Лабораторная работа № 1.4 «Лексический распознаватель»

6 марта 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является изучение использования детерминированных конечных автоматов с размеченными заключительными состояниями (лексических распознавателей) для решения задачи лексического анализа.

Индивидуальный вариант

for, forward, &&, ||, строковые литералы ограничены двойными кавычками, для включения кавычки в строковой литерал она предваряется знаком «» (но знак «» без последующей кавычки ошибкой не является), могут пересекать границы строк текста.

Реализация

Лексическая структура языка — регулярные выражения для доменов:

Идентификатор	Регулярное выражение
SPACE	(\s)+
IDENT	\w(\w|\d)*
DIGIT	(\d)+
FOR	for
FORWARD	forward
_&&_	&&
_||_	||
STRING	"(.|\")"

Граф недетерминированного распознавателя:

```
digraph NFA {
    rankdir=LR

    3 [shape=doublecircle]
    5 [shape=doublecircle]
    7 [shape=doublecircle]
    11 [shape=doublecircle]
    19 [shape=doublecircle]
    22 [shape=doublecircle]
    25 [shape=doublecircle]
    29 [shape=doublecircle]
    1 -> 2 [label=λ] // пробелы
    2 -> 3 [label=SPACE]
    3 -> 3 [label=SPACE]

    1 -> 4 [label=λ] // идентификаторы
    4 -> 5 [label=ALPHA]
    5 -> 5 [label=ALPHA]
    5 -> 5 [label=DIGIT]

    1 -> 6 [label=λ] // числа
    6 -> 7 [label=DIGIT]
    7 -> 7 [label=DIGIT]

    1 -> 8 [label=λ] // for
    8 -> 9 [label=f]
    9 -> 10 [label=o]
    10 -> 11 [label=r]

    1 -> 12 [label=λ] // forward
    12 -> 13 [label=f]
    13 -> 14 [label=o]
    14 -> 15 [label=r]
    15 -> 16 [label=w]
    16 -> 17 [label=a]
    17 -> 18 [label=r]
    18 -> 19 [label=d]

    1 -> 20 [label=λ] // &&
    20 -> 21 [label="&"]
    21 -> 22 [label="&"]

    1 -> 23 [label=λ] // ||
    23 -> 24 [label="|"]
    24 -> 25 [label="|"]
```

```
    1 -> 26 [label=λ] // строки
    26 -> 27 [label="\""]
    27 -> 27 [label=ANY]
    27 -> 28 [label="\\"]
    28 -> 27 [label=ANY]
    27 -> 29 [label="\""]
}
```

Граф детерминированного распознавателя:

```
digraph DFA {
    rankdir=LR

    2 [shape=doublecircle]
    3 [shape=doublecircle]
    4 [shape=doublecircle]
    5 [shape=doublecircle]
    6 [shape=doublecircle]
    7 [shape=doublecircle]
    8 [shape=doublecircle]
    9 [shape=doublecircle]
    10 [shape=doublecircle]
    11 [shape=doublecircle]
    13 [shape=doublecircle]
    15 [shape=doublecircle]
    18 [shape=doublecircle]

    1 -> 2 [label=SPACE]
    2 -> 2 [label=SPACE]

    1 -> 3 [label=ALPHA_adforw]
    1 -> 3 [label=A]
    1 -> 3 [label=D]
    1 -> 5 [label=F]
    1 -> 3 [label=O]
    1 -> 3 [label=R]
    1 -> 3 [label=W]
    3 -> 3 [label=ALPHA_adforw]
    3 -> 3 [label=A]
    3 -> 3 [label=D]
    3 -> 3 [label=F]
    3 -> 3 [label=O]
    3 -> 3 [label=R]
    3 -> 3 [label=W]
    3 -> 3 [label=DIGIT]

    1 -> 4 [label=DIGIT]
```

```
4 -> 4 [label=DIGIT]

5 -> 3 [label=ALPHA_adforw]
5 -> 3 [label=DIGIT]
5 -> 3 [label=A]
5 -> 3 [label=D]
5 -> 3 [label=F]
5 -> 6 [label=O]
5 -> 3 [label=R]
5 -> 3 [label=W]

6 -> 3 [label=ALPHA_adforw]
6 -> 3 [label=DIGIT]
6 -> 3 [label=A]
6 -> 3 [label=D]
6 -> 3 [label=F]
6 -> 3 [label=O]
6 -> 7 [label=R]
6 -> 3 [label=W]


7 -> 3 [label=ALPHA_adforw]
7 -> 3 [label=DIGIT]
7 -> 3 [label=A]
7 -> 3 [label=D]
7 -> 3 [label=F]
7 -> 3 [label=O]
7 -> 3 [label=R]
7 -> 8 [label=W]

8 -> 3 [label=ALPHA_adforw]
8 -> 3 [label=DIGIT]
8 -> 9 [label=A]
8 -> 3 [label=D]
8 -> 3 [label=F]
8 -> 3 [label=O]
8 -> 3 [label=R]
8 -> 3 [label=W]

9 -> 3 [label=ALPHA_adforw]
9 -> 3 [label=DIGIT]
9 -> 3 [label=A]
9 -> 3 [label=D]
9 -> 3 [label=F]
9 -> 3 [label=O]
9 -> 10 [label=R]
```

```
9 -> 3 [label=W]

10 -> 3 [label=ALPHA_adforw]
10 -> 3 [label=DIGIT]
10 -> 3 [label=A]
10 -> 11 [label=D]
10 -> 3 [label=F]
10 -> 3 [label=O]
10 -> 3 [label=R]
10 -> 3 [label=W]

11 -> 3 [label=ALPHA_adforw]
11 -> 3 [label=DIGIT]
11 -> 3 [label=A]
11 -> 3 [label=D]
11 -> 3 [label=F]
11 -> 3 [label=O]
11 -> 3 [label=R]
11 -> 3 [label=W]

1 -> 12 [label="&"]
12 -> 13 [label="&"]

1 -> 14 [label="|"]
14 -> 15 [label="|"]

1 -> 16 [label="\""]

16 -> 16 [label=ANY_OTHER]
16 -> 16 [label=SPACE]
16 -> 16 [label=ALPHA_adforw]
16 -> 16 [label=DIGIT]
16 -> 16 [label=A]
16 -> 16 [label=D]
16 -> 16 [label=F]
16 -> 16 [label=O]
16 -> 16 [label=R]
16 -> 16 [label=W]
16 -> 16 [label="&"]
16 -> 16 [label="|"]

16 -> 17 [label="\\"]
17 -> 17 [label="\\"]
16 -> 18 [label="\""]

17 -> 16 [label=ANY_OTHER]
```

```
    17 -> 16 [label=SPACE]
    17 -> 16 [label=ALPHA_adforw]
    17 -> 16 [label=DIGIT]
    17 -> 16 [label=A]
    17 -> 16 [label=D]
    17 -> 16 [label=F]
    17 -> 16 [label=O]
    17 -> 16 [label=R]
    17 -> 16 [label=W]
    17 -> 16 [label="&"]
    17 -> 16 [label="|"]
    17 -> 16 [label="\""]
}
```

Реализация распознавателя:

```
#include <iostream>
#include <fstream>
#include <string>
#include <unordered_map>
#include <unordered_set>

#include "lib/ProgramaIterator.cpp"

enum class SymbolFactor {
    ANY_OTHER,
    SPACE,
    DIGIT,
    ALPHA_adforw,
    A,
    D,
    F,
    O,
    R,
    W,
    LOGIC_AND,
    LOGIC_OR,
    DOUBLE_QUOTES,
    BACKSLAH,
};

auto classificateSymbol(char symbol) -> SymbolFactor {
    if (isspace(symbol)) {
        return SymbolFactor::SPACE;
    }
    if (isdigit(symbol)) {
        return SymbolFactor::DIGIT;
```

```
    }
    if (isalpha(symbol)) {
        switch (symbol) {
        case 'a':
            return SymbolFactor::A;
        case 'd':
            return SymbolFactor::D;
        case 'f':
            return SymbolFactor::F;
        case 'o':
            return SymbolFactor::O;
        case 'r':
            return SymbolFactor::R;
        case 'w':
            return SymbolFactor::W;
        default:
            return SymbolFactor::ALPHA_adforw;
        }
    }
    switch (symbol) {
    case '&':
        return SymbolFactor::LOGIC_AND;
    case '|':
        return SymbolFactor::LOGIC_OR;
    case '"':
        return SymbolFactor::DOUBLE_QUOTES;
    case '\\':
        return SymbolFactor::BACKSLAH;
    default:
        return SymbolFactor::ANY_OTHER;
    }
}

using LexerDFA = std::unordered_map<
        std::size_t,        // начальная вершина ребра
        std::unordered_map<
            SymbolFactor,   // символ перехода из вершины
            std::size_t     // конечная вершина ребра
        >
    >;

const LexerDFA
INDIVIDUAL_LEXER = {
    {1, {
        {SymbolFactor::SPACE,           2},
        {SymbolFactor::ALPHA_adforw,    3},
```

```
        {SymbolFactor::DIGIT,           4},
        {SymbolFactor::A,               3},
        {SymbolFactor::D,               3},
        {SymbolFactor::F,               5},
        {SymbolFactor::O,               3},
        {SymbolFactor::R,               3},
        {SymbolFactor::W,               3},
        {SymbolFactor::LOGIC_AND,       12},
        {SymbolFactor::LOGIC_OR,        14},
        {SymbolFactor::DOUBLE_QUOTES,   16},
    }},
    {2, {
        {SymbolFactor::SPACE,           2},
    }},
    {3, {
        {SymbolFactor::ALPHA_adforw,    3},
        {SymbolFactor::A,               3},
        {SymbolFactor::D,               3},
        {SymbolFactor::F,               3},
        {SymbolFactor::O,               3},
        {SymbolFactor::R,               3},
        {SymbolFactor::W,               3},
        {SymbolFactor::DIGIT,           3},
    }},
    {4, {
        {SymbolFactor::DIGIT,           4},
    }},
    {5, {
        {SymbolFactor::ALPHA_adforw,    3},
        {SymbolFactor::DIGIT,           3},
        {SymbolFactor::A,               3},
        {SymbolFactor::D,               3},
        {SymbolFactor::F,               3},
        {SymbolFactor::O,               6},
        {SymbolFactor::R,               3},
        {SymbolFactor::W,               3},
    }},
    {6, {
        {SymbolFactor::ALPHA_adforw,    3},
        {SymbolFactor::DIGIT,           3},
        {SymbolFactor::A,               3},
        {SymbolFactor::D,               3},
        {SymbolFactor::F,               3},
        {SymbolFactor::O,               3},
        {SymbolFactor::R,               7},
        {SymbolFactor::W,               3},
```

```
}},
{7, {
    {SymbolFactor::ALPHA_adforw,    3},
    {SymbolFactor::DIGIT,           3},
    {SymbolFactor::A,               3},
    {SymbolFactor::D,               3},
    {SymbolFactor::F,               3},
    {SymbolFactor::O,               3},
    {SymbolFactor::R,               3},
    {SymbolFactor::W,               8},
}},
{8, {
    {SymbolFactor::ALPHA_adforw,    3},
    {SymbolFactor::DIGIT,           3},
    {SymbolFactor::A,               9},
    {SymbolFactor::D,               3},
    {SymbolFactor::F,               3},
    {SymbolFactor::O,               3},
    {SymbolFactor::R,               3},
    {SymbolFactor::W,               3},
}},
{9, {
    {SymbolFactor::ALPHA_adforw,    3},
    {SymbolFactor::DIGIT,           3},
    {SymbolFactor::A,               3},
    {SymbolFactor::D,               3},
    {SymbolFactor::F,               3},
    {SymbolFactor::O,               3},
    {SymbolFactor::R,              10},
    {SymbolFactor::W,               3},
}},
{10, {
    {SymbolFactor::ALPHA_adforw,    3},
    {SymbolFactor::DIGIT,           3},
    {SymbolFactor::A,               3},
    {SymbolFactor::D,              11},
    {SymbolFactor::F,               3},
    {SymbolFactor::O,               3},
    {SymbolFactor::R,               3},
    {SymbolFactor::W,               3},
}},
{11, {
    {SymbolFactor::ALPHA_adforw,    3},
    {SymbolFactor::DIGIT,           3},
    {SymbolFactor::A,               3},
    {SymbolFactor::D,               3},
```

```
            {SymbolFactor::F,               3},
            {SymbolFactor::O,               3},
            {SymbolFactor::R,               3},
            {SymbolFactor::W,               3},
        }},
        {12, {
            {SymbolFactor::LOGIC_AND,       13},
        }},
        {14, {
            {SymbolFactor::LOGIC_OR,        15},
        }},
        {16, {
            {SymbolFactor::ANY_OTHER,       16},
            {SymbolFactor::SPACE,           16},
            {SymbolFactor::DIGIT,           16},
            {SymbolFactor::ALPHA_adforw,    16},
            {SymbolFactor::A,               16},
            {SymbolFactor::D,               16},
            {SymbolFactor::F,               16},
            {SymbolFactor::O,               16},
            {SymbolFactor::R,               16},
            {SymbolFactor::W,               16},
            {SymbolFactor::LOGIC_AND,       16},
            {SymbolFactor::LOGIC_OR,        16},
            {SymbolFactor::DOUBLE_QUOTES,   18},
            {SymbolFactor::BACKSLAH,        17},
        }},
        {17, {
            {SymbolFactor::ANY_OTHER,       16},
            {SymbolFactor::SPACE,           16},
            {SymbolFactor::DIGIT,           16},
            {SymbolFactor::ALPHA_adforw,    16},
            {SymbolFactor::A,               16},
            {SymbolFactor::D,               16},
            {SymbolFactor::F,               16},
            {SymbolFactor::O,               16},
            {SymbolFactor::R,               16},
            {SymbolFactor::W,               16},
            {SymbolFactor::LOGIC_AND,       16},
            {SymbolFactor::LOGIC_OR,        16},
            {SymbolFactor::DOUBLE_QUOTES,   16},
            {SymbolFactor::BACKSLAH,        17},
        }},
};
const std::unordered_map<std::size_t, std::string>
FINAL_STATES = {
```

```
    {2, "SPACE"},
    {3, "IDENT"},
    {4, "DIGIT"},
    {5, "IDENT"},
    {6, "IDENT"},
    {7, "_FOR_"},
    {8, "IDENT"},
    {9, "IDENT"},
    {10, "IDENT"},
    {11, "_FORWARD_"},
    {13, "_&&_"},
    {15, "_||_"},
    {18, "STRING"},
};

int main() {
    std::ifstream fileStream{"prog.txt"};
    ProgramaIterator progIter{fileStream};

    // считываем из потока токены
    while (!progIter.eof()) {
        std::string finToken{};
        std::string finTokenType{};

        // сичтываем очередной токен
        std::string token{};
        std::size_t state = 1;
        while (true) {

            // случай распознания лексемы
            if ( FINAL_STATES.contains(state) ) {
                finToken = token;
                finTokenType = FINAL_STATES.at(state);
            }

            // случай захода в состояние ловушки
            SymbolFactor symFactor = classificateSymbol(progIter.cur());
            if ( !INDIVIDUAL_LEXER.contains(state) ||
                 !INDIVIDUAL_LEXER.at(state).contains(symFactor)
            ) {
                break;
            }

            // совершаем переход по классу символов
            state = INDIVIDUAL_LEXER.at(state).at(symFactor);
```

```
            // записываем следующий символ
            token += progIter.cur();

            // конец потока
            if ( progIter.eof() ) {
                break;
            }
            progIter.next();
        }

        // возвращаемся в начало считанного токена
        progIter.prev(token.size());

        // выводим результат считывания токена
        if (finToken.empty()) {
            std::cout
                << "ERROR "
                << progIter.pos()
                << '\n';
            progIter.next();
        } else {
            if (finTokenType != "SPACE") {
                std::cout
                    << finTokenType
                    << ' '
                    << progIter.pos()
                    << ": "
                    << finToken
                    << '\n';
            }
            progIter.next(finToken.size());
        }
        progIter.resetProgramaBuffer();
    }

    return 0;
}
```

Тестирование

Входные данные

```
for forward&&hello
print
```

```
exford
forward123
123213
"321\\\\"32dwq
    sdewq ывфыьывцйу"
3213ewwqe
&&
||
&|
|&
```

Вывод на `stdout`

```
_FOR_ (1, 1): for
_FORWARD_ (1, 5): forward
_&&_ (1, 12): &&
IDENT (1, 14): hello
IDENT (2, 1): print
IDENT (3, 1): exford
IDENT (4, 1): forward123
DIGIT (5, 1): 123213
STRING (6, 1): "321\\\\"32dwq
    sdewq ╤Л╨▓╤Д╤Л╤М╤Л╨▓╤Ж╨╢╤Г"
DIGIT (8, 1): 3213
IDENT (8, 5): ewwqe
_&&_ (9, 1): &&
_||_ (10, 1): ||
ERROR (11, 1)
ERROR (11, 2)
ERROR (12, 1)
ERROR (12, 2)
```

Вывод

В результатом выполнения данной работы было изучено использование детерминированных конечных автоматов с размеченными заключительными состояниями (лексических распознавателей) для решения задачи лексического анализа.