Формальные языки

домашнее задание до 23:59 27.02

Шаблон кода для задания живет в соответсвующей ветке:
https://github.com/kajigor/fl_ifmo_2019_spr/tree/HW03

1. Реализовать парсер-комбинатор, который парсит список в общем виде, возвращая список элементов.

   `parseList elem delim lbr rbr minimumNumberElems`

   `elem` — парсер для элемента.

   `delim` — парсер для разделителя.

   `lbr` — парсер для открывающейся скобки.

   `rbr` — парсер для закрывающейся скобки.

   `minimumNumberElems` — минимальное количество элементов, которые должны быть в списке.

   Учтите, что скобки и разделители могут быть призвольного вида, не обязательно одинокими символами.

   Между элементами, разделителями и скобками может быть любое количество пробельных символов: пробелов, переносов строк, табуляций.

2. Реализовать парсер для автоматов. Используйте вашу реализацию комбинаторов, добавлять новые, конечно же, не запрещено.

   На вход принимает строку, на выход возвращает может быть автомат. Эта функция может не проверять автомат на корректность — только парсить синтаксическую структуру и возвращать `Just` значение типа `Automaton s q`, если синтаксических ошибок нет, и `Nothing` иначе. Если вы хотите сразу проверять автомат еще и на корректность, можно не городить 2 разных парсера, а сразу реализовывать следующее задание.

   Формат входа:

   - Описание автомата состоит из 5 списков, разделенных запятыми. Элементы в списках разделяются запятыми, скобки треугольные. Пока предполагаем, что запятые и скобки не могут быть ни символами алфавита, ни состояниями.
   - Порядок такой:
     - Список символов
     - Список состояний
     - Список из одного состояния — стартового
     - Список терминальных состояний
     - Список троек: `(состояние, символ, состояние)` — в круглых собках через запятую

3. Реализовать проверку автомата на корректность. Проверки должны выполняться как можно раньше: если начальное состояние не является элементом списка состояний, то список терминальных состояний не должен быть прочитан.

   Если не получается встроить проверки непосредственно в процесс анализа, реализуйте отдельную функцию проверки, которая принимает сам автомат.

Если вы не хотите писать на Haskell, реализуйте похожую на нашу библиотечку комбинаторов и используйте ее. Правила оформления работы остаются с прошлого домашнего задания.