Конечные автоматы

Романенко Владимир Васильевич,
к.т.н., доцент каф. АСУ ТУСУР

Программирование ДКА и ДМПА

Чтобы явно реализовать в программе ДКА или ДМПА, достаточно:

1) описать в программе структуры данных, описывающих все компоненты автомата;

2) реализовать в коде управляющее устройство автомата;

3) реализовать код для выполнения внедрённых действий.

Программирование ДКА

Сначала рассмотрим способы реализации в программе на ЯВУ ДКА.

- Обозначим количество состояний автомата

$$N = \#Q.$$

Тогда состояния будут иметь номера от 0 до $N - 1$ (или, если это удобно, от 1 до N).

- Количество элементов алфавита обозначим

$$M = \#\Sigma.$$

- Количество элементов множества заключительных состояний обозначим

$$K = \#F.$$

Программирование ДКА

Пример: ДКА $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2\}$;
- $\Sigma = \{a\text{-}z, A\text{-}Z, \_, 0\text{-}9, ,, \sqcup\}$;
- $\delta = \{((q_0, \sqcup), (q_0, \langle\rangle)), ((q_0, a\text{-}zA\text{-}Z\_), (q_1, \langle A_1\rangle)),$
  $((q_1, \sqcup), (q_2, \langle A_3\rangle)) , ((q_1, a\text{-}zA\text{-}Z\_), (q_1, \langle A_2\rangle)),$
  $((q_1, 0\text{-}9), (q_1, \langle A_2\rangle)), ((q_1, ,), (q_0, \langle A_3\rangle)),$
  $((q_2, \sqcup), (q_2, \langle\rangle)), ((q_2, ,), (q_0, \langle\rangle))\}$;
- $q_0 = q_0$;
- $F = \{q_1, q_2\}$.

Здесь $N = 3$, M – зависит от реализации, $K = 2$.

Программирование ДКА

Как описать множество состояний *Q*?

- Если состояния именованные, то они описываются в массиве строк, списке строк или любой другой подобной структуре, например:

```
const N = 3;
const states: array [0..N-1] of string = ('q0', 'q1', 'q2');
```

```
const int N = 3;
char states[N][3] = {"q0", "q1", "q2"};
```

```
string[] states = {"q0", "q1", "q2"};
```

Программирование ДКА

Как описать множество состояний Q?

- Если состояния нумерованные, то можно их специально не описывать, а просто положить $q_0 = 0$, $q_1 = 1$, $q_2 = 2$:

```
const N = 3;
const int N = 3;
```

Программирование ДКА

Как описать алфавит Σ?

- Это может быть просто массив символов:

```
const M = 68;
const alphabet: array [1..M] of char = ('_', 'a', 'b', ..., 'z', 'A', 'B', ..., 'Z', '0', '1', ..., '9', ',', #32, #9, #10, #13);
const int M = 68;
char alphabet[M] = {'_', 'a', 'b', ..., 'z', 'A', 'B', ..., 'Z', '0', '1', ..., '9', ',', ' ', '\t', '\r', '\n'};
char[] alphabet = {'_', 'a', 'b', ..., 'z', 'A', 'B', ..., 'Z', '0', '1', ..., '9', ',', ' ', '\t', '\r', '\n'};
```

Программирование ДКА

Как описать алфавит Σ?

- Это может быть массив строк:

```
const M = 4;
const alphabet: array [1..M] of string = ('_ab…zAB…Z', '0123456789', ',', #32#9#10#13);
const int M = 4;
char alphabet[M][54] = {"_ab…zAB…Z", "0123456789", ",", " \t\r\n"};
string[] alphabet = {"_ab…zAB…Z", "0123456789", ",", " \t\r\n"};
```

Программирование ДКА

Как описать алфавит Σ?

- Это может быть массив множеств символов:

```
const M = 4;
const alphabet: array [1..M] of set of char = (['_', 'a'.. 'z', 'A'.. 'Z'], ['0'.. '9'], [','], [#32, #9, #10, #13]);
```

```
const int M = 4;
set<char> alphabet[M] = …
HashSet<char>[] alphabet = …
…
```

Программирование ДКА

Как описать алфавит Σ?

- Можно использовать диапазоны:

```
const M = 6;
const alphabet: array [1..M] of string = ('_', 'a-z', 'A-Z', '0-9', ',', #32#9#10#13);
```

```
const int M = 6;
char alphabet[M][5] = {"_", "a-z", "A-Z", "0-9", ",", " \t\r\n"};
```

```
string[] alphabet = {"_", "a-z", "A-Z", "0-9", ",", " \t\r\n"};
```

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть матрица строк размером $N \times M$:

```
const delta: array [0..N-1, 1..M] of string = (
        ('q1', 'error', 'error', 'q0'),
        ('q1', 'q1', 'q0', 'q2'),
        ('error', 'error', 'q0', 'q2')
);
```

	a-z, A-Z, _	0-9	,	␣
q_0	$q_1, \langle A_1 \rangle$			q_0
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$
q_2			q_0	q_2

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть матрица строк размером $N \times M$:

```
char delta[N][M][6] = {
        {"q1", "error", "error", "q0"},
        {"q1", "q1", "q0", "q2"},
        {"error", "error", "q0", "q2"}
};
```

	a-z, A-Z, _	0-9	,	␣
q_0	$q_1, \langle A_1 \rangle$			q_0
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$
q_2			q_0	q_2

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть матрица строк размером $N \times M$:

```
string[,] delta = {
        {"q1", "error", "error", "q0"},
        {"q1", "q1", "q0", "q2"},
        {"error", "error", "q0", "q2"}
};
```

	a-z, A-Z, _	0-9	,	␣
q_0	$q_1, \langle A_1 \rangle$			q_0
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$
q_2			q_0	q_2

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть матрица целых чисел размером $N \times M$:

```
const ERROR = -1;
const delta: array [0..N-1, 1..M] of integer = (
        (1, ERROR, ERROR, 0),
        (1, 1, 0, 2),
        (ERROR, ERROR, 0, 2)
);
```

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть матрица целых чисел размером $N \times M$:

```
const int ERROR = -1;
int delta[N][M] = {
        {1, ERROR, ERROR, 0},
        {1, 1, 0, 2},
        {ERROR, ERROR, 0, 2}
};
```

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть матрица целых чисел размером $N \times M$:

```
const int ERROR = -1;
int[,] delta = {
        {1, ERROR, ERROR, 0},
        {1, 1, 0, 2},
        {ERROR, ERROR, 0, 2}
};
```

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть массив дуг графа:

```
type Edge = record
        from: string/integer;
        elem: integer/char/string/set of char;
        to: string/integer;
end;
const delta: array [1..Z] of Edge = …;
…
```

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть массив дуг графа:

```
struct Edge {
        char[L1]/integer from;
        integer/char/char[L2]/set<char> elem;
        char[L1]/integer to;
};
Edge delta[Z] = …;
…
```

Программирование ДКА

Как описать функцию переходов δ?

- Это может быть массив дуг графа:

```
struct Edge {
        public string/integer from;
        public integer/char/string/HashSet<char> elem;
        public string/integer to;
};
Edge[] delta = …;
…
```

Программирование ДКА

Дуги графа:

$(q_0, \sqcup, q_0)$,

$(q_0, a\text{-}zA\text{-}Z\_, q_1)$,

$(q_1, \sqcup, q_2)$,

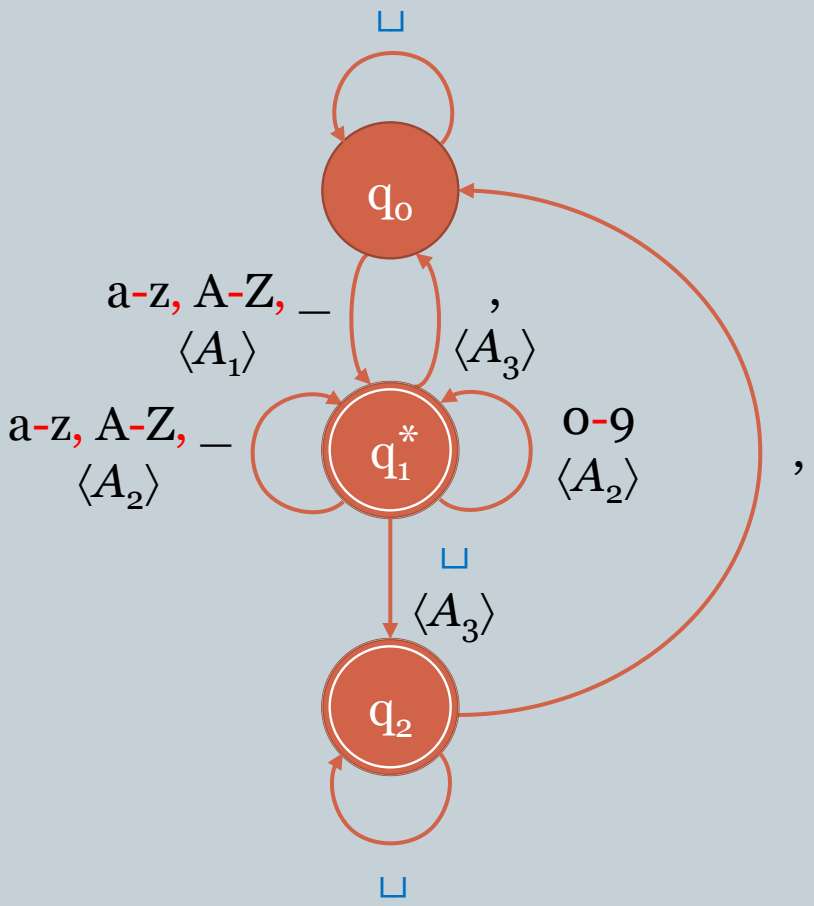
$(q_1, a\text{-}zA\text{-}Z\_, q_1)$,

$(q_1, 0\text{-}9, q_1)$,

$(q_1, ,, q_0)$,

$(q_2, \sqcup, q_2)$,

$(q_2, ,, q_0)$



Программирование ДКА

Как описать начальное состояние q_0?

- Это может быть строка:

```
const q0 = 'q0';
char q0[3] = "q0";
string q0 = "q0";
```

- Это может быть целое число:

```
const q0 = 0;
int q0 = 0;
```

Программирование ДКА

Как описать множество заключительных состояний *F*?

- Если состояния именованные, то используем массив строк, список строк и т.п.:

```
const K = 2;
const final: array [1..K] of string = ('q1', 'q2');
```

```
const int K = 3;
char final[K][3] = {"q1", "q2"};
```

```
string[] final = {"q1", "q2"};
```

Программирование ДКА

Как описать множество заключительных состояний *F*?

- Если состояния нумерованные, то используем массив (список и т.п.) целых чисел:

```
const K = 2;
const final: array [1..K] of integer = (1, 2);
```

```
const int K = 2;
int final[K] = {1, 2};
int[] final = {1, 2};
```

Программирование ДКА

Сокращённый вариант (без множества F).

- Если в функцию переходов будет добавлен маркер конца входной цепочки $\perp$, то его также нужно добавить в алфавит. Для его представления можно использовать любой непечатный символ (символ с кодом менее 32), не входящий в исходный алфавит языка.

Примечание. Маркер конца цепочки нужен в любом случае. Удобно брать символ с кодом 0. В языке C/C++ такой символ уже имеется в конце каждой строки, как и в языке Pascal для типа PChar. В языке C# нужно его добавить к строке вручную. Либо эмулировать появление этого символа, когда закончилась входная строка или файл.

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Получим следующий алфавит:

```
const M = …;
const alphabet: array [1..M] of char = (…, #0);
```

или

```
const alphabet: array [1..M] of string = (…, #0);
```

или

```
const alphabet: array [1..M] of set of char = (…, [#0]);
```

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Получим следующий алфавит:

```
const int M = …;
char alphabet[M] = {…, '\0'};
```

или

```
char alphabet[M][…] = {…, "\0"};
```

или

```
char end = '\0';
set<char> alphabet[M] = {…, set<char>(&end, &end+1)}; // проще alphabet[M-1].insert('\0');
```

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Получим следующий алфавит:

```
const int M = …;
char[] alphabet = {…, '\0'};
```

или

```
string[] alphabet = {…, "\0"};
```

или

```
HashSet<char>[] alphabet = {…, new HashSet<char>
(new char[] { '\0' })};
```

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Получим следующую функцию переходов:

```
const delta: array [0..N-1, 1..M] of string = (
        ('q1', 'error', 'error', 'q0', 'error'),
        ('q1', 'q1', 'q0', 'q2', 'halt'),
        ('error', 'error', 'q0', 'q2', 'halt')
);
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Получим следующую функцию переходов:

```
char delta[N][M][6] = {
        {"q1", "error", "error", "q0", "error"},
        {"q1", "q1", "q0", "q2", "halt"},
        {"error", "error", "q0", "q2", "halt"}
};
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Получим следующую функцию переходов:

```
string[,] delta = {
        {"q1", "error", "error", "q0", "error"},
        {"q1", "q1", "q0", "q2", "halt"},
        {"error", "error", "q0", "q2", "halt"}
};
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Или в виде матрицы целых чисел:

```
const ERROR = -1;
const HALT = 99;
const delta: array [0..N-1, 1..M] of integer = (
        (1, ERROR, ERROR, 0, ERROR),
        (1, 1, 0, 2, HALT),
        (ERROR, ERROR, 0, 2, HALT)
);
```

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Или в виде матрицы целых чисел:

```
const int ERROR = -1;
const int HALT = 99;
int delta[N][M] = {
        {1, ERROR, ERROR, 0, ERROR},
        {1, 1, 0, 2, HALT},
        {ERROR, ERROR, 0, 2, HALT}
};
```

Программирование ДКА

Сокращённый вариант (без множества *F*).

- Или в виде матрицы целых чисел:

```
const int ERROR = -1;
const int HALT = 99;
int[,] delta = {
        {1, ERROR, ERROR, 0, ERROR},
        {1, 1, 0, 2, HALT},
        {ERROR, ERROR, 0, 2, HALT}
};
```

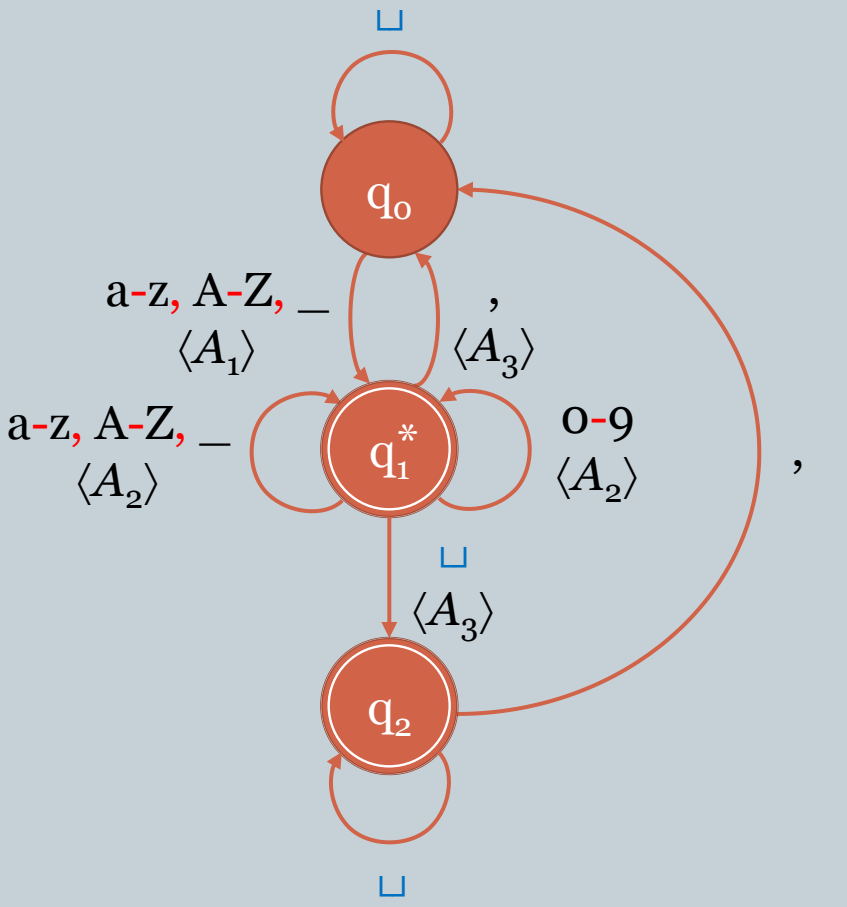
Программирование ДКА

Сокращённый вариант (без множества F).

- В виде массива или списка дуг графа, тогда добавятся следующие дуги:

$(q_1, \perp, \text{HALT})$,

$(q_2, \perp, \text{HALT})$



Программирование ДКА

Как включить в синтаксис действия?

- Если действия именованные, то их можно описать в массиве строк размером $N \times M$:

```
const actions: array [0..N-1, 1..M] of string = (
        ('a1', '', '', '', ''),
        ('a2', 'a2', 'a3', 'a3', 'a3'),
        ('', '', '', '', '')
);
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Как включить в синтаксис действия?

- Если действия именованные, то их можно описать в массиве строк размером $N \times M$:

```
char actions[N][M][6] = {
        {"a1", "", "", "", ""},
        {"a2", "a2", "a3", "a3", "a3"},
        {"", "", "", "", ""}
};
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Как включить в синтаксис действия?

- Если действия именованные, то их можно описать в массиве строк размером $N \times M$:

```
string[,] actions = {
        {"a1", "", "", "", ""},
        {"a2", "a2", "a3", "a3", "a3"},
        {"", "", "", "", ""}
};
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Как включить в синтаксис действия?

- Если действия нумерованные, то их можно описать в массиве целых чисел:

```
const actions: array [0..N-1, 1..M] of integer = (
        (1, 0, 0, 0, 0),
        (2, 2, 3, 3, 0),
        (0, 0, 0, 0, 0)
);
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Как включить в синтаксис действия?

- Если действия нумерованные, то их можно описать в массиве целых чисел:

```
int actions[N][M] = {
        {1, 0, 0, 0, 0},
        {2, 2, 3, 3, 3},
        {0, 0, 0, 0, 0}
};
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	q_1, $\langle A_1 \rangle$			q_0	
q_1	q_1, $\langle A_2 \rangle$	q_1, $\langle A_2 \rangle$	q_0, $\langle A_3 \rangle$	q_2, $\langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Как включить в синтаксис действия?

- Если действия нумерованные, то их можно описать в массиве целых чисел:

```
int[,] actions = {
        {1, 0, 0, 0, 0},
        {2, 2, 3, 3, 3},
        {0, 0, 0, 0, 0}
};
```

	a-z, A-Z, _	0-9	,	␣	⊥
q_0	$q_1, \langle A_1 \rangle$			q_0	
q_1	$q_1, \langle A_2 \rangle$	$q_1, \langle A_2 \rangle$	$q_0, \langle A_3 \rangle$	$q_2, \langle A_3 \rangle$	HALT, $\langle A_3 \rangle$
q_2			q_0	q_2	HALT

Программирование ДКА

Как включить в синтаксис действия?

- Если учесть, что размеры массивов delta и actions совпадают, можно их объединить в один массив:

```
const delta: array [0..N-1, 1..M, 1..2] of string = (
        (('q1', 'a1'), ('error', ''), ('error', ''), ('q0', ''), ('error', '')),
        (('q1', 'a2'), ('q1', 'a2'), ('q0', 'a3'), ('q2', 'a3'), ('halt', 'a3')),
        (('error', ''), ('error', ''), ('q0', ''), ('q2', ''), ('halt', ''))
);
const delta: array [0..N-1, 1..M, 1..2] of integer = (
        ((1, 1), (ERROR, 0), (ERROR, 0), (0, 0), (ERROR, 0)),
        ((1, 2), (1, 2), (0, 3), (2, 3), (HALT, 3)),
        ((ERROR, 0), (ERROR, 0), (0, 0), (q, 0), (HALT, 0))
);
```

Программирование ДКА

Как включить в синтаксис действия?

- Если учесть, что размеры массивов delta и actions совпадают, можно их объединить в один массив:

```
char delta[N][M][2][6] = {
        {{"q1", "a1"}, {"error", ""}, {"error", ""}, {"q0", ""}, {"error", ""}},
        {{"q1", "a2"}, {"q1", "a2"}, {"q0", "a3"}, {"q2", "a3"}, {"halt", "a3"}},
        {{"error", ""}, {"error", ""}, {"q0", ""}, {"q2", ""}, {"halt", ""}}
};
int delta[N][M][2] = {
        {{q, 1}, {ERROR, 0}, {ERROR, 0}, {0, 0}, {ERROR, 0}},
        {{q, 2}, {1, 2}, {0, 3}, {2, 3}, {HALT, 3}},
        {{ERROR, 0}, {ERROR, 0}, {0, 0}, {2, 0}, {HALT, 0}}
};
```

Программирование ДКА

Как включить в синтаксис действия?

- Если учесть, что размеры массивов delta и actions совпадают, можно их объединить в один массив:

```
string[,,] delta = {
        {{"q1", "a1"}, {"error", ""}, {"error", ""}, {"q0", ""}, {"error", ""}},
        {{"q1", "a2"}, {"q1", "a2"}, {"q0", "a3"}, {"q2", "a3"}, {"halt", "a3"}},
        {{"error", ""}, {"error", ""}, {"q0", ""}, {"q2", ""}, {"halt", ""}}
};
int[,,] delta = {
        {{q, 1}, {ERROR, 0}, {ERROR, 0}, {0, 0}, {ERROR, 0}},
        {{q, 2}, {1, 2}, {0, 3}, {2, 3}, {HALT, 3}},
        {{ERROR, 0}, {ERROR, 0}, {0, 0}, {2, 0}, {HALT, 0}}
};
```

Программирование ДМПА

Рассмотрим способы реализации в программе на ЯВУ ДМПА.

- Обозначим количество элементов алфавита магазина

$$G = \#\Gamma.$$

- Количество возможных сочетаний (элемент алфавита языка, элемент алфавита магазина) обозначим P.

	0-9, *e*	+, ∅	+, +	+, ×	×, ∅	×, +	×, ×	⊥, ∅	⊥, +	⊥, ×
q_0	q_1, *e*, $\langle A_1 \rangle$									
q_1	q_1, *e*, $\langle A_2 \rangle$	q_0, +, $\langle A_3 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, ×, $\langle A_3 \rangle$	q_0, +×, $\langle A_3 \rangle$	q_0, ×, $\langle A_4 \rangle$	HALT, $\langle A_3 \rangle$	q_2, *e*, $\langle A_5 \rangle$	q_2, *e*, $\langle A_5 \rangle$
q_2								HALT	q_2, *e*, $\langle A_6 \rangle$	q_2, *e*, $\langle A_6 \rangle$

Программирование ДМПА

Пример. ДМПА $P = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$:

- $Q = \{q_0, q_1, q_2\}$;
- $\Sigma = \{0\text{-}9, +, \times\}$;
- $\Gamma = \{+, \times\}$;
- $\delta = \{((q_0, 0\text{-}9, e), (q_1, e, \langle A_1 \rangle)), ((q_1, 0\text{-}9, e), (q_1, e, \langle A_2 \rangle)), ((q_1, +, \varnothing), (q_0, +, \langle A_3 \rangle)), ((q_1, +, +), (q_0, +, \langle A_4 \rangle)), ((q_1, +, \times), (q_0, +, \langle A_4 \rangle)), ((q_1, \times, \varnothing), (q_0, \times, \langle A_3 \rangle)), ((q_1, \times, +), (q_0, +\times, \langle A_3 \rangle)), ((q_1, \times, \times), (q_0, \times, \langle A_4 \rangle)), ((q_1, \perp, \varnothing), (\text{HALT}, e, \langle A_3 \rangle)), ((q_1, \perp, +), (q_2, e, \langle A_5 \rangle)), ((q_1, \perp, \times), (q_2, e, \langle A_5 \rangle)), ((q_2, \perp, \varnothing), (\text{HALT}, e, \langle\rangle)), ((q_2, \perp, +), (q_2, e, \langle A_6 \rangle)), ((q_2, \perp, \times), (q2, e, \langle A_6 \rangle))\}$;

Программирование ДМПА

Пример. ДМПА $P = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$:

- $q_0 = q_0$;
- $\gamma_0 = e$;
- $F = \{q_1, q_2\}$.

Здесь $N = 3$, M – зависит от реализации, $K = 2$, $G = 2$, $P = 10$.

	0-9, e	+, ∅	+, +	+, ×	×, ∅	×, +	×, ×	⊥, ∅	⊥, +	⊥, ×
q_0	q_1, e, $\langle A_1 \rangle$									
q_1	q_1, e, $\langle A_2 \rangle$	q_0, +, $\langle A_3 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, ×, $\langle A_3 \rangle$	q_0, +×, $\langle A_3 \rangle$	q_0, ×, $\langle A_4 \rangle$	HALT, $\langle A_3 \rangle$	q_2, e, $\langle A_5 \rangle$	q_2, e, $\langle A_5 \rangle$
q_2								HALT	q_2, e, $\langle A_6 \rangle$	q_2, e, $\langle A_6 \rangle$

Программирование ДМПА

Элементы Q, Σ, q_0, F описываются так же, как в случае ДКА (в данном случае без F лучше обойтись).

- Например, как описать алфавит Σ? Доступны все рассмотренные ранее способы:

```
const M = 4;
const alphabet: array [1..M] of string = ('0-9', '+', '*', #0);
```

```
const int M = 4;
char alphabet[M][4] = {"0-9", "+", "*", "\0"};
string[] alphabet = {"0-9", "+", "*", "\0"};
```

Программирование ДМПА

Как описать алфавит магазина Г?

- В данном случае достаточно массива символов:

```
const G = 2;
const stack_alphabet: array [1..G] of char = ('+', '*');
const int G = 2;
char stack_alphabet[G] = {'+', '*'};
char[] stack_alphabet = {'+', '*'};
```

Однако, при необходимости можно, как и для Σ, использовать строки, множества, диапазоны и т.д.

Программирование ДМПА

Как описать допустимые пары (a, z), где $a \in \Sigma \cup \{e\} \cup \{\perp\}$, $z \in \Gamma \cup \{e\}$?

- Можно использовать индексы из массивов alphabet и stack_alphabet:

```
const P = 10;

const IGNORE = -1;

const EMPTY = -2;

const pairs: array [1..P, 1..2] of integer = ((1, IGNORE),
(2, EMPTY), (2, 1), (2, 2), (3, EMPTY), (3, 1), (3, 2),
(4, EMPTY), (4, 1), (4, 2));
```

Программирование ДМПА

Как описать допустимые пары (a, z), где $a \in \Sigma \cup \{e\} \cup \{\perp\}$, $z \in \Gamma \cup \{e\}$?

- Можно использовать индексы из массивов alphabet и stack_alphabet:

```
const int P = 10;
const int IGNORE = -1;
const int EMPTY = -2;
int pairs[P][2] = {{1, IGNORE}, {2, EMPTY}, {2, 1}, {2, 2}, {3, EMPTY}, {3, 1}, {3, 2}, {4, EMPTY}, {4, 1}, {4, 2});
```

Программирование ДМПА

Как описать допустимые пары (a, z), где $a \in \Sigma \cup \{e\} \cup \{\perp\}$, $z \in \Gamma \cup \{e\}$?

- Можно использовать индексы из массивов alphabet и stack_alphabet:

```
const int P = 10;
const int IGNORE = -1;
const int EMPTY = -2;
int[,] pairs = {{1, IGNORE}, {2, EMPTY}, {2, 1}, {2, 2},
{3, EMPTY}, {3, 1}, {3, 2}, {4, EMPTY}, {4, 1}, {4, 2}};
```

Программирование ДМПА

- Упрощённый вариант. Можно не задавать массивы alphabet и stack_alphabet, а только массив pairs:

```
const P = 10;

const pairs: array [1..P, 1..2] of string = (('0-9', #1), ('+', #0), ('+', '+'), ('+', '*'), ('*', #0), ('*', '+'), ('*', '*'), (#0, #0), (#0, '+'), (#0, '*'));
```

Также можно использовать множества символов и т.п.

Программирование ДМПА

- Упрощённый вариант. Можно не задавать массивы alphabet и stack_alphabet, а только массив pairs:

```
const int P = 10;
char pairs[P][2][4] = {{"0-9", "\1"}, {"+", "\0"},
{"+", "+"}, {"+", "*"}, {"*", "\0"}, {"*", "+"}, {"*", "*"},
{"\0", "\0"}, {"\0", "+"}, {"\0", "*"}};
```

Также можно использовать множества символов и т.п.

Программирование ДМПА

- Упрощённый вариант. Можно не задавать массивы alphabet и stack_alphabet, а только массив pairs:

```
const int P = 10;
string[,] pairs = {{"0-9", "\1"}, {"+", "\0"},
{"+", "+"}, {"+", "*"}, {"*", "\0"}, {"*", "+"}, {"*", "*"},
{"\0", "\0"}, {"\0", "+"}, {"\0", "*"}};
```

Также можно использовать множества символов и т.п.

Программирование ДМПА

Как описать функцию переходов δ?

- Это может быть матрица строк размером $N \times P \times 2$:

```
const delta: array [0..N-1, 1..P, 1..2] of string = (
        (('q1', ''), ('error', ''), ..., ('error', '')),
        (('q1', ''), ('q0', '+'), ('q0', '+'), ('q0', '+'),
          ('q0', '*'), ('q0', '+*'), ('q0', '*'),
          ('halt', ''), ('q2', ''), ('q2', '')),
        (('error', ''), ..., ('error', ''), ('halt', ''), ('q2', ''), ('q2', ''))
);
```

	0-9, *e*	+, ∅	+, +	+, ×	×, ∅	×, +	×, ×	⊥, ∅	⊥, +	⊥, ×
q_0	q_1, *e*, $\langle A_1 \rangle$									
q_1	q_1, *e*, $\langle A_2 \rangle$	q_0, +, $\langle A_3 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, ×, $\langle A_3 \rangle$	q_0, +×, $\langle A_3 \rangle$	q_0, ×, $\langle A_4 \rangle$	HALT, $\langle A_3 \rangle$	q_2, *e*, $\langle A_5 \rangle$	q_2, *e*, $\langle A_5 \rangle$
q_2								HALT	q_2, *e*, $\langle A_6 \rangle$	q_2, *e*, $\langle A_6 \rangle$

Программирование ДМПА

Как описать функцию переходов δ?

- Это может быть матрица строк размером $N \times P \times 2$:

```
char delta[N][P][2][6] = {
        {{"q1", ""}, {"error", ""}, ..., {"error", ""}},
        {{"q1", ""}, {"q0", "+"}, {"q0", "+"}, {"q0", "+"},
         {"q0", "*"}, {"q0", "+*"}, {"q0", "*"},
         {"halt", ""}, {"q2", ""}, {"q2", ""}},
        {{"error", ""}, ..., {"error", ""},
         {"halt", ""}, {"q2", ""}, {"q2", ""}}};
```

	0-9, *e*	+, ∅	+, +	+, ×	×, ∅	×, +	×, ×	⊥, ∅	⊥, +	⊥, ×
q_0	q_1, *e*, $\langle A_1 \rangle$									
q_1	q_1, *e*, $\langle A_2 \rangle$	q_0, +, $\langle A_3 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, ×, $\langle A_3 \rangle$	q_0, +×, $\langle A_3 \rangle$	q_0, ×, $\langle A_4 \rangle$	HALT, $\langle A_3 \rangle$	q_2, *e*, $\langle A_5 \rangle$	q_2, *e*, $\langle A_5 \rangle$
q_2								HALT	q_2, *e*, $\langle A_6 \rangle$	q_2, *e*, $\langle A_6 \rangle$

Программирование ДМПА

Как описать функцию переходов δ?

- Это может быть матрица строк размером $N \times P \times 2$:

```
string[,,] delta = {
        {{"q1", ""}, {"error", ""}, ..., {"error", ""}},
        {{"q1", ""}, {"q0", "+"}, {"q0", "+"}, {"q0", "+"},
         {"q0", "*"}, {"q0", "+*"}, {"q0", "*"},
         {"halt", ""}, {"q2", ""}, {"q2", ""}},
        {{"error", ""}, ..., {"error", ""},
         {"halt", ""}, {"q2", ""}, {"q2", ""}}};
```

	0-9, *e*	+, ∅	+, +	+, ×	×, ∅	×, +	×, ×	⊥, ∅	⊥, +	⊥, ×
q_0	$q_1, e, \langle A_1 \rangle$									
q_1	$q_1, e, \langle A_2 \rangle$	$q_0, +, \langle A_3 \rangle$	$q_0, +, \langle A_4 \rangle$	$q_0, +, \langle A_4 \rangle$	$q_0, \times, \langle A_3 \rangle$	$q_0, +\times, \langle A_3 \rangle$	$q_0, \times, \langle A_4 \rangle$	HALT, $\langle A_3 \rangle$	$q_2, e, \langle A_5 \rangle$	$q_2, e, \langle A_5 \rangle$
q_2								HALT	$q_2, e, \langle A_6 \rangle$	$q_2, e, \langle A_6 \rangle$

Программирование ДМПА

Как описать функцию переходов δ?

- Это может быть матрица целых чисел, если состояния нумерованные, а для элементов алфавита магазина указываются их индексы в массиве stack_alphabet.
- Если в автомат добавить действия, то, аналогично ДКА, их можно описать или в отдельном массиве actions, или также в массиве delta. В этом случае его размер станет $N \times P \times 3$.
- Как и в случае ДКА, можно хранить массив дуг графа функции переходов. В структуру Edge нужно будет только добавить поле action.
- Можно описать структуру, соответствующую результату функции переходов, и хранить массив таких структур размера $N \times P$, и т.д.

Программирование ДМПА

Пример такой структуры:

```
type DeltaCell = record
        state: string/integer;
        stack: integer/char/string;
        action: string/integer;
end;
const delta: array [0..N-1, 1..P] of DeltaCell = …;
…
```

Программирование ДМПА

Пример такой структуры:

```
struct DeltaCell {
        char[L1]/integer state;
        integer/char/char[L2] stack;
        char[L3]/integer action;
};
DeltaCell delta[N][P] = …;
…
```

Программирование ДМПА

Пример такой структуры:

```
struct DeltaCell {
        public string/integer state;
        public integer/char/string stack;
        public string/integer action;
};
DeltaCell[,] delta = …;
…
```

Программирование ДКА и ДМПА

В коде программы необходимо реализовать:

1. Алгоритм работы управляющего устройства конечного автомата (приведён в методическом пособии). Это должна быть отдельная функция или процедура, инвариантная к анализируемому языку L.
2. Внедрённые в синтаксис автомата действия. Реализуются в виде отдельной функции. Типы параметров и возвращаемого значения были оговорены ранее.

Разбор по символам и по лексемам

Возможны три варианта разбора цепочки:

- Посимвольный. Автомат считывает входную цепочку посимвольно, т.е. считывающая головка может передвинуться только на один символ за один такт работы автомата. Алфавит языка включает только отдельные символы.
- По лексемам. Алфавит языка включает некоторые лексемы (ключевые слова), и автомат за один такт считывает одну лексему.
- Смешанный.

Разбор по символам и по лексемам

Пример. Пусть язык *L* описывает вложенные операторы языка Pascal «**begin end**;». Учитывая, что необходимо проверять их парность, используем ДМПА с посимвольным разбором. Операторы отделяются друг от друга разделительными символами (пробелами, табуляциями, знаками возврата каретки и перехода на новую строку) в произвольном количестве, но не менее одного (в таблице обозначены символом подчеркивания). Также пробелы могут окружать знак «;».

Разбор по символам и по лексемам

	b, *e*	e, b	g, *e*	i, *e*	n, *e*	d, *e*	;, *e*	␣, *e*	⊥, ∅
q_0	q_1, b	q_6, *e*						q_0, *e*	HALT
q_1		q_2, b							
q_2			q_3, *e*						
q_3				q_4, *e*					
q_4					q_5, *e*				
q_5								q_0, *e*	
q_6					q_7, *e*				
q_7						q_8, *e*			
q_8							q_0, *e*	q_8, *e*	

Примеры

Получили ДМКА $P = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$;
- $\Sigma = \{b, e, g, i, n, d, ;, \sqcup\}$;
- $\delta = \{\ldots\}$;
- $q_0 = q_0$;
- $F = \{q_0\}$.

Разбор по символам и по лексемам

Для разбора по лексемам алфавит языка разбивается на три части:

1. Подмножество символов-разделителей (или пробельных символов) $\Sigma_S \subset \Sigma$.

2. Подмножество символов, являющихся знаками пунктуации $\Sigma_P \subset \Sigma$.

3. Подмножество символов, из которых составляются лексемы $\Sigma_L \subset \Sigma$.

$$\Sigma = \Sigma_S \cup \Sigma_P \cup \Sigma_L.$$

Разбор по символам и по лексемам

Тогда

	begin, *e*	**end, b**	**;, *e***	**⊥, ∅**
q_0	q_0, b	q_1, *e*		HALT
q_1			q_0, *e*	

- $Q = \{q_0, q_1\}$;
- $\Sigma = \{\text{begin, end, ;}\}$;
- $\delta = \{\ldots\}$;
- $q_0 = q_0$;
- $F = \{q_0\}$.

Разбор по символам и по лексемам

Смешанный разбор. Появляются дополнительные проблемы – как формально описать в автомате, какие части входной цепочки нужно обрабатывать посимвольно, а какие по лексемам? Решение:

- специальная пометка для элементов алфавита (столбцов таблицы переходов, состояний);
- построение еще одного автомата (ДКА), осуществляющего предварительную разбивку входной цепочки на поток лексем.

Лабораторная работа №1

Порядок выполнения лабораторной работы:

1. Описать требуемый язык заданным способом (в виде ДКА или ДМПА).
2. Написать программу, реализующую требуемый механизм синтаксического анализа.
3. Внедрить в синтаксис анализатора действия для проверки семантики языка или его интерпретации.
4. Протестировать программу.
5. Написать отчёт, включающий все требуемые пункты (в т.ч. формальное описание построенного анализатора) и удовлетворяющий требованиям ОС ТУСУР 01-2013.

Лабораторная работа №1

Вариант №1. Требования к программе:

- Должны быть явно описаны все компоненты ДКА (можно, но не обязательно, использовать ДМПА) – Q (если состояния именованные), Σ, δ, q_0, F (если маркер $\perp$ не добавлен в функцию переходов).
- Должно быть реализовано управляющее устройство автомата в виде функции или процедуры, работающее по описанному в пособии алгоритму.
- Должны быть описаны внедрённые в синтаксис автомата действия в виде отдельной функции или процедуры. Действия должны обеспечивать проверку всей семантики языка.

Лабораторная работа №1

Вариант №1. Требования к программе:

- При запуске программа должна считывать входную цепочку из файла с именем input.txt. Программа должна корректно завершать свою работу независимо от содержимого входного файла.
- Результаты работы программа должна вывести на консоль или в выходной файл output.txt. При этом, если входная цепочка содержала ошибку, в выходных данных необходимо указать её положение (номер строки и позицию в строке). Также можно указать дополнительные сведения о причине ошибки.

Лабораторная работа №1

Пример:

	+, –	.	0-9	⊥
q_0	q_1	q_2	q_3	
q_1		q_2	q_3	
q_2			q_4	
q_3		q_4	q_3	HALT
q_4			q_4	HALT

Знак можно использовать только в начале числа

Ожидается символ «0-9»

Лабораторная работа №1

Вариант №2. Требования к программе:

- Должны быть явно описаны все компоненты ДМПА – Q (если состояния именованные), Σ, Γ, δ, q_0, γ_0, F (если маркер $\perp$ не добавлен в функцию переходов).
- Должно быть реализовано управляющее устройство автомата в виде функции или процедуры, работающее по описанному в пособии алгоритму.
- Должны быть описаны внедрённые в синтаксис автомата действия в виде отдельной функции или процедуры. Действия должны обеспечивать построение дерева заданного выражения (или кода, если этап построения дерева пропускается).

Лабораторная работа №1

Вариант №2. Требования к программе:

- При запуске программа должна считывать входную цепочку из файла с именем input.txt. Программа должна корректно завершать свою работу независимо от содержимого входного файла.
- Результаты работы программа должна вывести в выходной файл output.txt. При этом, если входная цепочка содержала ошибку, сообщение об этом выводится в выходной файл, а таблица имён и код не формируются.

Лабораторная работа №1

Вариант №2. Способы формирования кода:

- выражение → ДМПА → построение дерева внедрёнными действиями → дерево → …;
- выражение → ДМПА → построение ОПЗ внедрёнными действиями → ОПЗ → …;
- выражение → ДМПА → построение ОПЗ, где операндами являются узлы дерева → дерево → …;
- выражение → ДМПА → использование стека ОПЗ для хранения кусков кода → код → …;
- и т.д.

Способы генерации псевдокода

Вариант №1. Строим дерево по строке символов:

1. На входе алгоритма имеем некоторый узел дерева node и строку с частью математического выражения expr.

2. Ищем в строке знаки операций, не заключенные в скобки, в порядке приоритета. Наименьший приоритет у присваивания («=»), средний у сложения («+»), наивысший – у умножения («*»).

3. Если знак операции найден в позиции pos строки expr, то записываем его в поле oper узла дерева node. Далее два раза вызываем рекурсивно данный алгоритм – сначала для левого поддерева node.left и для подстроки строки expr, расположенной слева от позиции pos, и затем для правого поддерева node.right и для подстроки строки expr, расположенной справа от позиции pos.

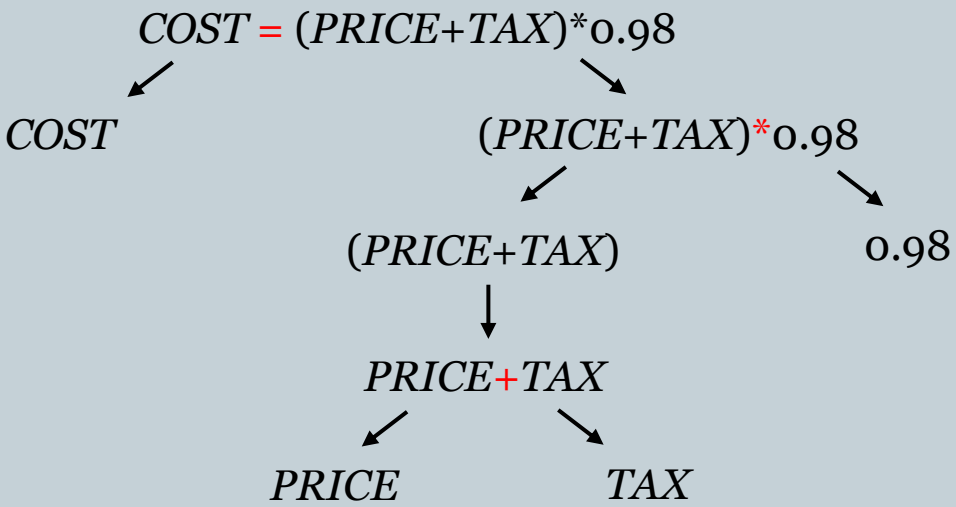
4. Если знак не найден, то:

4.1. Если при этом первым символом строки expr является открывающая скобка «(», а последним – закрывающая «)», то удалить их и вернуться на шаг 2.

4.2. В противном случае имеем лист дерева. Записываем в поле oper узла дерева node все выражение expr – оно содержит либо идентификатор, либо константу. Указатели node.left и node.right обнулить.

Способы генерации псевдокода

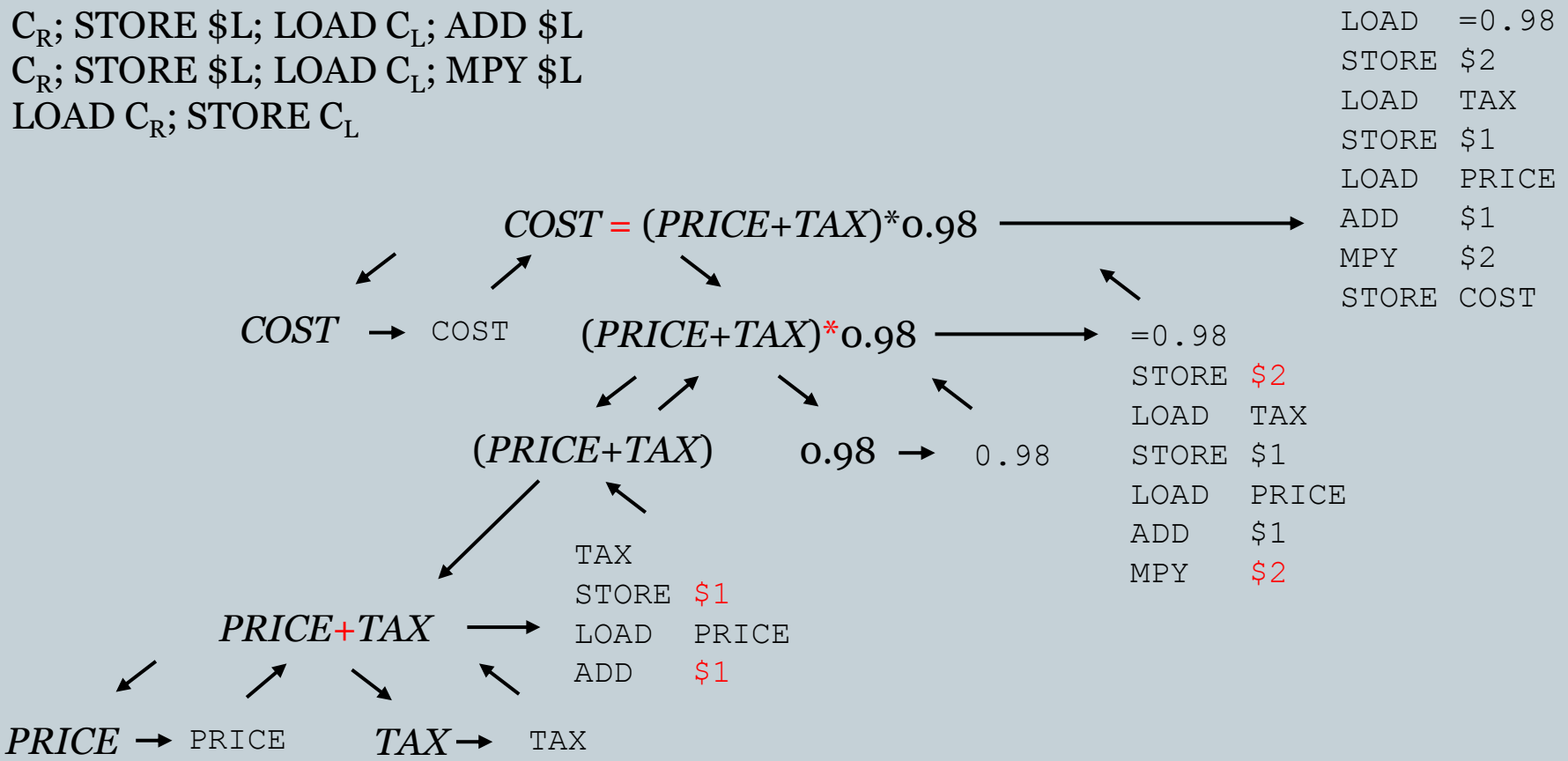
Пример:



Способы генерации псевдокода

Либо вместо дерева в рекурсии сразу строим код:

C_R; STORE \$L; LOAD C_L; ADD \$L
C_R; STORE \$L; LOAD C_L; MPY \$L
LOAD C_R; STORE C_L



Способы генерации псевдокода

Вариант №2. Строим дерево по лексемам:

1. На входе алгоритма имеем некоторый узел дерева node и список лексем с частью математического выражения expr.

2. Ищем лексемы с операциями, не заключенные в скобки, в порядке приоритета. Наименьший приоритет у присваивания («=»), средний у сложения («+»), наивысший – у умножения («*»).

3. Если знак операции найден в лексеме с номером pos списка expr, то записываем его в поле oper узла дерева node. Далее два раза вызываем рекурсивно данный алгоритм – сначала для левого поддерева node.left и для части списка expr, расположенной слева от лексемы с номером pos, и затем для правого поддерева node.right и для части списка expr, расположенной справа от лексемы с номером pos.

4. Если знак не найден, то:

4.1. Если при этом первой лексемой списка expr является открывающая скобка «(», а последней – закрывающая «)», то удалить их и вернуться на шаг 2.

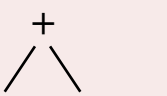
4.2. В противном случае имеем лист дерева. Записываем в поле oper узла дерева node единственную лексему из expr – она содержит либо идентификатор, либо константу. Указатели node.left и node.right обнулить.

Способы генерации псевдокода

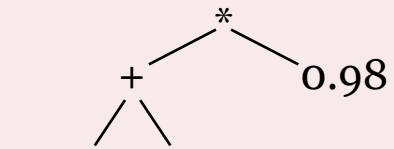
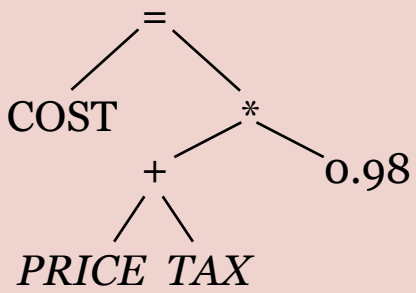
Вариант №3. Строим дерево или код по ОПЗ:

Используем алгоритм вычисления ОПЗ, но на стек вместо значений записываем элементы дерева (листья и поддеревья).

$$COST = (PRICE+TAX)*0.98 \Rightarrow COST\ PRICE\ TAX + 0.98\ * =$$

Строка	Стек
PRICE TAX + 0.98 * =	*COST*
TAX + 0.98 * =	*PRICE* *COST*
+ 0.98 * =	*TAX* *PRICE* *COST*
0.98 * =	 *PRICE TAX* *COST*

Способы генерации псевдокода

Строка	Стек
* =	0.98  *PRICE TAX* *COST*
=	 *PRICE TAX* COST
	 *PRICE TAX*

Способы генерации псевдокода

Вариант №3. Строим дерево или код по ОПЗ:

Используем алгоритм вычисления ОПЗ, но на стек вместо значений записываем фрагменты кода.

$$COST = (PRICE+TAX)*0.98 \Rightarrow COST\ PRICE\ TAX + 0.98\ *\ =$$

Строка	Стек
PRICE TAX + 0.98 * =	COST
TAX + 0.98 * =	PRICE COST
+ 0.98 * =	TAX PRICE COST
0.98 * =	TAX STORE $1 LOAD PRICE ADD $1 COST

Способы генерации псевдокода

Строка	Стек
* =	0.98 TAX STORE $1 LOAD PRICE ADD $1 COST
=	=0.98 STORE $2 LOAD TAX STORE $1 LOAD PRICE ADD $1 MPY $2 COST
	LOAD =0.98 STORE $2 LOAD TAX ...

Способы генерации псевдокода

Вариант №4. Вместо ОПЗ, ДМПА генерирует дерево или сразу код:

	0-9, *e*	+, ∅	+, +	+, ×	×, ∅	×, +	×, ×	⊥, ∅	⊥, +	⊥, ×
q_0	q_1, *e*, $\langle A_1 \rangle$									
q_1	q_1, *e*, $\langle A_2 \rangle$	q_0, +, $\langle A_3 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, +, $\langle A_4 \rangle$	q_0, ×, $\langle A_3 \rangle$	q_0, +×, $\langle A_3 \rangle$	q_0, ×, $\langle A_4 \rangle$	HALT, $\langle A_3 \rangle$	q_2, *e*, $\langle A_5 \rangle$	q_2, *e*, $\langle A_5 \rangle$
q_2								HALT	q_2, *e*, $\langle A_6 \rangle$	q_2, *e*, $\langle A_6 \rangle$

$\langle A_1 \rangle$: BUFFER := *a*;

$\langle A_2 \rangle$: BUFFER := BUFFER + *a*;

$\langle A_3 \rangle$: STACK ← BUFFER, очистить BUFFER;

$\langle A_4 \rangle$: $\langle A_3 \rangle$; пока ПРИОР(*z*) ≥ ПРИОР(*a*), выполнять $\langle A_6 \rangle$;

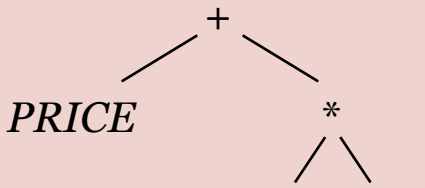
$\langle A_5 \rangle$: $\langle A_3 \rangle$; $\langle A_6 \rangle$;

$\langle A_6 \rangle$: *M* → OP, STACK → C_R, STACK → C_L, STACK ← ДЕРЕВО/КОД(OP, C_L, C_R).

Способы генерации псевдокода

Строка	M	STACK
COST = *PRICE* + *TAX**0.98		
= *PRICE* + *TAX**0.98		*COST*
PRICE + *TAX**0.98	=	*COST*
+ *TAX**0.98	=	*PRICE* *COST*
*TAX**0.98	+ =	*PRICE* *COST*
*0.98	+ =	*TAX* *PRICE* *COST*
0.98	* + =	*TAX* *PRICE* *COST*

Способы генерации псевдокода

Строка	M	STACK
	* + =	0.98 *TAX* *PRICE* *COST*
	+ =	 *TAX* 0.98 *PRICE* *COST*
	=	 *COST*
		…

Способы генерации псевдокода

Строка	M	STACK
COST = *PRICE* + *TAX**0.98		
= *PRICE* + *TAX**0.98		COST
PRICE + *TAX**0.98	=	COST
+ *TAX**0.98	=	PRICE COST
*TAX**0.98	+ =	PRICE COST
*0.98	+ =	TAX PRICE COST
0.98	* + =	TAX PRICE COST

Способы генерации псевдокода

Строка	M	STACK
	* + =	0.98 TAX PRICE COST
	+ =	0.98 STORE $1 LOAD TAX MPY $1 PRICE COST
	=	0.98 STORE $1 LOAD TAX MPY $1 STORE $2 LOAD PRICE ADD $2 COST

Способы генерации псевдокода

Строка	M	STACK
		LOAD 0.98 STORE $1 LOAD TAX MPY $1 STORE $2 LOAD PRICE ADD $2 STORE COST