Homework №7

Вадим Шабашов

Задание 1

Написать КС грамматику для языка описания конечных автоматов из предыдущих домашних заданий.

Решение:

Сделать КС грамматику можно, но с небольшими изменениями: я требовал в описании языка, чтобы были перечислены все списки смежности для всех вершин. Это удобно в при реализации, но не понятно, как это прикрутить с помощью КС. Нам нужно помнить все состояния автомата, которые есть по условию, и проверять, что они все описаны в edges. Кажется, что это вообще не получится сделать, т.к. число состояний конечного автомата может быть произвольным и неизвестным заранее. Поэтому я решил выкинуть это требование.

Здесь вообще говоря был тонкий момент: у меня в коде во время обработки лексером нет вообще никакой проверки, что это валидный автомат (и даже что вообще автомат). Например, не контролируется количество элементов в списках. Это отдельно проверяю позже. Но тогда это задание превращается в написание лексера для json и перестает быть связанным с конечными автоматами и моим форматом. Поэтому попытался найти баланс.

Пример файла на языке для автоматов в прикрепленном файле example.json

1. $V_T = \{`a - z`, `A - Z`, `0 - 9`, `"`, ` : `, `(`, `)`, `,`, `{`, `}`, `[`, `]`, `_`, ` `\}$
2. $V_N = \{S, W, C, B, O, A, L_C, L_P, L_{AO}, L_L, E_O, E_E\}$:
   - S — стартовое состояние
   - W — пробелы, переносы строки, табы
   - C — строка (то есть штука в "")
   - B — true и false
   - O — операция над автоматами
   - A — автомат
   - L_C — список строк
   - L_P — список пар из имени автомата и его определения (определение может быть списком из операций)
   - L_{AO} — список автоматов и операций
   - L_L — список списков из двух строк (для обычных ребер)
   - E_O — список обычных ребер
   - E_E — список эпсилон ребер
3. P:
   - $S \to W\{L_P\}W$

- $L_P \to \begin{bmatrix} W"C"W : W\{A\}W, L_P \\ W"C"W : W\{A\}W \\ W"C"W : W[L_{AO}]W, L_P \\ W"C"W : W[L_{AO}]W \end{bmatrix}$
-

$$
\begin{aligned}
&A \to \\
&\to W"glossary"W : W[L_C]W, && \cup \\
&\cup \quad W"states"W : W[L_C]W, && \cup \\
&\cup \quad W"initial\_state"W : W"C"W, && \cup \\
&\cup \quad W"terminal\_states"W : W[L_C]W, && \cup \\
&\cup \quad W"is\_dfa"W : WBW, && \cup \\
&\cup \quad W"edges"W : W\{E_O\}W, && \cup \\
&\cup \quad W"edges\_epsilon"W : W\{E_E\}W
\end{aligned}
$$

- $L_{AO} \to WAW, WOW, L_{AO} \mid W"C"W, WOW, L_{AO} \mid WAW \mid W"C"W$
- $L_C \to W"C"L_C | W"C"W$
- $E_E \to W"C"W : W[L_C]W, E_E \mid W"C"W : W[L_C]W$
- $E_O \to W"C"W : W[L_L]W, E_O \mid W"C"W : W[L_L]W$
- $L_L \to W[W"C"W, W"C"W]W, L_L \mid W[W"C"W, W"C"W]W \mid \varepsilon$
- $O \to "union" \mid "diff" \mid "intersect" \mid "concat" \mid "star"$
- $C \to aC \mid ... \mid zC \mid AC \mid ... \mid ZC \mid 0C \mid ... \mid 9C \mid \varepsilon$
- $B \to `true` \mid `false`$
- $W \to \backslash nW \mid \backslash n \mid `\ `W \mid `\ ` \mid \backslash tW \mid \backslash t \mid \varepsilon$

Логика в P довольно простая, но есть возня с пробелами и переносами, т.к. разрешаем их почти везде и в любом количестве.

Задание 2

Промоделировать работу алгоритма Эрли на грамматике из задания 1 и строке, использующей все синтаксические конструкции языка. Промоделировать работу алгоритма Эрли на любой некорректной строке.

Решение:

1. Т.к. формат довольно высокоуровневый (чтобы визуально читался хорошо), то очень много вспомогательных символов. Я попытался взять самый маленький автомат, но чтобы уместить все конструкции языка, все равно вышло 185 символов.

```
{
  "a": {
        "glossary":["0"],
        "states": ["A"],
        "initial_state": "A",
        "terminal_states": ["A"],
        "is_dfa": true,
        "edges": {
          "A": [["A","0"]]
        },
        "edges_epsilon": {
          "A": ["A"]
        }
  },
  "b": ["a", "union", "a"]
}
```

Попробую на автомате выше показать работу алгоритма Эрли. Для простоты буду пропускать работу с пробелами и переносами строки. Также сожму для краткости слова (нет смысла их полностью расписывать; там все идентично):

- "$glossary$" $\to$ "g"
- "$states$" $\to$ "s"
- "$initial\_state$" $\to$ "i"
- "$terminal\_states$" $\to$ "t"
- "is_dfa" $\to$ "d"
- "$edges$" $\to$ "e"
- "$edges\_epsilon$" $\to$ "p"
- "$union$" $\to$ "u"

Тогда строка выглядит:

$$\{\text{"}a\text{"} : \{\text{"}g\text{"} : [\text{"}0\text{"}], \text{"}s\text{"} : [\text{"}A\text{"}], \text{"}i\text{"} : \text{"}A\text{"}, \text{"}t\text{"} : [\text{"}A\text{"}], \text{"}d\text{"} : true, \text{"}e\text{"} : \{\text{"}A\text{"} : [[\text{"}A\text{"}, \text{"}0\text{"}]]\},$$

$$\text{"}p\text{"} : \{\text{"}A\text{"} : [\text{"}A\text{"}]\}\}, \text{"}b\text{"} : [\text{"}a\text{"}, \text{"}u\text{"}, \text{"}a\text{"}]\}$$

Сначала пытался сделать вручную:

(a) $S(0)$:

$$S' \to .S \quad 0$$
$$S \to \{L_P\} \quad 0$$

(b) $S(1)$:

$$S \to \{.L_P\} \quad 0$$
$$L_P \to .\text{"}C\text{"} : \{A\}, L_P | .\text{"}C\text{"} : \{A\} | .\text{"}C\text{"} : \{L_{AO}\}, L_P | .\text{"}C\text{"} : \{L_{AO}\} \quad 1$$

(c) $S(2)$:

$$L_P \to \text{".}C\text{"} : \{A\}, L_P|\text{".}C\text{"} : \{A\}|\text{".}C\text{"} : \{L_{AO}\}, L_P|\text{".}C\text{"} : \{L_{AO}\} \quad 1$$
$$C \to .aC \mid ... \mid .zC \mid .AC \mid ... \mid .ZC \mid .0C \mid ... \mid .9C| \, . \quad 2$$
$$L_P \to \text{"}C\text{."} : \{A\}, L_P|\text{"}C\text{."} : \{A\}|\text{"}C\text{."} : \{L_{AO}\}, L_P|\text{"}C\text{."} : \{L_{AO}\} \quad 1$$

(d) $S(3)$:

$$C \to a.C \quad 2$$
$$C \to . \quad 3$$
$$C \to aC. \quad 2$$
$$L_P \to \text{"}C\text{."} : \{A\}, L_P|\text{"}C\text{."} : \{A\}|\text{"}C\text{."} : \{L_{AO}\}, L_P|\text{"}C\text{."} : \{L_{AO}\} \quad 1$$

(e) $S(4)$:

$$L_P \to \text{"}C\text{".} : \{A\}, L_P|\text{"}C\text{".} : \{A\}|\text{"}C\text{".} : \{L_{AO}\}, L_P|\text{"}C\text{".} : \{L_{AO}\} \quad 1$$

(f) $S(5)$:

$$L_P \to \text{"}C\text{"} : .\{A\}, L_P|\text{"}C\text{"} : .\{A\}|\text{"}C\text{"} : .\{L_{AO}\}, L_P|\text{"}C\text{"} : .\{L_{AO}\} \quad 1$$

(g) ...

Здесь разобрался, как именно алгоритм работает. Но повторять все шаги до конца — это будет очень много. Поэтому нагуглил реализованный алгоритм Эрли на питоне и сделал вывод результата в файл log.txt.

Ссылка на реализованный алгоритм:

https://github.com/tomerfiliba/tau/blob/master/earley3.py

Код для запуска приложил к дз.