Лабораторная работа № 7-8

$$T = \{i, :, *, (,)\}, V = \{S, F, L\},$$

$$P = \{S \to (F: L), S \to F, F \to L *, F \to i, L \to F\}$$

Построить МП-автомат P и расширенный МП-автомат по КС-грамматики G = (T, V, P, S), без левой рекурсии. Написать последовательность тактов автоматов для выделенной цепочки. Определить свойства автоматов. Лаб. 7. -2.1. Лаб. 8. -2.2.

1. Изучить алгоритмы построения **МП-автомат P и расширенного МП-автомата по заданной КС-грамматике (см. раздел 3.4.). Ответьте на вопрос, чем отличается МП-автомат P от расширенного МП-автомата.**

2. Выполнить построение согласно алгоритмам**. Смотрите пример и последовательность выполнения работы из раздела 3.4. Практическая работа 4.**

2.1. А). Построить МП-автомат по КС-грамматике G, используя алгоритм 3.8.

4. T ={i, :, -,(,)}, V = {S, F, L}, P = { $S \to (F: L), S \to F, F \to L *, F \to i, L \to F$ }

Используя алгоритм, получим:

МП = ({q}, {i,:,*,(,)}, { i,:,*,(,), S, F, L}, δ, q_0, S, {q}),

в котором функция переходов δ определяется следующим образом:

1. δ(q_0, ε, S) = {(q, (F:L)}
2. δ(q, ε, F) = {(q, L*), (q, i)}
3. δ(q, ε, L) = {(q, F)}
4. δ(q, a, a) = {(q, ε)} для всех a ϵ ∑ = { i,:,*,(,)}.

В). Определить последовательность тактов МП-автомата для выделенной цепочки.

Цепочка (i*:i)

Тогда последовательность тактов:

(q_0 ,(i*:i), S) ⊢
(q, (i*:i), (F:L)) ⊢

(q, (i*:i), (L*:F) ⊢
(q, (i*:i), (F*:F) ⊢
(q, (i*:i), (i*:F) ⊢
(q, i), i) ⊢
(q, ε, ε).

2.2. А). Построить расширенный МП-автомат, используя алгоритм 3.9.

Построение расширенного РМП-автомата (Q, □, Г, □, q_0, z_0, F), используем алгоритм 3.9, получим РМП = ({q, r}, {i,*,-,(,)}, { i,*,-,(,), S, F, L}, δ, q, δ, {r}), где функция переходов δ определена следующим образом:

1. δ(q, a, ε) = {q, a} для всех a∈∑={i, +, ^, (,)}.
2. δ(q, ε, (F:L)) = {(q, S)}
3. δ(q, ε, L*) = {(q, F)}
4. δ(q, ε, i) = {(q, F)}
5. δ(q, ε, F) = {(q, L)}
6. δ(q, ε, ⊥S) = {(r, ε)}

В). Определить последовательность тактов расширенного МП-автомата при анализе входной выделенной цепочки Р.

При анализе входной цепочки (i*:i) расширенный РМП-автомат выполнит следующую последовательность тактов:

(q, (i*:i), ⊥) ⊢[1] (q, i*:i), ⊥(i)
⊢ [4](q, *:i), ⊥ F)
⊢ [5](q, *:i), ⊥ L)
⊢ [5](q, *:i), ⊥(-L))
⊢ [3](q, ^i), ⊥(F))
⊢ [1](q, i), ⊥(F*)
⊢ [1](q, ε), ⊥ (F*:i)
⊢[1] (q, ε, ⊥ (F*:L))
⊢[4] (q, ε, ⊥ (F*:L))
⊢ [2](q, ε, ⊥S)
⊢[6] (q, ε, ε)

3. Определить свойства построенных МП-автоматов.

МП-автомат и расширенный РМП-автоматы – детерминированные автоматы.

Код

```
class State { // абстрактный класс с чисто виртуальной функцией
public:
	virtual void parse(char c) = 0;
};
//подкласс для объекта с состоянием правильного разбора
class OK :public State {
public:
	OK() {}
	virtual void parse(char c) { cout << "OK" << endl; }
};
//подкласс для объекта с состоянием не правильного разбора
class ERROR :public State {
public:
	ERROR() {}
	virtual void parse(char c) { cout << "ERROR string" << endl; }
};
//класс для автомата агрегация по ссылке объектов классов OK, ERROR
class Automate {
public:
	static State* state; // полиморфная переменная
	static ERROR* error;
	static OK* ok;

	Automate(string String) {
		i = 0;
		this->String = String;
	}
	char getNextChar() {// получить следующий символ из строки
		if (i<(int)String.length()) { char ch = String[i]; i++; return ch; }
		else { return ' '; }
		//при окончании строки возвращается пробел
	}
	virtual void parse() { // начать разбор
		while ((state != error) && (state != ok)) {
			state->parse(getNextChar());
		}
		state->parse(getNextChar());// принцип подстановки
	}
private:
	string String;
	int i;
}; // end class
   // присвоение начальных значений переменным автомата
State* Automate::state = NULL;
ERROR* Automate::error = new ERROR;
OK* Automate::ok = new OK;

// определение подкласса для объекта автомата, анализирующего
// контекстно-свободную грамматику
class AutomateCF : public Automate {
public:
	AutomateCF(string String) :Automate(String) { c = ' '; }
	virtual void parse() {// замещение функции parse в объекте
					// класса Automate
		c = getNextChar(); // вызов функции класса автомат
		E(); // вызов метода
		state->parse(getNextChar());
	}
	void T() { // реализация функции по диаграмме
		cout << "step1 T=" << c << endl;
		//if ((c == 'a' || c == 'b' || c == 'c')) {
```

```
		if ((c == 'i')) {
			c = getNextChar();
			cout << "step2 T=" << c << endl;
		}
		if (c == ')') { c = getNextChar(); }
		else if (c == '(') { c = getNextChar(); E(); }
	}

	void E() {// реализация функции по диаграмме
		cout << "step0 E=" << c << endl;
		T(); // вызов функции
		cout << "step1 E=" << c << endl;
		while ((c == '*' || c == ':')) {
			c = getNextChar(); T(); // вызов функции
			cout << "step2 E=" << c << endl;
		}
		if (c == ' ') { state = ok; }
		else { state = error; }
	}
private:
	char c;
};
// программа ввода строки с клавиатуры
string getString() {
	//string String = "c+d-dkf-n";
	string String = "i*:i*";//i*:*i
	// string String = "a+b";
	char c;
	int N = (int)String.length();
	cout << "Enter string " << N << " char " << endl;
	for (int i = 0; i < N; i++) {
		cin >> c;
		String[i] = c;
	}
	return String;
}

int main() {
	//создание объекта автомат
	Automate * a = new AutomateCF(getString());
	//инициализация начальных значений
	a->state = Automate::ok;
	a->parse(); //синтаксический анализ
	return 0;

}
```