Homework №8

Вадим Шабашов

Задание 1

Решение:

1. Грамматика из прошлого ДЗ принадлежит к $LL(1)$. Объяснить можно было бы построив таблицу и убедившись, что в ней все ячейки имеют не более 1 правила. Но это уж очень долго... Поэтому попробую словами объяснить, почему по заданной грамматике всегда строится ровно одно левостороннее дерево. Надеюсь, это будет достаточно строго.

2. Сперва поймем, что W, B, C, O — однозначно строят левые деревья, т.к. в них максимум есть один рекурсивный вызов нетерминала в правой части (этот нетерминал потом однозначно раскрывается в парсере).

3. L_L— однозначно строит левое дерево, т.к. вызовы C и W — однозначно строят, а вызов L_L единственный.

   Это собственно и будет основной идеей доказательства. Мы структурной индукцией проходим снизу вверх. Сначала доказали, что базовые W, B, C и O выводят левые деревья, затем доказываем, что состоящее из них L_L выводит левое дерево. Итд. Мы как бы "поднимаемся вверх". Но тут надо быть аккуратным: не обязательно нетерминалы, выводящие левые деревья, будут вместе в правой части давать только левое дерево (пример CC). Тут важно, что у меня все нетерминалы разделены каким-нибудь нетерминальным символом (", {}, итд) или же не имеют общих символов вообще (например, W и все остальное). Это гарантирует, что разбор будет строиться однозначно. Левое дерево будет единственным.

4. Аналогично доказываем для L_C.

5. E_O и E_E — однозначно строят левое дерево, т.к. C и L_L однозначно строят, а также потому, что в них всего один рекурсивный вызов E_O и E_E будет потом.

6. Для A — в нем нет рекурсивных вызовов, а все нетерминалы в правой части однозначно строят дерево. Отсюда A будет однозначно строить левое дерево.

7. L_{AO} однозначно строит, потому что A и O — однозначно строят левое дерево, а также всего один рекурсивный вызов.

8. L_P однозначно строит, потому что A и L_{AO} — однозначно строят левое дерево, а также всего один рекурсивный вызов.

9. S однозначно строит, потому что W и L_P — однозначно строят левое дерево.

Задание 2

Реализовать парсер для вашего языка описаний конечных автоматов (с операциями над ними) при помощи AntLR

Решение:

1. Реализацию парсера на ANTLR добавил в файл `src/task2/Automata.g4`

   Этот файл, после компиляции командой ниже, создает лексер и парсер в папке `src/task2/antlr_files`:

   ```
   antlr4 -Dlanguage=Python3 Automata.g4 -visitor -o antlr_files
   ```

2. Для проверки парсинга написал `src/task2/task2.py`. В нем передаю файл с автоматом в найденные лексер и парсер, а они сохраняют свой вывод в `src/task2/result`

3. Создал два файла: `src/task2/correct_example.json` — с правильной грамматикой;

   и `src/task2/incorrect_example.json` — с неправильной грамматикой. Они сохраняют результаты соответственно в `src/task2/result/log_correct.json` и `src/task2/result/log_incorrect.json`

   В неправильной грамматике добавил заменил строку `"is_dfa": true` на `"is_dfa": "Nope"`. В этот момент парсер и падает с ошибкой.

4. Для вывода дерева написал свой класс, который обходит построенное синтаксическое дерево и печатает имя каждой вершины и что в ней распарсилось.

   Также добавил стандартный вывод ANTLR (`toStringTree`)

Задание 3

Промоделировать работу алгоритма $LL(1)$ на грамматике языка арифметических выражений с операциями $+$ и $\cdot$ над целыми числами с естественными приоритетами и ассоциативностью.

Решение:

1. Грамматика языка:

$$
\begin{array}{lcl}
S & \to & M\ E \\
E & \to & +\ M\ E | \varepsilon \\
M & \to & B\ R \\
R & \to & \cdot\ B\ R \mid \varepsilon \\
B & \to & (S) \mid 0 \mid\ -N\ P \mid N\ P \\
P & \to & 0P \mid 1P \mid 2P \mid 3P \mid 4P \mid 5P \mid 6P \mid 7P \mid 8P \mid 9P \mid \varepsilon \\
N & \to & 1|2|3|4|5|6|7|8|9
\end{array}
$$

2. Воспользуемся кодом с сайта:

   https://www.geeksforgeeks.org/compiler-design-ll1-parser-in-python/

   Создал `src/task3/task3.py`, в котором реализована грамматика. Проверяю работоспособность на строке:

   ```
   sample_input_string = "( 1 + 1 ) * 2 + ( - 6 )"
   ```

   Результаты лежат в `src/task3/result/log.txt`