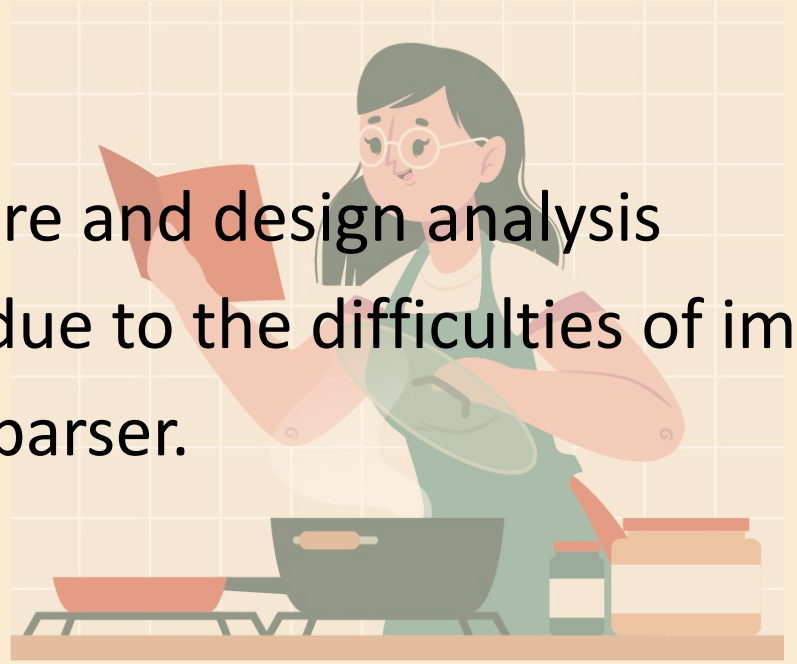
MICHELIN

A Code which Cooks

Michelin's Parser Design

Contents

At this phase of the project, few important things to take a note of are:

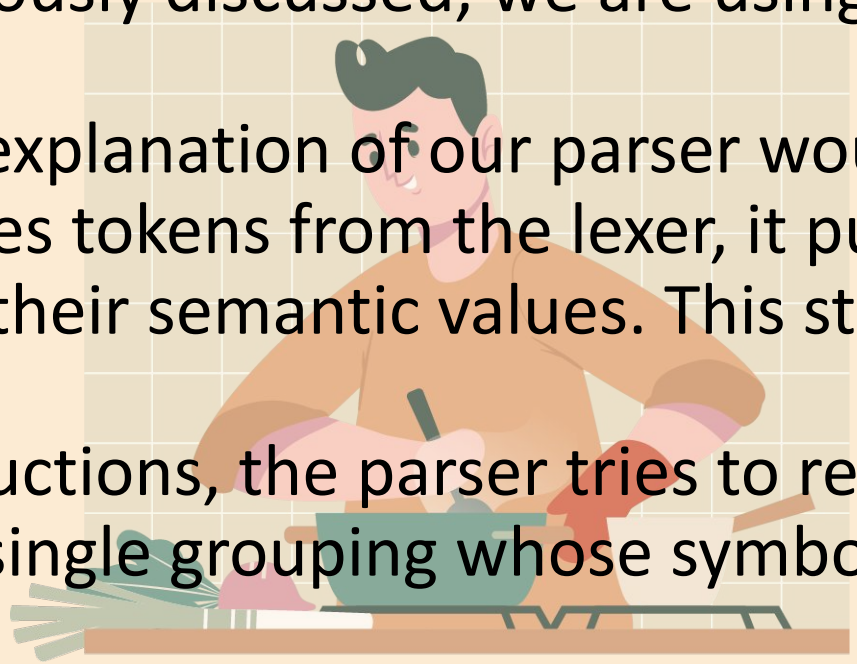


1. Parser's structure and design analysis
2. Changes made due to the difficulties of implementation
3. Working of the parser.

Background of the Parser

As we have previously discussed, we are using Ocamlyacc to make our parser.

A brief subjective explanation of our parser would be:

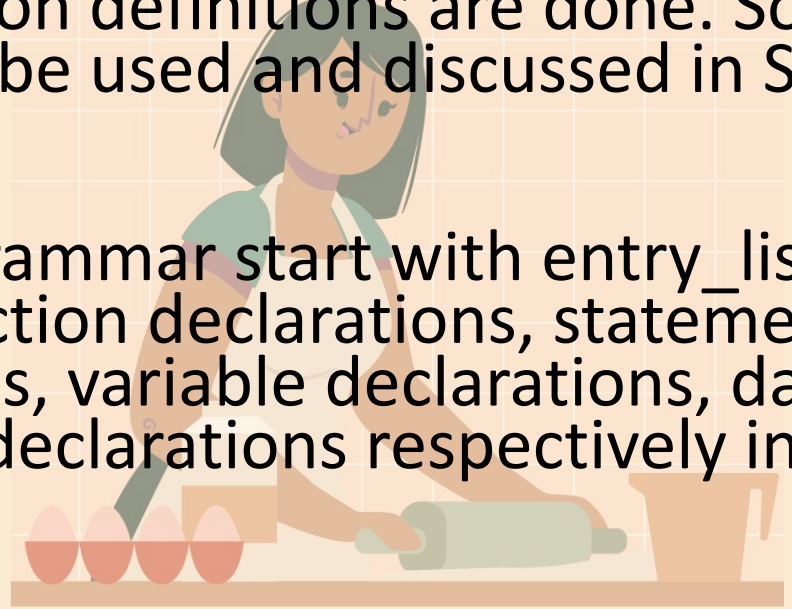


- As the parser takes tokens from the lexer, it pushes them onto a stack along with their semantic values. This stack is called as parser stack.
- By shifts and reductions, the parser tries to reduce the entire input down to a single grouping whose symbol is the grammar's start-symbol.
- Hence our parser is basically a bottom-up parser.

Structural Overview

The structure of Michelin's parser starts with types of how variable definitions, operator definitions, class definitions, argument definitions and function definitions are done. Some of the operators which will effectively be used and discussed in Semantic analyser are also added.

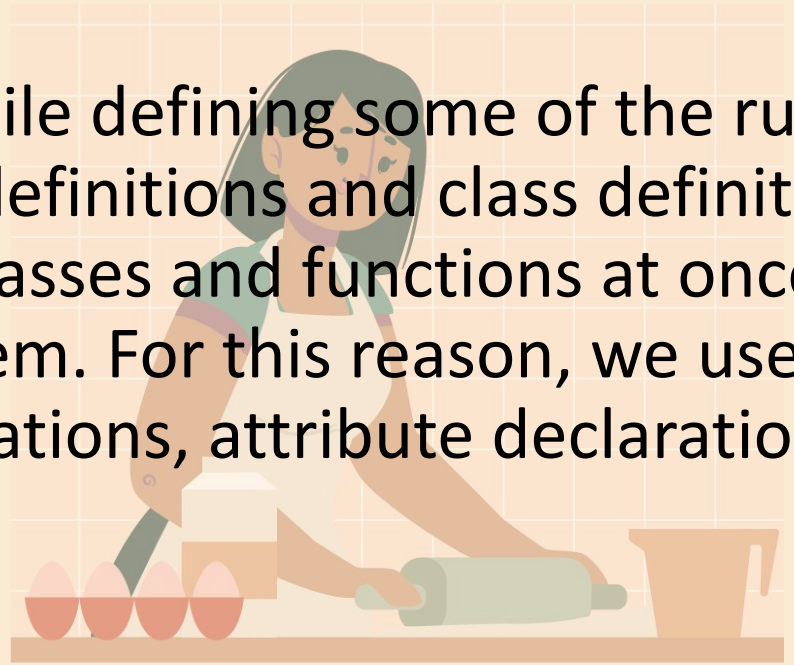
The rules of our grammar start with entry_list, then we go forward to entry, function declarations, statement declarations, argument declarations, variable declarations, data type declarations and then expression declarations respectively in the same order.



Structural Overview

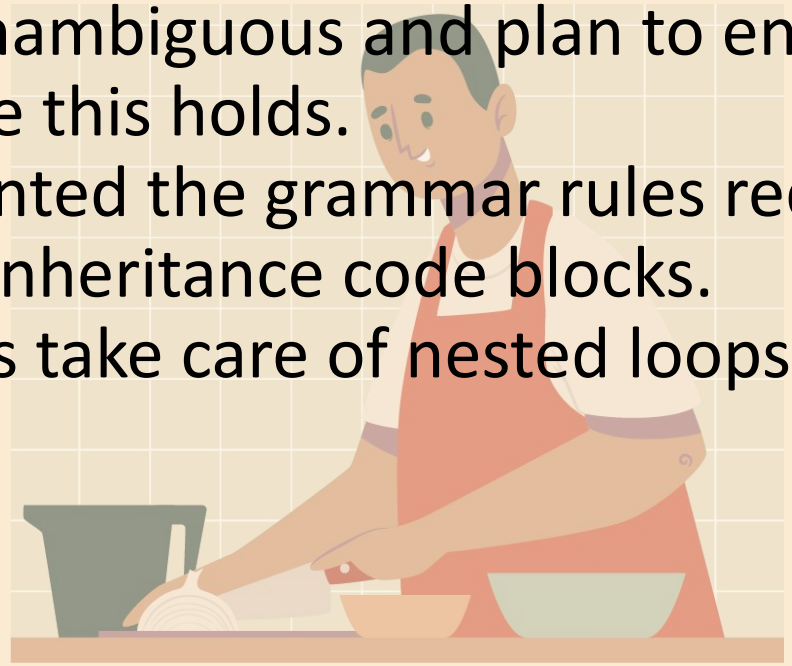
Usage as Lists

We faced an issue while defining some of the rules for our parser. In writing the function definitions and class definitions, we were not able to define both classes and functions at once with the respective tokens for each of them. For this reason, we used list structure to parse function declarations, attribute declarations and body of the code as well.



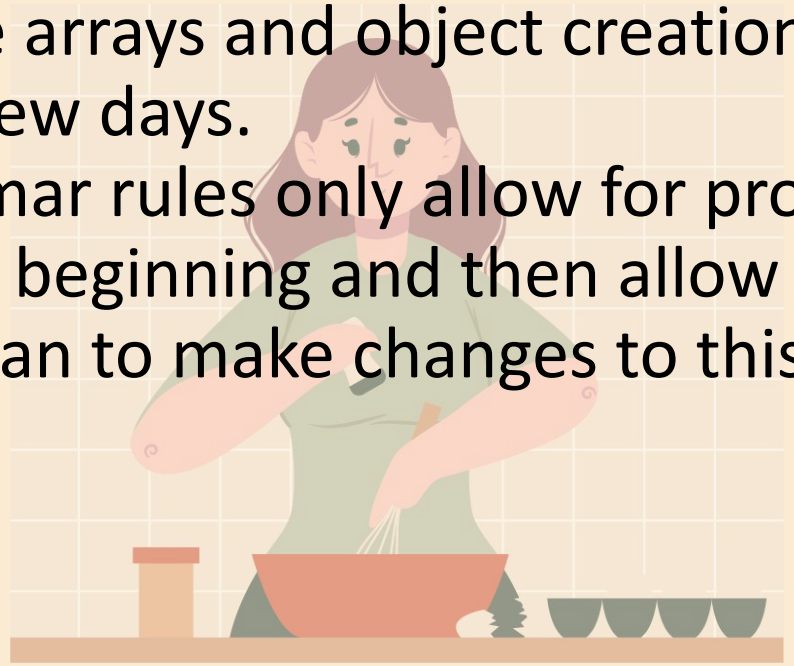
Key Points to Notice

- Our compiler is a LALR parser.
- Our grammar is unambiguous and plan to ensure that any further rules added ensure this holds.
- We have implemented the grammar rules required to parse through the class inheritance code blocks.
- Our grammar rules take care of nested loops as well.



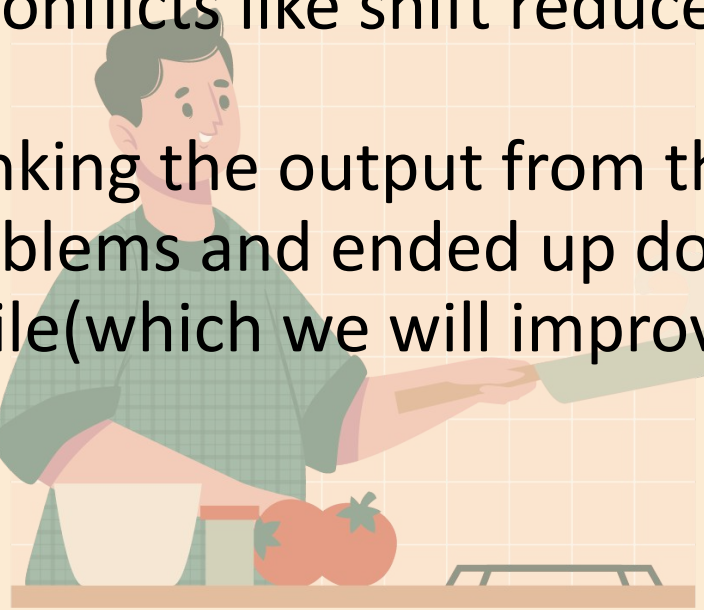
Obstacles in the design

- We encountered problems in the implementation of certain grammar rules like arrays and object creation and will implement these in the next few days.
- Our current grammar rules only allow for programs that declare all variables at the beginning and then allow us to declare statements , we plan to make changes to this and make it more flexible.



Challenges Faced

- In the process of defining our shift reduce parser we encountered various reduction conflicts like shift reduce conflicts and reduce reduce conflicts.
- In the process of linking the output from the lexer to the parser we faced many problems and ended up doing it through an intermediary text file(which we will improve upon).



Team Details

Sai Sidhardha Grandhi
CS19BTECH11050
Project Manager
GitHub ID: G-Sidhardha

Krishn Vishwas Kher
ES19BTECH11015
Language Guru
GitHub ID: KrishnKher

Mukkavalli Bharat Chandra
ES19BTECH11016
System Architect
GitHub ID: chandra3000

Sujeeth Reddy
ES19BTECH11022
System Integrator
GitHub ID: Sujeeth13

Sree Prathyush Chinta
CS19BTECH11043
Tester
GitHub ID: Prathyush-1886

Vemulapalli Aditya
CS19BTECH11025
System Integrator
GitHub ID: VEMULAPALLI-ADITYA

Praneeth Nistala
CS19BTECH11054
Tester
Github ID: Praneeth-Nistala

Thank You