Лабораторная работа №3

Регулярные грамматики и конечные автоматы

Цель работы состоит в практическом построении лексического анализатора, распознающего цепочки символов генерируемых регулярными грамматиками.

Краткие теоретические сведения

Фазы компиляции включают лексический анализ, синтаксический анализ, семантический анализ, фазу генерации внутреннего представления программы, оптимизацию кода и генерацию объектного модуля.

В реальных компиляторах порядок фаз компиляции может быть несколько иным. Некоторые из них, например, лексический анализ и синтаксический анализ, целесообразно объединить в одну фазу. В интерпретаторах некоторые этапы компиляции могут отсутствовать.

В основе построения лексических анализаторов лежат регулярные грамматики. В настоящей лабораторной работе под регулярной грамматикой понимается *леволинейная* грамматика, не принадлежащая иерархии Хомского[1]. Для грамматик этого типа существует алгоритм разбора, который отвечает на вопрос: принадлежит ли анализируемая цепочка языку, порождаемому этой грамматикой, или нет. Алгоритм разбора таков:

- ✓ первый символ исходной цепочки $w = a_1 a_2 ... a_n \perp$ заменяется нетерминалом A, для которого в грамматике есть правило вывода $\mathrm{A} \to a_1$ (производится «свертка» терминала a_1 к нетерминалу A);
- ✓ до тех пор, пока не будет прочтена вся цепочка, выполняются следующие шаги:
- ✓ если в правилах грамматики имеется продукция $\mathrm{B} \to \mathrm{A} a_i \quad (i=2,3,..,n)$, то полученный на предыдущем шаге разбора нетерминал A, а также расположенный непосредственно справа от него очередной терминал a_i входной цепочки заменяется нетерминалом B.

Это эквивалентно построению дерева разбора восходящим методом: на каждом шаге алгоритма строится один из уровней дерева разбора от листьев к корню.

При работе этого алгоритма возможны следующие ситуации:

[1] Леволинейные грамматики не используются на практике.

- прочитана вся цепочка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге свертка привела к символу S. Это означает, что исходная цепочка $w = a_1 a_2 ... a_n \perp$ принадлежит языку $\mathbf{L}(\mathcal{G})$ генерируемому грамматикой $\mathcal{G}$.

- прочитана вся цепочка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге «свертка» не привела к символу S. Это означает, что исходная цепочка отвергается, она не принадлежит языку $\mathbf{L}(\mathcal{G})$.

- на некотором шаге не нашлось нужной «свертки», т.е. для полученного на предыдущем шаге нетерминала A и расположенного непосредственно справа от него очередного терминала a_i исходной цепочки не нашлось нетерминала B, для которого в грамматике было бы правило вывода $\mathrm{B} \to \mathrm{A}a_\mathrm{i}$. Это означает, что исходная цепочка отвергается, она не принадлежит языку $\mathbf{L}(\mathcal{G})$.

- на некотором шаге работы алгоритма оказалось, что есть более одной подходящей «свертки», т.е. в грамматике разные нетерминалы имеют правила вывода с одинаковыми правыми частями, и поэтому непонятно, к какому из них производить «свертку». Это говорит о *недетерминированности разбора.* Анализ этой ситуации будет дан ниже.

Пусть разбор на каждом шаге детерминированный. Распознаватель полностью характеризуется таблицей, строки которой помечены нетерминальными символами грамматики, а столбцы – терминальными. Значение каждого элемента таблицы – это нетерминальный символ, к которому можно свернуть конкатенацию нетерминала и терминала, помечающие соответствующие строку и столбец.

Пример 1. Для грамматики $\mathcal{G} = \left(\{a, b, \perp\}, \{\mathrm{S,A,B,C}\}, \mathbf{P}, \mathrm{S}\right)$ с правилами $\mathbf{P} = \{\mathrm{S} \to \mathrm{C} \perp, \mathrm{C} \to \mathrm{A}b \mid \mathrm{B}a, \mathrm{B} \to b \mid \mathrm{C}b\}$ таблица переходов такова

Табл. Таблица переходов

	a	b	⊥
C	A	B	S
A	-	C	-
B	C	-	-
S	-	-	-

Прочерк ставится в том случае, если «свертки» нет. Таблице переходов в соответствие ставится направленный помеченный графу, который показан на рис. 1.

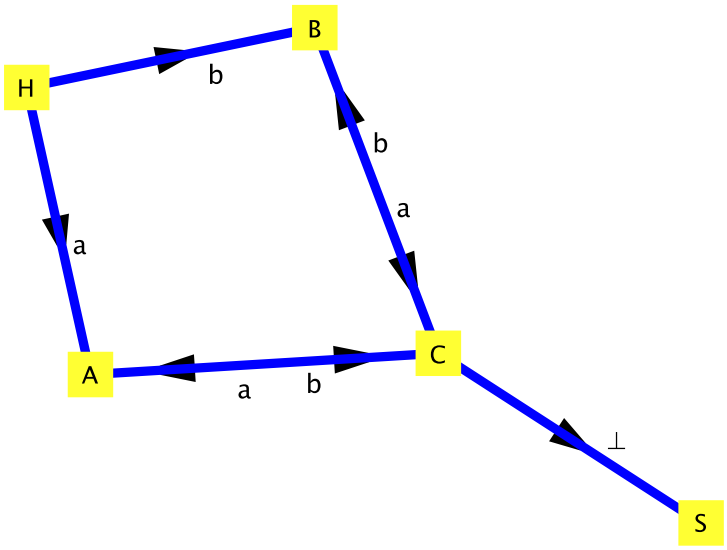

Рис. 1: Диаграмма переходов из **Примера** 1

Граф строится следующим образом:

- ✓ Строятся вершины графа, помеченные нетерминалами грамматики. Для каждого нетерминала грамматики граф содержит только одну вершину. Добавляется еще одна вершина, помеченная символом, отличным от символов нетерминалов, например, H. Эти вершины называют состояниями, H – начальное состояние.
- ✓ Вершины соединяются дугами по следующим правилам:
  - для каждого правила грамматики вида $\mathrm{A} \to a$ дугой соединяются состояния от H к A. Дуга помечается символом a.
  - для каждого правила $\mathrm{A} \to \mathrm{B}a$ дугой соединяются состояния от B к A, дуга помечается символом a.

Цепочки языка порождаются выводом в грамматике, например, для грамматики $\mathcal{G}$ с правилами $\mathbf{P} = \{\mathrm{S} \to \mathrm{C}\perp, \mathrm{C} \to \mathrm{A}b \mid \mathrm{B}a, \mathrm{B} \to b \mid \mathrm{C}b, \mathrm{A} \to a \mid \mathrm{C}a\}$ из **Примера** 1 вывод таков $\mathrm{S} \Rightarrow \mathrm{C}\perp \Rightarrow \mathrm{A}b\perp \Rightarrow \mathrm{C}ab\perp \Rightarrow \mathrm{B}aab\perp \Rightarrow \mathrm{C}baab\perp \Rightarrow \mathrm{B}abaab\perp \Rightarrow babaab\perp$.

Пусть имеются леворекурсивные правила $\mathrm{A} \to \mathrm{A}\gamma \mid \alpha$. Цепочки символов, порождаемые этими двумя правилами таковы $w = \alpha\gamma^* \in \mathbf{L}(\mathcal{G})$. Эти же цепочки можно порождаются другими праволинейными правилами $\mathrm{A} \to \alpha\mathrm{A}'$, $\mathrm{A} \to \gamma\mathrm{A}' \mid \varepsilon$, где A' – дополнительный нетерминал.

Алгоритм разбора по диаграмме состояний:

1. Начальным текущим состоянием объявляется H.

2. Пока цепочка полностью не прочитана, выполняются следующие шаги. Считывается текущий символ входной цепочки и осуществляется переход из текущего состояния в другое состояние по дуге, помеченной этим символом цепочки. Новое состояние становится текущим.
3. При работе этого алгоритма возможны следующие исходы:
   I. Прочитана вся цепочка. На каждом шаге находилась единственная дуга, помеченная очередным символом анализируемой цепочки. В результате последнего перехода распознаватель пришел в состояние S. Это означает, что исходная цепочка принадлежит языку $\mathbf{L}(\mathcal{G})$.
   II. Прочитана вся цепочка. Распознаватель пришел в состояние, отличное от состояния S. Это означает, что исходная цепочка не принадлежит языку $\mathbf{L}(\mathcal{G})$.
   III. На некотором шаге не нашлось дуги, выходящей из текущего состояния и помеченной очередным анализируемым символом. Это означает, что исходная цепочка не принадлежит языку $\mathbf{L}(\mathcal{G})$.
   IV. На некотором шаге работы алгоритма оказалось, что есть несколько дуг, выходящих из текущего состояния, помеченных очередным анализируемым символом, но ведущих в разные состояния. Это говорит о *недетерминированности разбора*. Анализ этой ситуации будет приведен ниже.

Диаграмма состояний определяет конечный автомат, построенный по регулярной грамматике, который допускает множество цепочек, генерируемых грамматикой. Состояния и дуги ДС – это графическое изображение функции переходов конечного автомата из состояния в состояние. Очередной анализируемый символ совпадает с символом-меткой дуги перехода.

Среди всех состояний выделяется начальное состояние. Вначале автомат находится в этом состоянии. Всегда имеется конечное состояние, в котором автомат завершает работу. Переходом в это состояние автомат допускает проанализированную им цепочку.

Определение. Конечный автомат (КА) – это пятерка $(\mathbf{Q}, \mathbf{\Sigma}, \delta, \mathbf{F}, S)$, где $\mathbf{Q}$ – конечное множество состояний; $\mathbf{\Sigma}$ – конечное множество допустимых входных символов; δ – отображение множества $\mathbf{Q} \times \mathbf{\Sigma}$ во множество $\mathbf{Q}$, то есть $\delta : \mathbf{Q} \times \mathbf{\Sigma} \to \mathbf{Q}$; $S \in \mathbf{Q}$ – начальное состояние КА; $\mathbf{F} \subset \mathbf{Q}$ – конечное множество заключительных состояний. Значение функции перехода $\delta(\mathrm{A}, a) = \mathrm{B}$ означает, что из состояния $\mathrm{A} \in \mathbf{Q}$ по входному символу $a \in \mathbf{\Sigma}$ происходит переход в состояние B. Запись $\delta(\mathrm{A}, a) = \varnothing$ говорит, что следующий шаг автомата невозможен.

Конечный автомат *допускает цепочку* $a_1 a_2 \ldots a_n$, если $\delta(\mathrm{H}, a_1) = \mathrm{A}_1$, $\delta(\mathrm{A}_1, a_2) = \mathrm{A}_2$, …, $\delta(\mathrm{A}_{\mathrm{n-2}}, a_{\mathrm{n-1}}) = \mathrm{A}_{\mathrm{n-1}}$, $\delta(\mathrm{A}_{\mathrm{n-1}}, a_{\mathrm{n}}) = \mathrm{S}$, где $a_i \in \mathbf{\Sigma}$, $\mathrm{A}_{\mathrm{j}} \in \mathrm{Q}$ $\left(\mathrm{j}=1,2,\ldots,\mathrm{n}\text{-}1; \mathrm{i}=1,2,\ldots,\mathrm{n}\right)$, H – начальное состояние, S – одно из заключительных состояний.

Для работы с диаграммами состояний вводятся соглашения:

- ✓ если из одного состояния в другое выходит несколько дуг, помеченных разными символами, то будем изображать одну дугу, помеченную всеми этими символами;
- ✓ непомеченная дуга будет соответствовать переходу при любом символе, кроме тех, которыми помечены другие дуги, выходящие из этого состояния.
- ✓ введем состояние ошибки (ER); переход в это состояние будет означать, что исходная цепочка языку не принадлежит.

По диаграмме состояний написан анализатор для регулярной грамматики. Для грамматики из **Примера** 1 анализатор будет таким:

```
#include <stdio.h>
#include <iostream>
int main()
{
std::cout << "Labo #3. Finite automata recognizer.\nGramma with left-recursive
productions: \nS->Cc\nC->Ab|Ba\nA->Ba\B->Ab" << std::endl;
enum states {H, A, B, C, S, ER}; /* the set of states */
enum states CS; /* CS is a current state */
FILE *fp;/*file pointer */
char c;
int cnt = 0;
std::string name;
name = "data.txt";
CS = H;
std::cout << "Data file: " << name << "\n\n\n";
fp = fopen("data.txt","r");
c = fgetc(fp);
cnt++;
printf("char %c number %d\n", c,cnt);
do
{
switch(CS)
 {
 case H:
       if(c == 'a')
             {
             c = fgetc(fp);
             cnt++;
             printf("char %c number %d\n", c,cnt);
             CS = A;
             }
       else if(c == 'b')
             {
             c = fgetc(fp);
             cnt++;
             printf("char %c number %d\n", c,cnt);
             CS = B;
             }
       else CS = ER;
       break;
 case A:
       if(c == 'b')
             {
             c = fgetc(fp);
             cnt++;
             printf("char %c number %d\n", c,cnt);
             CS = C;
             }
       else CS = ER;
       break;
```

```
        case B:
              if(c == 'a')
                    {
                    c = fgetc(fp);
                    cnt++;
                    printf("char %c number %d\n", c,cnt);
                    CS = C;
                    }
              else CS = ER;
              break;
        case C:
              if(c =='a')
                    {
                    c = fgetc(fp);
                    cnt++;
                    printf("char %c number %d\n", c,cnt);
                    CS = A;
                    }
              else if(c == 'b')
                    {
                    c = fgetc(fp);
                    cnt++;
                    printf("char %c number %d\n", c,cnt);
                    CS = B;
                    }
              else if(c == 'c') CS = S;
              else CS = ER;
              break;
        }
} while(CS != S && CS != ER);
              if(CS == ER)
                    {
                    printf("Rejected chain in position %d\n", cnt);
                    return -1;
                    }
              else
                    {
                    printf("Accepted chain of length %d\n", cnt);
                    return 0;
                    }
}
```

При анализе по регулярной грамматике может оказаться, что несколько нетерминалов имеют одинаковые правые части, и поэтому неясно, к какому из них делать «свертку» (см. описание алгоритма). В терминах диаграммы состояний это означает, что из одного состояния выходит несколько дуг, ведущих в разные состояния, но помеченных одним и тем же символом.

Пример 2. Для грамматики $\mathcal{G} = \left(\{a, b, \perp\}, \{S, A, B\}, \mathbf{P}, S\right)$ с множеством продукций $\mathbf{P} = \{\mathrm{S} \to \mathrm{A}a, \mathrm{A} \to a \mid \mathrm{B}b, \mathrm{B} \to b \mid \mathrm{B}b\}$, распознаватель будет недетерминированным, поскольку нетерминалы A и B обладают одинаковыми правыми частями – $\mathrm{B}b$. Такой распознаватель называется недетерминированным конечным автоматом.

Определение. Недетерминированный конечный автомат (НКА) – это пятерка $(\mathbf{Q}, \mathbf{\Sigma}, \delta, \mathbf{F}, S)$, где $\mathbf{Q}$ – конечное множество состояний; $\mathbf{\Sigma}$ – конечное множество допустимых входных символов; δ – отображение множества $\mathbf{Q} \times \mathbf{\Sigma}$ во множество подмножеств $\mathbf{Q}$, то есть $\delta : \mathbf{Q} \times \mathbf{\Sigma} \to \mathbb{P}(\mathbf{Q})$; $S \in \mathbf{Q}$ – начальное состояние НКА; $\mathbf{F} \subset \mathbf{Q}$ –

конечное множество заключительных состояний. Значение функции перехода $\delta(\mathrm{A}, a) = \{\mathrm{B}_1, \mathrm{B}_2, ..., \mathrm{B}_n\}$ означает, что из состояния $\mathrm{A} \in \mathbf{Q}$ по входному символу $a \in \mathbf{\Sigma}$ можно осуществить переход в любое из состояний $\mathrm{B}_i \in \mathbf{Q} \quad (i = 1, 2, ..., n)$, запись $\delta(\mathrm{A}, a) = \varnothing$ говорит, что следующий шаг автомата невозможен.

На практике НКА не используются, поскольку моделируются с использованием переборных алгоритмов. НКА играют ключевую роль в теории формальных языков. В теории формальных языков доказана теорема, что для любого НКА всегда можно построить детерминированный КА, распознающий тот же язык, распознает и НКА.

Индивидуальные задания

1	Дана грамматика: G = ({Q, P, R, S}, {0, 1, *, ⊥, /}, V, Q), где V: S→0R R→/Q | ⊥ Q→1P P→*S | ⊥ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
2	Дана грамматика: S→C⊥ B→B1|0|D0 C→B1|C1 D→D0|0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
3	Дана грамматика: S→C⊥ C→B1 B→0|D0 D→B1 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
4	Дана грамматика: S→A0 A→A0|S1|0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
5	Дана грамматика: S→0A|1S A→0A|1B B→0B|1B|⊥ Определить язык, который она порождает; построить ДС;

	написать на Си анализатор.
6	Дана грамматика: S→B⊥ A→B1|0 B→A1|C1|B0|1 C→A0|B1 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
7	Дана грамматика: G = ({S, C, D}, {0, 1}, P, S), где P: S→1C | 0D C→0D|0S|1 D→1C|1S|0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
8	Дана грамматика: S→ B0|0 B→ B0|C1|0|1 C→ B0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
9	Дана грамматика: S → Sb|Aa|a|b A → Aa|Sb|a Определить язык, который она порождает; построить ДС; написать на Си анализатор.
10	Дана грамматика: S→ A0|A1|B1|0|1 A→ A1|B1|1 B→ A0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
11	Дана грамматика: S→ S0|A1|0|1 A→ A1|B0|0|1 B→ A0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
12	Дана грамматика: S→ A⊥ A→ A0|A1|B1 B→ B0|C0|0 C→ C1|1 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
13	Дана грамматика: G = ({I, J, K, M, N}, {0, 1, ~, !}, P, I), где P: I→0J | 1K | 0M

	J→~K | 0M K→~M | 0J | 0N M→1K | ! N→0I|1I|! Определить язык, который она порождает; построить ДС; написать на Си анализатор.
14	Дана грамматика: G=({S, A, B, C}, {a, b, c}, P, S), где P: S→ aA | bB | aC A→ bA | bB | c B→ aA | cC | b C→ bB | bC |a Определить язык, который она порождает; построить ДС; написать на Си анализатор.
15	Дана грамматика: G = ({K, L, M, N}, {a, b, +, –, ⊥}, P, K), где P: K → aL | bM L→ –N|–M M→ +N N→ aL | bM | ⊥ Определить язык, который она порождает; построить ДС; написать на Си анализатор.
16	Дана грамматика: G = ({X, Y, Z, V, W}, {0, 1, x, y, z}, P, X), где P: X → yY | zZ Y→1V Z→0W|0Y V → xZ | xW | 1 W→1Y|0 Определить язык, который она порождает; построить ДС; написать на Си анализатор.
17	Дана грамматика: G = ({S, A, B, C, D}, {a, b, c, d, ⊥}, P, S), где P: S → aA | bB A → cC | ⊥ C → cC | cA B → dD | ⊥ D → dD |dB Определить язык, который она порождает; построить ДС; написать на Си анализатор.
18	Дана грамматика: G = ({S, A, B, C, D}, {a, b, c, d, #, ⊥}, P, S), где P: S→aA | bB | cC A→dD B→#D D→dD | dB | ⊥ C→cD Определить язык, который она порождает; построить ДС;

	написать на Си анализатор.
19	Даны две грамматики G1 и G2, порождающие языки L1 и L2. Построить регулярную грамматику для L1 ∪ L2. Построить ДС и написать на Си анализатор для полученной грамматики. **G1**: S →S1 | A0 A→A1|0 **G2**: S →A1 | B0 | E1 A →S1 B →C1 | D1 C →0 D →B1 E→E0|1
20	Даны две грамматики G1 и G2, порождающие языки L1 и L2. Построить регулярную грамматику для пересечения L1 и L2. Для полученной грамматики построить ДС и написать на Си анализатор. **G1**: S →S1 | A0 A→A1|0 **G2**: S →A1 | B0 | E1 A →S1 B →C1 | D1 C →0 D →B1 E→E0|1
21	Для данной грамматики • определить ее тип; • определить, какой язык она порождает; • написать регулярную грамматику, эквивалентную данной; • построить ДС и анализатор на Си. S→0S|S0|D D→DD|1A|ε A→0B|ε B→0A|0

Литература

[1] Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов. 3-е изд. – СПб.: Питер, 2010. – 400 с.

[2] Ахо А.В., Лам М. С., Сети Р., Ульман Дж. Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: Вильямс, 2008. – 1184 с. (разделы 3.4,3.6, 3.7)

[3] Волкова И.А., Руденко Т.В. Формальные грамматики и языки. Элементы теории трансляции: Учебное пособие. – Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 1999. – 62 с.

[4] Свердлов С.З. Языки программирования и методы трансляции: Учебное пособие. — СПб.: Питер, 2007. — 638 с. (стр. 221–230)