easier usage later in the parsing. If the next_word() reaches the end of the file, the method will return None.

Example:
Input: "Task4 takes 2 needs Task1 and (Task2 or Task3)"
Output: "Task4", "takes", "2", "needs", "Task1", "and", "(", "Task2", "or", "Task3" , ")"

2.2 Tokenizer

The Tokenizer class takes words from the WordParser and returns tokens with the corresponding type. The most important method is get_token(), which reads a word from the file using an instance of the WordParser class and returns a Token object representing the word. The Token object is created by matching against a set of known words such as AND, OR, NEEDS, TAKES, OPEN_BRACKET, CLOSE_BRACKET or know word structure such as NUMBER or NAME. If the word does not match any of these the method raises an exception. The next_token() method allows retrieving a token. The peek_token() method allows a peek at the next token by using the get_token() method and storing the result in a buffer.

Example
Input: "Task4", "takes", "2", "needs", "Task1", "and", "(", "Task2", "or", "Task3" , ")"
Output(Only writing the TokenType): NAME, TAKES, NUMBER, NEEDS, NAME, AND, OPEN_BRACKET, NAME, OR, NAME, CLOSING_BRACKET

2.3 Taskparser

The Taskparser class can create a list of tasks. It does this using the tokens from the Tokenizer. The TaskParsers builds a task object, which includes a name, duration, and dependencies. The TaskParser first calls the get_task() method. This method calls get_name(), get_duration(), and get_dependencies() method. Each method compares the next token corresponds to the structure given by the grammar. Getting the tree of dependencies is the most complex one. The get_dependencies_list() method gathers all the tokens related to the dependencies, and the evaluate_dependencies() method evaluates the expression. The evaluate_dependencies_without_brackets() and evaluate_dependencies_with_brackets() methods are used to simplify the list of dependencies into a single task node. This process creates all tasks and returns the list.

Example
Input of Tokens(Only writing the TokenType): NAME, TAKES, NUMBER, NEEDS, NAME, AND, OPEN_BRACKET, NAME, OR, NAME, CLOSING_BRACKET
Printed Task:
Name: Task4
Duration: 2
Depedencies: And(Task1,Or(Task2,Task3)

Dependencies is a Tree with the OperatorNode as the root. The left child is Task1 and the right child is a OperatorNode that has Task2 and Task3 as children.

3 Additional remarks

Dependency expressions containing parentheses are first evaluated until they contain no parentheses. Then expressions remain, which can be simplified by the evaluate_dependencies_without_brackets() method. The "and" operators are evaluated first, followed by the "or" operators. The simplify() method searches for the first operator and creates a new token containing an OperatorNode having children left and right that are a TaskNode or another tree. Through this process, the tree is created from the bottom up. It terminates when the dependencies contain only one token.

4 Deviations

It was assumed that words have to be separated. E.g "Task1takes..." will fail, because the keyword "takes" is missing, and "Task1takes" gets identified as the name of the Task.

5 Potential extentions or alternate designs

Move the evaluation of dependencies into a single class if the specification is changed or additional operators are implemented. Adding new TokenTypes can be easily implemented in the Token class. These can be used for changes of the grammar or the TaskParser.

6 Bugs

The error messages for invalid input might be unhelpful.