Московский государственный университет имени
М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Отчёт о выполнении задания практикума

«Построение регулярной грамматики по конечному автомату»

Студент 324 группы
Д. А. Григорьев

Москва, 2024

1 Постановка задачи

По заданному конечному автомату восстановить соответствующую регулярную праволинейную формальную грамматику, включающую алфавиты (множества) терминальных и нетерминальных символов и набор правил грамматики, а также начальный символ грамматики. В случае недетерминированности автомата необходимо построить, кроме грамматики, эквивалентный ему детерминированный автомат (детерминированный автомат, распознающий тот же самый язык).

1.1 Базовые требования

1. Необходимость наличия инструмента для компиляции и запуска программы - stack;
2. Наличие тестов для проверки корректности работы программы;
3. Необходимый функционал для обработки некоторых ошибок во входных данных;
4. Вывод в отдельный файл построенных ДКА и грамматики в виде текста;
5. Построение регулярной праволинейной грамматики;
6. Перевод НКА в ДКА.

2 Модули проекта

Проект состоит из следующих модулей:

- `Check.hs` — проверка входных данных;
- `NfaToDfa.hs` — реализация перевода НКА в ДКА;
- `Print.hs` — вывод в файл;
- `Read.hs` — считывание из файла;
- `Types.hs` — объявление основных типов;
- `TransformToGrammar.hs` — основной функционал для перевода автомата в грамматику.

В модуле `Types.hs` описаны следующие типы:

- `Symbol` — описывает символ из алфавита;
- `Transition` — описывает переход в автомате;
- `State` — описывает состояние в автомате;
- `DFA` — описывает конечный автомат;
- `NonTerminal` — описывает нетерминал в грамматике;
- `Production` — описывает один продукт в грамматике;
- `TransitionGr` — описывает правило в грамматике;
- `RegularGrammar` — описывает регулярную грамматику.

В модуле `NfaToDfa.hs` описаны следующие функции:

- `buildDFA` и `allIter` — построение ДКА по НКА алгоритмом Томпсона и итерация алгоритма по всем символам в алфавите;
- `move` — вычисление множества состояний, в которые можно перейти по символу из каждого состояния из данного множества;
- `hasTrans` и и `hasTrans1` — проверяют наличие соответсвующего перехода или его половины в автомате;
- `createTransition` — создание нового перехода в новом ДКА;
- `makeState` — создание новых состояний в новом ДКА.

В модуле `TransformToGrammar.hs` реализованы следующие функции:

- `getInitial` — возвращает входное состояние;
- `getFinals` — возвращает все финальные состояния;
- `transitionsToGrammar` — переводит все переходы в автомает в правила грамматики;
- `addFinal` — добавление финальных состояний автомата в грамматику;
- `getAllNonTerms` — переводит все состояния автомата в нетерминалы грамматики;

- `makeGrammar` — осуществляет перевод ДКА в регулярную праволинейную грамматику.

В модуле `Read.hs` реализованы следующие функции:

- `readAlph` — считывает алфавит автомата;
- `readStates` — считывает состояния автомата;
- `getOneTr` — переводит строку в переход автомата;
- `readTrans` —считывает все переходы в автомате;
- `readDFA` — приводит текстовое представление автомата во внутреннее представление в программе;
- `setTr` — назначает переходы уже считанному автомату.

В модуле `Print.hs` реализованы следующие функции:

- `symbolToString` — переводит алфавит в строку;
- `productionToString` — переводит продукцию грамматики в строку;
- `transitionGrToString` — переводит правило грамматики в строку;
- `getTransStr` —переводит все правила грамматики в строку;
- `nonTermStr` — переводит нетерминал грамматики в строку;
- `printTransition` — переводит все переходы в автомате в строку;
- `printStates` — переводит все состояния ДКА в строку;
- `printGrammar` — переводит грамматику в строку;
- `printDFA` — переводит ДКА в строку.

В модуле `Check.hs` реализованы следующие функции:

- `checkTrans` — проверяет, все ли переходы записаны должным обрзом;
- `checkFinal` — проверяет, есть ли в автомате допускающие состояния;
- `checkInitial` — проверяет, есть ли в автомате начальное состояние;
- `checkAlph` —проверяет, нет переходов по символу не из алфавита;
- `checkAll` — вызывает все вышеперечисленные проверки.

3 Используемые библиотеки/модули

При реализации использовались следующие встроенные библиотеки/модули:

- `Data.List` — функции nub для удаления дубликатов и функция intercalate для вставки спец. символа | между продуктами грамматики;
- `Data.Set` — встроенный тип для реализации множеств для перевода НКА в ДКА по алгоритму Томпсона.

4 Сценарии работы с приложением

- Для компиляции из терминала: stack build из папки проекта, затем stack exec grammar-exe. На данный момент входной файл называется auto.txt, но это название можно всегда указывать отдельно при запуске программы, главное, чтобы файл был в той же директории, что и проект;
- Выходные данные будут сохранятся в папку output в ту же директорию, что и сам проект.
- В проекте в папке Tests хранятся примеры тестов для запуска программы.
- Автомат задается следующим образом:
  - В начале в одной строке перечисляются все символы алфавита.
  - Затем на каждой строке в квадратных скобках перечисляются имена состояний. Допускающие состояния заключаются в еще одни квадратные скобки.
  - В конце на каждой строке перечисляются переходы в виде: состояние символ состояние. Состояния пишутся в квадратных скобках.
- Грамматика задается следующим образом:
  - В начале в одной строке пишется стартовый символ грамматики.
  - Затем в одной строке перечисляются все символы алфавита.
  - Затем на каждой строке перечисляются имена нетерминалов.
  - В конце на каждой строке перечисляются правила грамматики: Правая и левая часть каждого правила разделяются знаком ->, а сами правила – знаком \n , например: B -> aN|bN \n N -> cN|d.