Лексический анализ

Гладков Артемий Николаевич

25.09.2024

Этапы построения транслятора

Классическая схема компиляции программы:

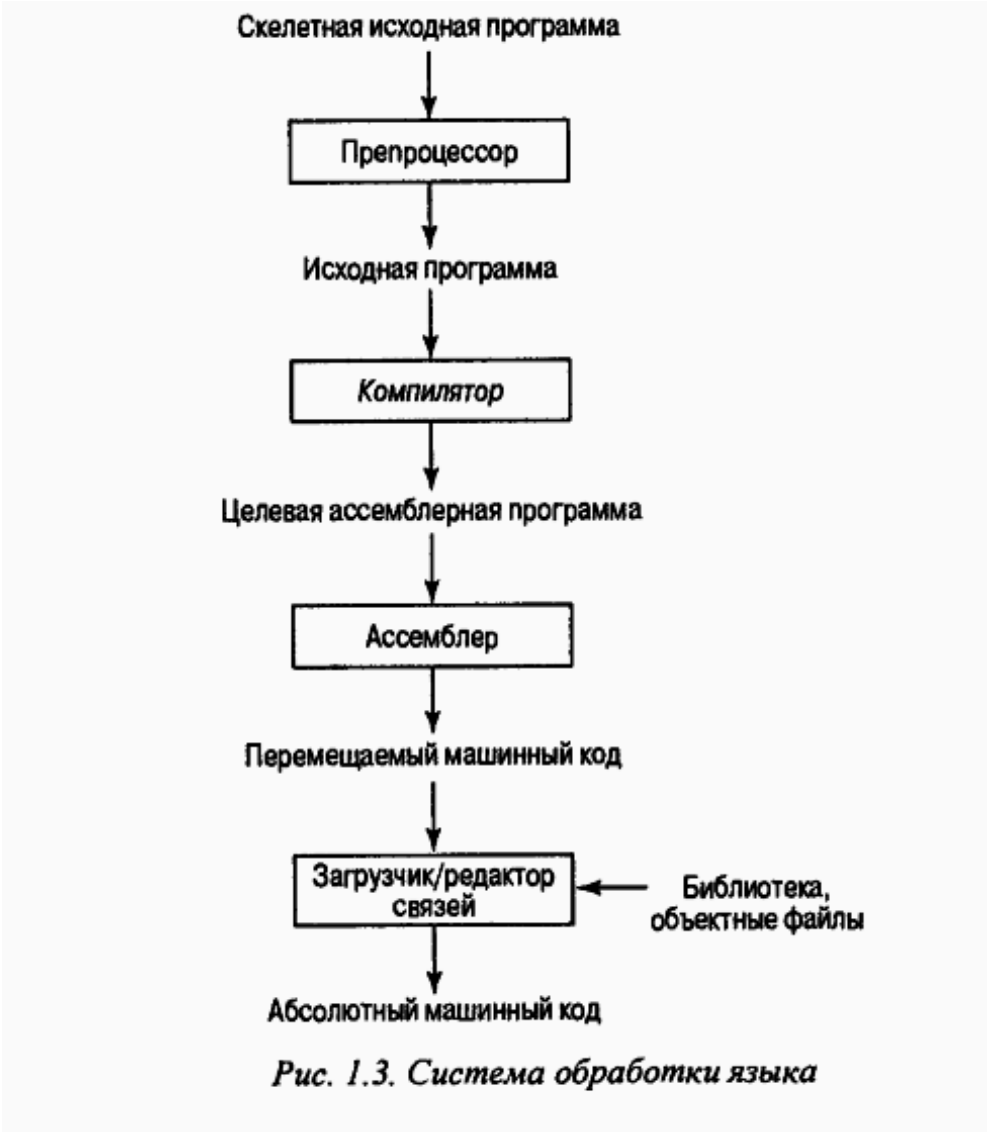


Рис. 1.3. Система обработки языка

Что будет пройдено в рамках курса

- Лексический анализ (Лаб. работа №1)
- Синтаксический анализ (Лаб. работа №2)
- Семантический анализ (Лаб. работа №3)
- Генерация внутренних представлений (дополнительное задание в Лаб. работе №3)
- Исполнение (дополнительное задание в Лаб. работе №3)

Что читать?

В. А. Соколов Введение в теорию формальных языков, Ярославль, ЯрГУ, - 2014г.

M-язык

$P \to \textbf{program}\ D_1\ B\perp$
$D_1 \to \textbf{var}\ D\ \{;D\}$
$D \to I\{;I\} : [\textbf{int}|\textbf{bool}]$
$B \to \textbf{begin}\ S\{;S\}\ \textbf{end}$
$S \to I := E \mid \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S \mid \textbf{while}\ E\ \textbf{do}\ S \mid B \mid \textbf{read}(I) \mid \textbf{write}(E)$
$E \to E_1 \mid E_1[= \mid < \mid > \mid ! =]E_1$
$E_1 \to T\{[+|-|\textbf{or}]T\}$
$T \to F\{[*|/|\textbf{and}]F\}$
$F \to I \mid N \mid L \mid \textbf{not}\ F \mid (E)$
$L \to \textbf{true} \mid \textbf{false}$
$I \to C \mid IC \mid IR$
$N \to R \mid NR$
$C \to a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z$
$R \to 0 \mid 1 \mid \ldots \mid 9$

M-язык

$P \to \textbf{program}\ D_1\ B\perp$
$D_1 \to \textbf{var}\ D\ \{;D\}$
$D \to I\{;I\} : [\textbf{int}|\textbf{bool}]$
$B \to \textbf{begin}\ S\{;S\}\ \textbf{end}$
$S \to I := E \mid \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S \mid \textbf{while}\ E\ \textbf{do}\ S \mid B \mid \textbf{read}(I) \mid \textbf{write}(E)$
$E \to E_1 \mid E_1[= \mid < \mid > \mid ! =]E_1$
$E_1 \to T\{[+|-|\textbf{or}]T\}$
$T \to F\{[*|/|\textbf{and}]F\}$
$F \to I \mid N \mid L \mid \textbf{not}\ F \mid (E)$
$L \to \textbf{true} \mid \textbf{false}$
$I \to C \mid IC \mid IR$
$N \to R \mid NR$
$C \to a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z$
$R \to 0 \mid 1 \mid \ldots \mid 9$

Полужирным выделены **ключевые слова**.
$\perp$ — символ конца программы.

M-язык

$P \to \textbf{program}\ D_1\ B\perp$
$D_1 \to \textbf{var}\ D\ \{;D\}$
$D \to I\{;I\} : [\textbf{int}|\textbf{bool}]$
$B \to \textbf{begin}\ S\{;S\}\ \textbf{end}$
$S \to I := E \mid \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S \mid \textbf{while}\ E\ \textbf{do}\ S \mid B \mid \textbf{read}(I) \mid \textbf{write}(E)$
$E \to E_1 \mid E_1[= \mid < \mid > \mid ! =]E_1$
$E_1 \to T\{[+|-|\textbf{or}]T\}$
$T \to F\{[*|/|\textbf{and}]F\}$
$F \to I \mid N \mid L \mid \textbf{not}\ F \mid (E)$
$L \to \textbf{true} \mid \textbf{false}$
$I \to C \mid IC \mid IR$
$N \to R \mid NR$
$C \to a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z$
$R \to 0 \mid 1 \mid \ldots \mid 9$

Полужирным выделены **ключевые слова**.
$\perp$ — символ конца программы.
$\{\alpha\} = \alpha^* = \alpha^n, n \geq 0$

M-язык

$P \to \textbf{program}\ D_1\ B\perp$
$D_1 \to \textbf{var}\ D\ \{;D\}$
$D \to I\{;I\} : [\textbf{int}|\textbf{bool}]$
$B \to \textbf{begin}\ S\{;S\}\ \textbf{end}$
$S \to I := E \mid \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S \mid \textbf{while}\ E\ \textbf{do}\ S \mid B \mid \textbf{read}(I) \mid \textbf{write}(E)$
$E \to E_1 \mid E_1[= \mid < \mid > \mid ! =]E_1$
$E_1 \to T\{[+|-|\textbf{or}]T\}$
$T \to F\{[*|/|\textbf{and}]F\}$
$F \to I \mid N \mid L \mid \textbf{not}\ F \mid (E)$
$L \to \textbf{true} \mid \textbf{false}$
$I \to C \mid IC \mid IR$
$N \to R \mid NR$
$C \to a \mid b \mid \ldots \mid z \mid A \mid B \mid \ldots \mid Z \leftarrow$ Просто заглавные буквы.
$R \to 0 \mid 1 \mid \ldots \mid 9$

Полужирным выделены **ключевые слова**.
$\perp$ — символ конца программы.
$\{\alpha\} = \alpha^* = \alpha^n, n \geq 0$

Задача лексического анализа

Лексический анализ (ЛА) – первый этап процесса компиляции. Символы, составляющие исходную программу, группируются в отдельные лексические элементы, называемые **лексемами**. Лексический анализ важен для процесса компиляции по нескольким причинам:

- замена в программе идентификаторов, констант, ограничителей и служебных слов лексемами для более удобной обработки;
- лексический анализ уменьшает длину программы, устраняя из неё несущественные пробелы и комментарии;
- если будет изменена кодировка программы, то это отразится только на лексическом анализаторе.

Принадлежность строки языку

Для лексического анализа применяют регулярные грамматики (мы будем рассматривать леволинейные).

Принадлежность строки языку

Для лексического анализа применяют регулярные грамматики (мы будем рассматривать леволинейные).
Задача: проверить принадлежность строки $a_1 \dots a_n \perp$ левосторонней грамматике G.

Принадлежность строки языку

Для лексического анализа применяют регулярные грамматики (мы будем рассматривать леволинейные).
Задача: проверить принадлежность строки $a_1 \dots a_n \perp$ левосторонней грамматике G.
Алгоритм проверки принадлежности строки языку:

- первый символ строки $a_1 a_2 \dots a_n \perp$ заменяем нетерминалом A, для которого в грамматике есть правило вывода $A \to a_1$;
- затем, пока не считаем символ $\perp$, выполняем следующие шаги: полученный на предыдущем шаге нетерминал A и расположенный непосредственно справа терминал a_i заменяем нетерминалом B, для которого в грамматике есть правило вывода $B \to A a_i$.

Принадлежность строки языку

Возможные варианты по ходу алгоритма:

- прочитана вся строка; на последнем шаге свертка произошла к символу S. Тогда $a_1 a_2 \dots a_n \perp \in L(G)$;
- прочитана вся строка; на последнем шаге свертка произошла к символу, отличному от S. Тогда $a_1 a_2 \dots a_n \perp \notin L(G)$;
- на некотором шаге не нашлось нужной «свертки», то есть не нашлось подходящего правила вывода $B \to A a_i$. Тогда $a_1 a_2 \dots a_n \perp \notin L(G)$;
- на некотором шаге работы алгоритма оказалось, что есть более одной подходящей свертки, то есть в грамматике разные нетерминалы имеют правила вывода с одинаковыми правыми частями. Это говорит о недетерминированности разбора.

Лексический анализатор

Рассмотрим грамматику: $G = (\{S, A, B, C\}, \{a, b, \perp\}, S, P)$

P:

$S \to C\perp$

$C \to Ab \mid Ba$

$A \to a \mid Ca$

$B \to b \mid Cb$

$$abaa\perp \Rightarrow Abaa\perp \Rightarrow Caa\perp \Rightarrow Aa\perp,$$

Нет подходящей продукции $\Rightarrow abaa \notin L(G)$.

$$abab\perp \Rightarrow Abab\perp \Rightarrow Cab\perp \Rightarrow Ab\perp \Rightarrow C\perp \Rightarrow S,$$

Свернули строку к S $\Rightarrow abab \in L(G)$.

Лексический анализатор

Рассмотрим грамматику: $G = (\{S, A, B, C\}, \{a, b, \perp\}, S, P)$

P:

$S \to C\perp$

$C \to Ab \mid Ba$

$A \to a \mid Ca$

$B \to b \mid Cb$

Матрица переходов:

	a	b	$\perp$
C	A	B	S
A	-	C	-
B	C	-	-
S	-	-	-

$$abaa\perp \Rightarrow Abaa\perp \Rightarrow Caa\perp \Rightarrow Aa\perp,$$

Нет подходящей продукции $\Rightarrow abaa \notin L(G)$.

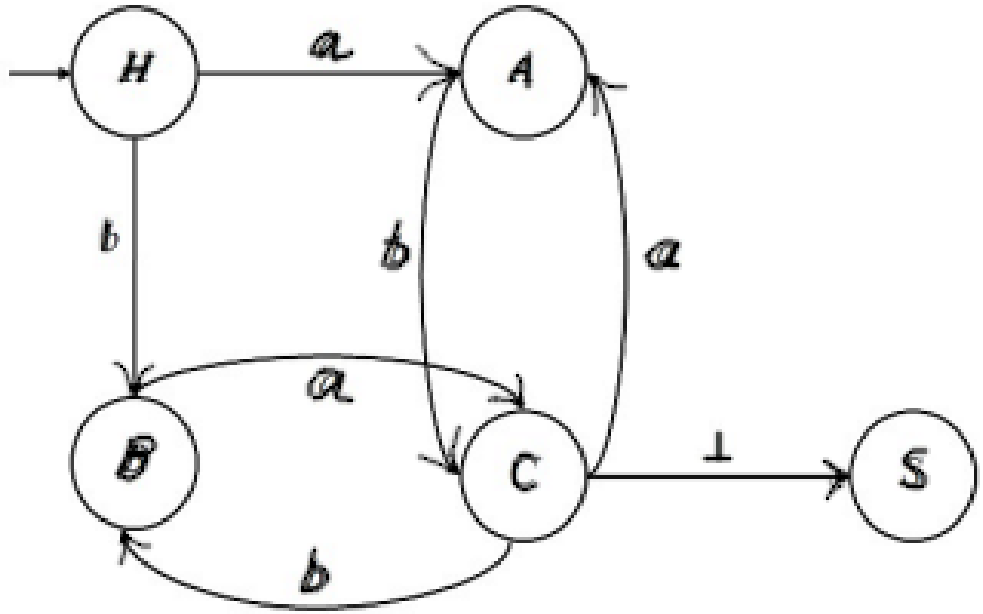
$$abab\perp \Rightarrow Abab\perp \Rightarrow Cab\perp \Rightarrow Ab\perp \Rightarrow C\perp \Rightarrow S,$$

Свернули строку к S $\Rightarrow abab \in L(G)$.

Лексический анализатор

Рассмотрим грамматику: $G = (\{S, A, B, C\}, \{a, b, \perp\}, S, P)$

Диаграмма состояний:



Матрица переходов:

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

P:

$S \to C\perp$

$C \to Ab \mid Ba$

$A \to a \mid Ca$

$B \to b \mid Cb$

$$abaa\perp \Rightarrow Abaa\perp \Rightarrow Caa\perp \Rightarrow Aa\perp,$$

Нет подходящей продукции $\Rightarrow abaa \notin L(G)$.

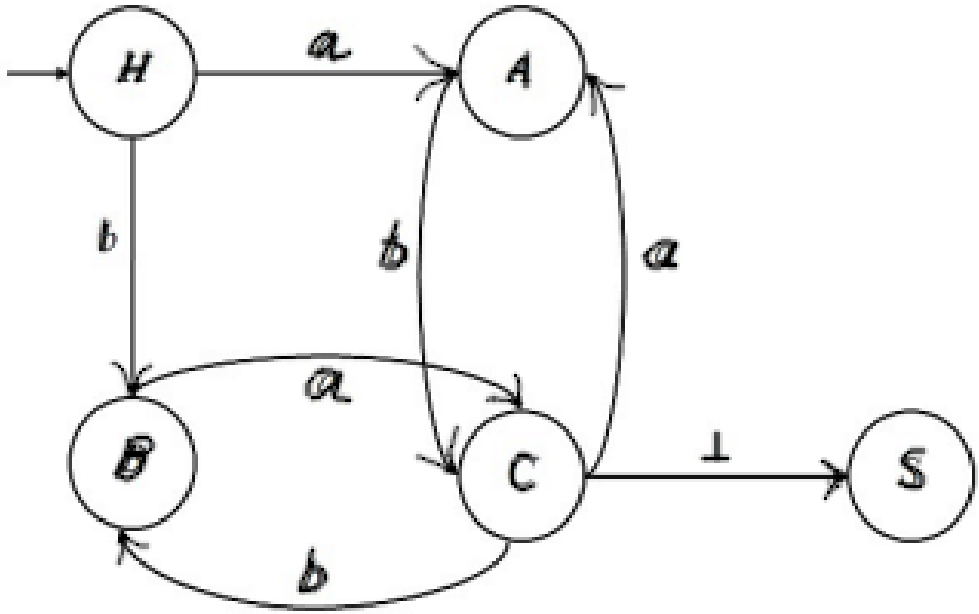
$$abab\perp \Rightarrow Abab\perp \Rightarrow Cab\perp \Rightarrow Ab\perp \Rightarrow C\perp \Rightarrow S,$$

Свернули строку к S $\Rightarrow abab \in L(G)$.

Лексический анализатор

Рассмотрим грамматику: $G = (\{S, A, B, C\}, \{a, b, \perp\}, S, P)$

Диаграмма состояний:



Матрица переходов:

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

P:

$S \to C\perp$

$C \to Ab \mid Ba$

$A \to Ha \mid Ca$

$B \to Hb \mid Cb$

$$Habaa\perp \Rightarrow Abaa\perp \Rightarrow Caa\perp \Rightarrow Aa\perp,$$

Нет подходящей продукции $\Rightarrow abaa \notin L(G)$.

$$Habab\perp \Rightarrow Abab\perp \Rightarrow Cab\perp \Rightarrow Ab\perp \Rightarrow C\perp \Rightarrow S,$$

Свернули строку к S $\Rightarrow abab \in L(G)$.

О недетерминированном разборе

- Как говорилось ранее, при проведении свёртки строки по грамматике, возможно возникновение неоднозначности.
- Но! Одним из важных результатов теории автоматов является эквивалентность класса **недетерминированных** и **детерминированных конечных автоматов**.
- Поэтому, при возникновении неоднозначности, диаграмму переходов необходимо детерминизировать.
- Таким образом, проблема недетерминированного разбора в лексическом анализе является полностью решённой.

О недетерминированном разборе

- Как говорилось ранее, при проведении свёртки строки по грамматике, возможно возникновение неоднозначности.
- Но! Одним из важных результатов теории автоматов является эквивалентность класса **недетерминированных** и **детерминированных конечных автоматов**.
- Поэтому, при возникновении неоднозначности, диаграмму переходов необходимо детерминизировать.
- Таким образом, проблема недетерминированного разбора в лексическом анализе является полностью решённой.

О недетерминированном разборе

- Как говорилось ранее, при проведении свёртки строки по грамматике, возможно возникновение неоднозначности.
- Но! Одним из важных результатов теории автоматов является эквивалентность класса **недетерминированных** и **детерминированных конечных автоматов**.
- Поэтому, при возникновении неоднозначности, диаграмму переходов необходимо детерминизировать.
- Таким образом, проблема недетерминированного разбора в лексическом анализе является полностью решённой.

Реализация

Строим программу по диаграмме состояний:

```
#include <iostream>
using namespace std;

// Символ конца строки
#define EOF '&'
//Множество состояний
enum class State { H, A, B,
    C, S, ER };
// Текущий символ
char curr='\0';
// текущее состояние
State state = State::H;

void nextChar() {
   cin >> curr;
}
void H() {
   if (curr == 'a') {
      nextChar();
      state = State::A;
   } else if (curr == 'b') {
      nextChar();
      state = State::B;
   } else state = State::ER;
}
```

```
void A() {
   if (curr == 'b') {
      nextChar();
      state = State::C;
   } else state = State::ER;
}
void B() {
   if (curr == 'a') {
      nextChar();
      state = State::C;
   } else state = State::ER;
}
void C() {
   if (curr == 'a') {
      nextChar();
      state = State::A;
   }
   else if (curr == 'b') {
      nextChar();
      state = State::B;
   }
   else if (curr == EOF)
      state = State::S;
   else
      state = State::ER;
}
```

```
bool check() {
   nextChar();
   while (state!=State::ER &&
         state!=State::S){
      switch (state) {
         case State::H: H(); break;
         case State::A: A(); break;
         case State::B: B(); break;
         case State::C: C(); break;
      }
   }
   if (state == State::ER)
      return false;
   if (state == State::S)
      return true;
}

int main()
{
   cout << check();
}
```

Лексический анализатор для М-языка

Вход: исходная программа.
Выход: список лексем.
Типы лексем:

- служебные слова – 1,
- ограничители – 2,
- константы – 3,
- идентификаторы – 4.

Каждой лексеме сопоставляется пара:
(тип_лексемы, указатель_на_информацию_о_ней).
Далее именно эту пару будем называть **лексемой**.
В списке выше после тире указан номер класса. Это значение будет занесено в поле тип_лексемы для соответствующих лексем.

Лексический анализатор для M-языка

Получим грамматику:

$R \to$ **begin** $|$ **end** $| \ldots |$ **not**

$D \to ; | : | := | \ldots | / | *$

$N \to 0 \mid 1 \mid \ldots \mid 9 \mid N0 \mid \ldots \mid N9$

$I \to a \mid b \mid \ldots \mid z \mid Ia \mid \ldots \mid Iz \mid I0 \mid \ldots \mid I9$

Лексический анализатор для M-языка

Цель анализатора — не просто выявить принадлежность строки языку, а выделить лексему и передать её для дальнейшей обработки.
То есть, появляются дополнительные действия для формирования лексем, которые надо выполнить, при разборе строки.
Для этого добавим в дуги метки действий $D_1, D_2 \ldots, D_n$:

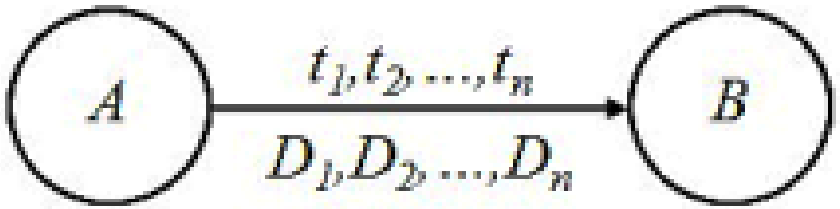


Диаграмма состояний для M-языка

Введём обозначения:

- buf – буфер для накопления символов лексемы;
- c – очередной входной символ;
- d – переменная для формирования числового значения константы;
- TW – таблица служебных слов M-языка;
- TD – таблица ограничителей M-языка;
- TID – таблица идентификаторов анализируемой программы;
- TNUM – таблица чисел-констант, используемых в программе.

Таблицы TW и TD заполнены заранее. TID и TNUM формируются в процессе анализа.
Пусть tabl – имя типа таблиц, ptabl – указатель на tabl.

Диаграмма состояний для M-языка

Определим функции:

- void clear(); – очистка буфера buf;
- void add(); – добавление символа c в конец буфера buf;
- int look(ptabl T); – поиск в таблице T лексемы из буфера buf; результат: индекс лексемы в таблице либо -1, если такой лексемы в таблице T нет;
- int putl(ptabl T); – запись в таблицу T лексемы из буфера buf, если ее там не было; результат: индекс добавленного элемента;

Диаграмма состояний для M-языка

Определим функции:

- int putnum(); – запись в TNUM константы из d, если ее там не было; результат: индекс добавленного элемента;
- void makelex(int k, int i); – формирование и вывод внутреннего представления лексемы; k – номер класса, i – номер в классе;
- void gc(); – функция, читающая из входного потока очередной символ исходной программы и заносящая его в переменную c;

Диаграмма состояний для M-языка

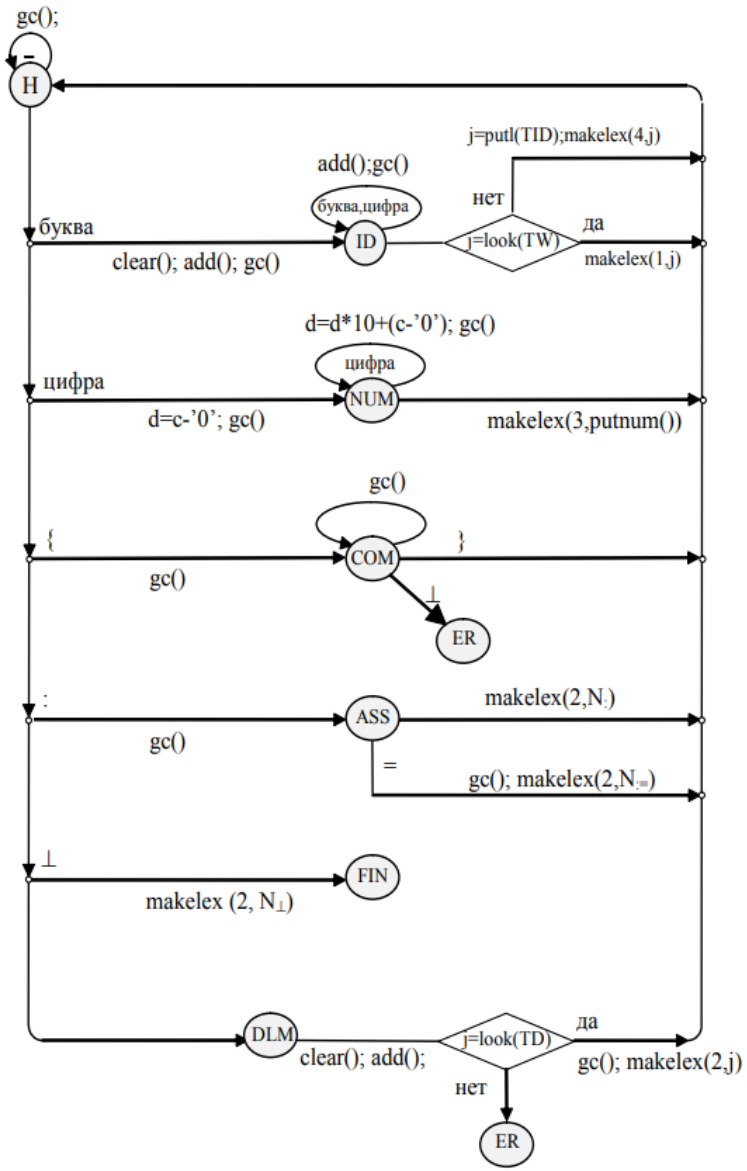


Диаграмма состояний для M-языка

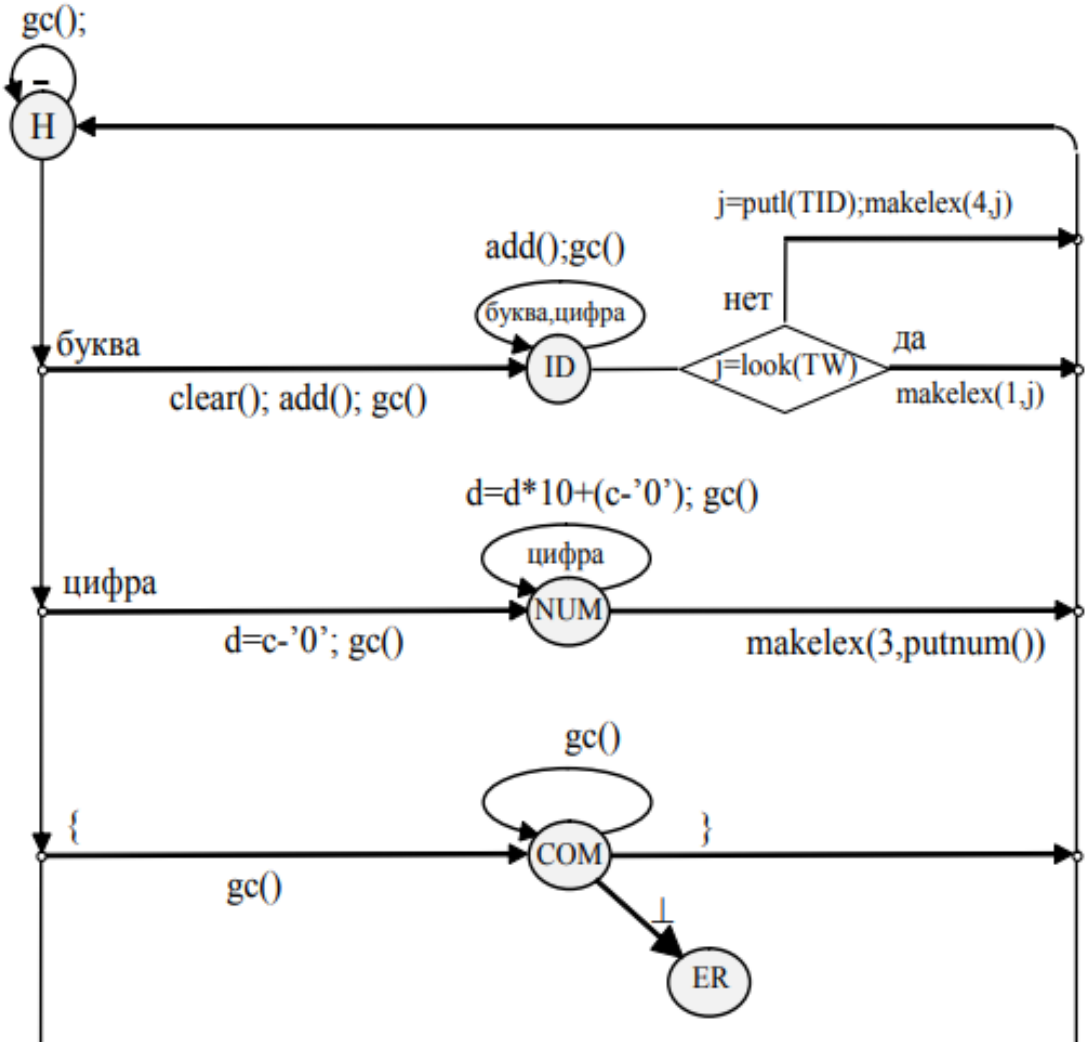
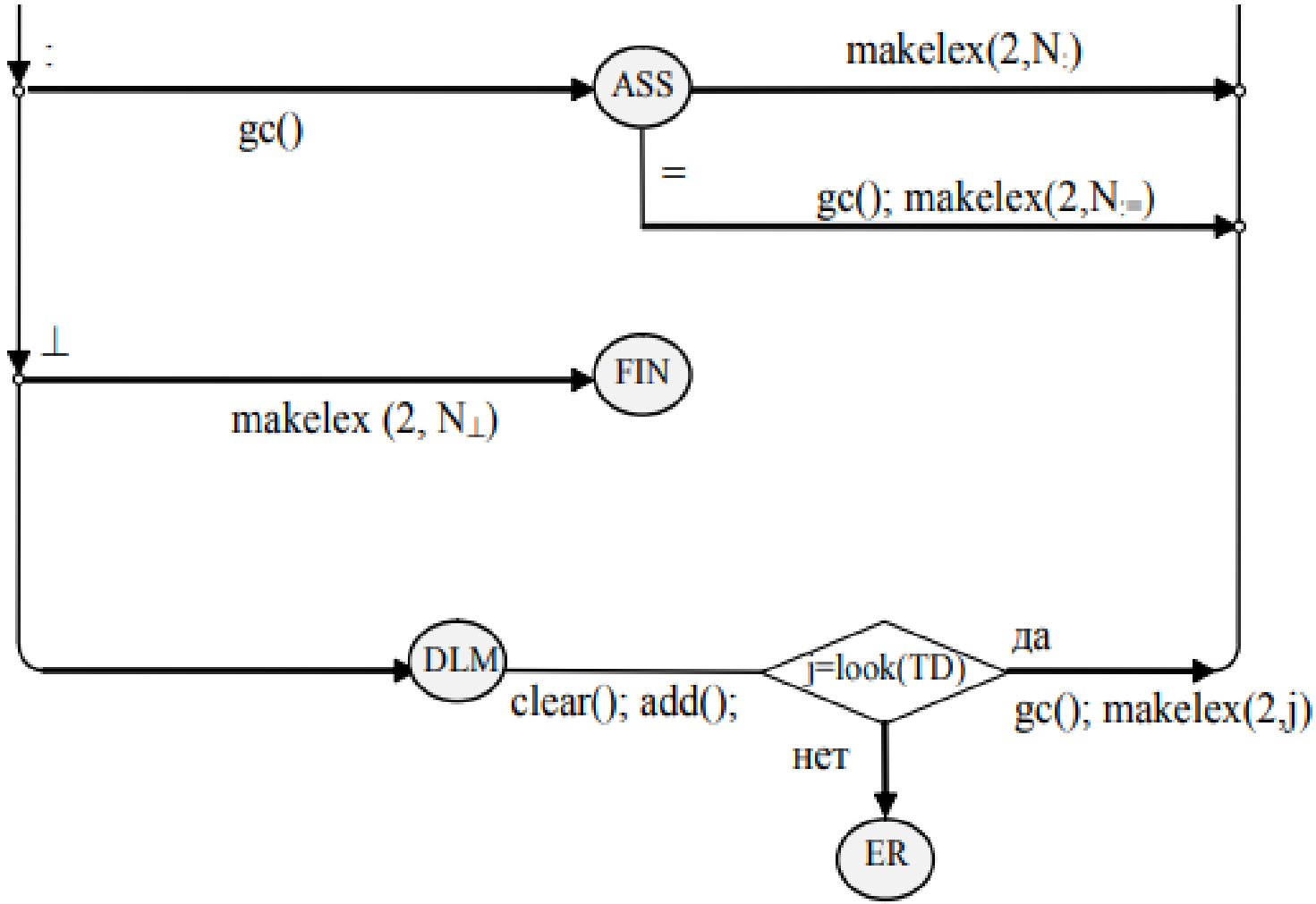


Диаграмма состояний для M-языка



Результат работы лексического анализатора

```
while a < b do begin
  {if beleberda just commentary
  write(a);}
  a := a + 1;
  c := c * a
end  &
```

⇓

(0, 9) (3, 0) (1, 7) (3, 1) (0, 10) (0, 4) (3, 0) (1, 3) (3, 0) (1, 10) (2, 2) (1, 1)
(3, 2) (1, 3) (3, 2) (1, 12) (3, 0) (0, 5) (0, 18)

Или для большей наглядности сопоставим id лексемы значение из таблицы лексем.

⇓

(0, while) (3, a) (1, <) (3, b) (0, do) (0, begin) (3, a) (1, :=) (3, a)
(1, +) (2, 1) (1, ;) (3, c) (1, :=) (3, c) (1, *) (3, a) (0, end) (0, &)

Результат работы лексического анализатора

```
while a < b do begin
  {if beleberda just commentary
  write(a);}
  a := a + 1;
  c := c * a
end  &
```

⇓

(0, 9) (3, 0) (1, 7) (3, 1) (0, 10) (0, 4) (3, 0) (1, 3) (3, 0) (1, 10) (2, 2) (1, 1)
(3, 2) (1, 3) (3, 2) (1, 12) (3, 0) (0, 5) (0, 18)

Или для большей наглядности сопоставим id лексемы значение из таблицы лексем.

⇓

(0, while) (3, a) (1, <) (3, b) (0, do) (0, begin) (3, a) (1, :=) (3, a)
(1, +) (2, 1) (1, ;) (3, c) (1, :=) (3, c) (1, *) (3, a) (0, end) (0, &)

Результат работы лексического анализатора

```
while a < b do begin
  {if beleberda just commentary
  write(a);}
  a := a + 1;
  c := c * a
end  &
```

⇓

(0, 9) (3, 0) (1, 7) (3, 1) (0, 10) (0, 4) (3, 0) (1, 3) (3, 0) (1, 10) (2, 2) (1, 1)
(3, 2) (1, 3) (3, 2) (1, 12) (3, 0) (0, 5) (0, 18)

Или для большей наглядности сопоставим id лексемы значение из таблицы лексем.

⇓

(0, while) (3, a) (1, <) (3, b) (0, do) (0, begin) (3, a) (1, :=) (3, a)
(1, +) (2, 1) (1, ;) (3, c) (1, :=) (3, c) (1, *) (3, a) (0, end) (0, &)

Что дальше?

- По диаграмме состояний реализовать программу. Опустим этот момент в презентации. Пример можно посмотреть в книжке.

Что дальше?

- По диаграмме состояний реализовать программу. Опустим этот момент в презентации. Пример можно посмотреть в книжке.
- Посмотреть видео с моей реализацией лексического анализатора.

Что дальше?

- По диаграмме состояний реализовать программу. Опустим этот момент в презентации. Пример можно посмотреть в книжке.
- Посмотреть видео с моей реализацией лексического анализатора.
- Сделать лабораторную работу №1 по лексическому анализу.

Лабораторная работа №1

- Разбивка по вариантам и сами варианты будут выложены в курсе в moodle. И туда же нужно прикрепить решения.
- Что требуется? — Нарисовать диаграмму состояний. Написать программу по этой диаграмме состояний.
- Дедлайн: 4 недели.
- Когда сдавать? — На следующих парах или online(назначим созвон на один из вечеров).