Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Теория автоматов и формальных языков

ЛАБОРАТОРНАЯ РАБОТА № 1-2

Тема: Построение модели автомата Мили

Выполнил:

студент гр. 1ПИб-02-2оп-23

Кринкин Олег Алексеевич

Проверил:

доцент, к.т.н. Ганичева Оксана Георгиевна

ЗАДАНИЕ

Часть 1.

В учебнике Карпова Ю. Г. "Теория автоматов" на стр. 112-114 разобрать пример1 автомата "Электронные часы" и автомат калькулятор.

Необходимо ознакомиться с теорией из учебника Карпова Ю.Г. Теория автоматов - стр. 114-117 и взять его за основу.

Построить СВОЙ вариант автомата Электронные часы. Автомат должен выводить текущие дату, время, месяц, день недели.

По заданному описанию:

1. Определить составные компоненты автомата Мили (X, У, S, s0, функция

переходов, функция выходов);

1. построить граф переходов в терминологии автомата Мили (см. лекцию. У вас он будет отличаться от представления в учебнике);
2. построить таблицу переходов;
3. построить таблицу выходов;
4. построить эквивалентный автомат Мура;
5. рассмотреть произвольную входную последовательность (длиной не менее 15 символов) и построить протокол работы своего автомата для этой последовательности. Построить этот протокол работы и для автомата Мили, и для автомата Мура.

Отчет должен содержать результаты выполнения по всем пунктам 1 - 6.

Часть 2.

Написать программу на языке высокого уровня, моделирующую работу автомата,

построенного в 1-ой работе (электронные часы или калькулятор).

Перед написанием программы обязательно прочитать теорию в учебнике Карпова Ю. Г.

"Теория автоматов" стр. 99-100 (глава 3, стр. 95-118)

В программе должны быть предусмотрены:

- реализация с использованием оператора выбора Switch (внешний выбор по состояниям,

внутренний - по входным символам).

- ввод исходной строки входных символов (произвольной длины)

- отображение информации по входным символам, внутренним состояниям, выходным

символам (протокол работы автомата - см. лекции)

6) Содержание отчета:

- Задание;

- описание работы автомата;

- Таблица переходов/выходов;

- граф конечного автомата;

- структурная схема автомата (см. стр.100 учебника Карпова, Теория автоматов);

- текст рабочей программы с комментариями;

- результаты тестирования автомата

- вывод по работе (должен содержать характеристику построенного автомата).

ХОД РАБОТЫ

Часть 1.

Для описания были выбраны настоящие электронные часы, имеющие большое количество состояний. Некоторые их функции были признаны лишними для выполнения лабораторной работы (например включение/выключение подсветки, установка второго времени и таймер), поэтому они не были учтены.

1. Определены составные компоненты автомата Мили:

X:

Y:

S:

1. Исходя из определённых выше составных компонентов построен следующий граф переходов автомата Мили (рис. 1):

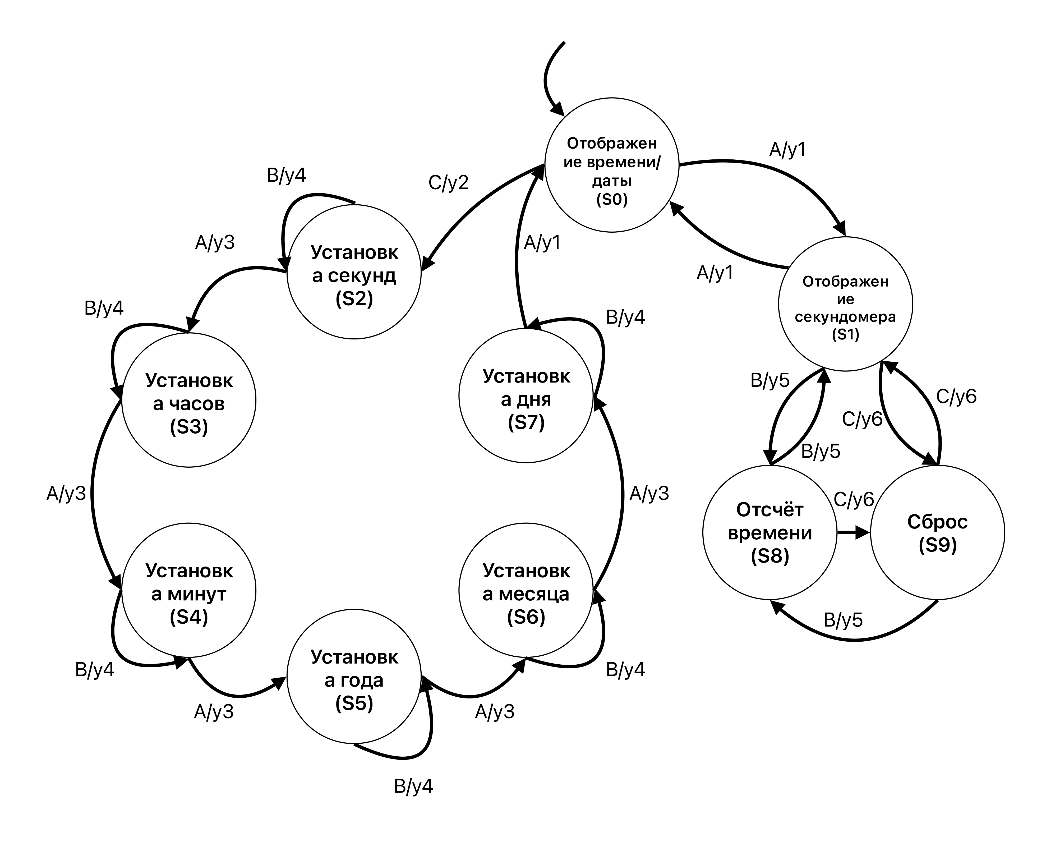


Рис. 1. Граф переходов

1. Также исходя из составных компонентов выше построена таблица переходов (табл. 1):

Таблица 1: Таблица переходов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X/S* | *S0* | *S1* | *S2* | *S3* | *S4* | *S5* | *S6* | *S7* | *S8* | *S9* |
| *X1(a)* | *S1* | *S0* | *S3* | *S4* | *S5* | *S6* | *S7* | *S0* | *-* | *-* |
| *X2(b)* | *-* | *S8* | *S2* | *S3* | *S4* | *S5* | *S6* | *S7* | *S1* | *S8* |
| *X3(c)* | *S2* | *S9* | *-* | *-* | *-* | *-* | *-* | *-* | *S9* | *S1* |

1. Построена таблица выходов (табл. 2):

Таблица 2: Таблица выходов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X/S* | *S0* | *S1* | *S2* | *S3* | *S4* | *S5* | *S6* | *S7* | *S8* | *S9* |
| *X1(a)* | *Y1* | *Y1* | *Y3* | *Y3* | *Y3* | *Y3* | *Y3* | *Y1* | *-* | *-* |
| *X2(b)* | *-* | *Y5* | *Y4* | *Y4* | *Y4* | *Y4* | *Y4* | *Y4* | *Y5* | *Y5* |
| *X3(c)* | *Y2* | *Y6* | *-* | *-* | *-* | *-* | *-* | *-* | *Y6* | *Y6* |

1. Построена совмещённая таблица (табл. 3):

Таблица 3: Совмещённая таблица

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *X/S* | *S0* | *S1* | *S2* | *S3* | *S4* | *S5* | *S6* | *S7* | *S8* | *S9* |
| *X1(a)* | *S1/Y1* | *S0/Y1* | *S3/Y3* | *S4/Y3* | *S5/Y3* | *S6/Y3* | *S7/Y3* | *S0/Y1* | *-* | *-* |
| *X2(b)* | *-* | *S8/Y5* | *S2/Y4* | *S3/Y4* | *S4/Y4* | *S5/Y4* | *S6/Y4* | *S7/Y4* | *S1/Y5* | *S8/Y5* |
| *X3(c)* | *S2/Y2* | *S9/Y6* | *-* | *-* | *-* | *-* | *-* | *-* | *S9/Y6* | *S1/Y6* |

1. Исходя из построенных ранее данных был построен следующий граф для аналогичного автомата Мура (рис. 2):

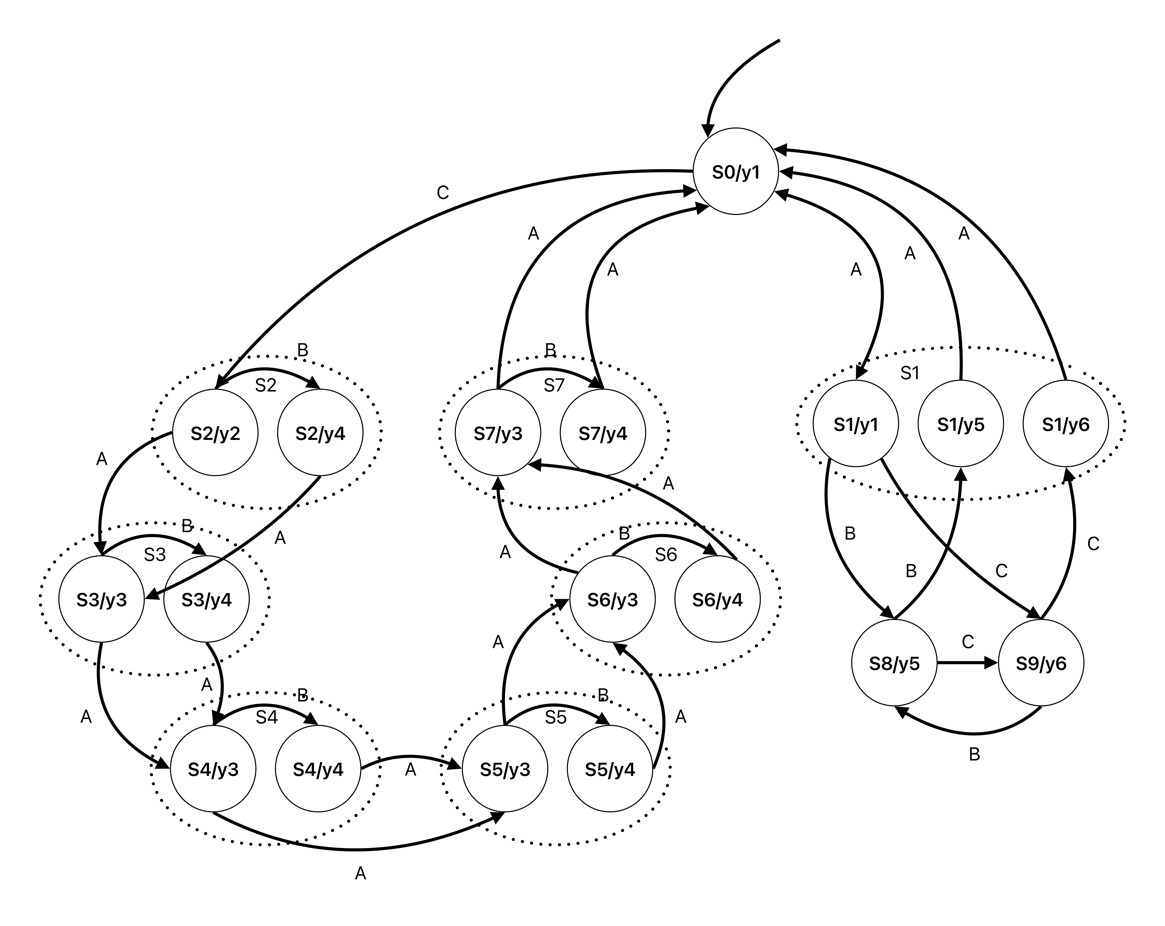


Рис. 2. Граф переходов автомата Мура

1. Установлена возможная входная последовательность для автомата вида (табл. 3):

Таблица 3: Протокол работы

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *t* | *t1* | *t2* | *t3* | *t4* | *t5* | *t6* | *t7* | *t8* | *t9* | *t10* |
| *X* | *C* | *A* | *A* | *B* | *B* | *A* | *A* | *A* | *A* | *A* |
| *S* | *S0* | *S2* | *S3* | *S4* | *S4* | *S4* | *S5* | *S6* | *S7* | *S0* |
| *Y* | *Y2* | *Y3* | *Y3* | *Y4* | *Y4* | *Y3* | *Y3* | *Y3* | *Y1* | *Y1* |

Часть 2.

1. По описанному автомату была спроектирована программа, имитирующая его работу, строя при этом таблицы переходов, выходов и протокола. Программа принимает на вход последовательность входных символов либо отдельно (кнопками), либо в виде строки и при помощи операций ветвления обрабатывает входные символы, добавляя для каждого соответствующий элемент в таблицы.
2. Для написанной программы составлена её структурная схема (рис. 3):

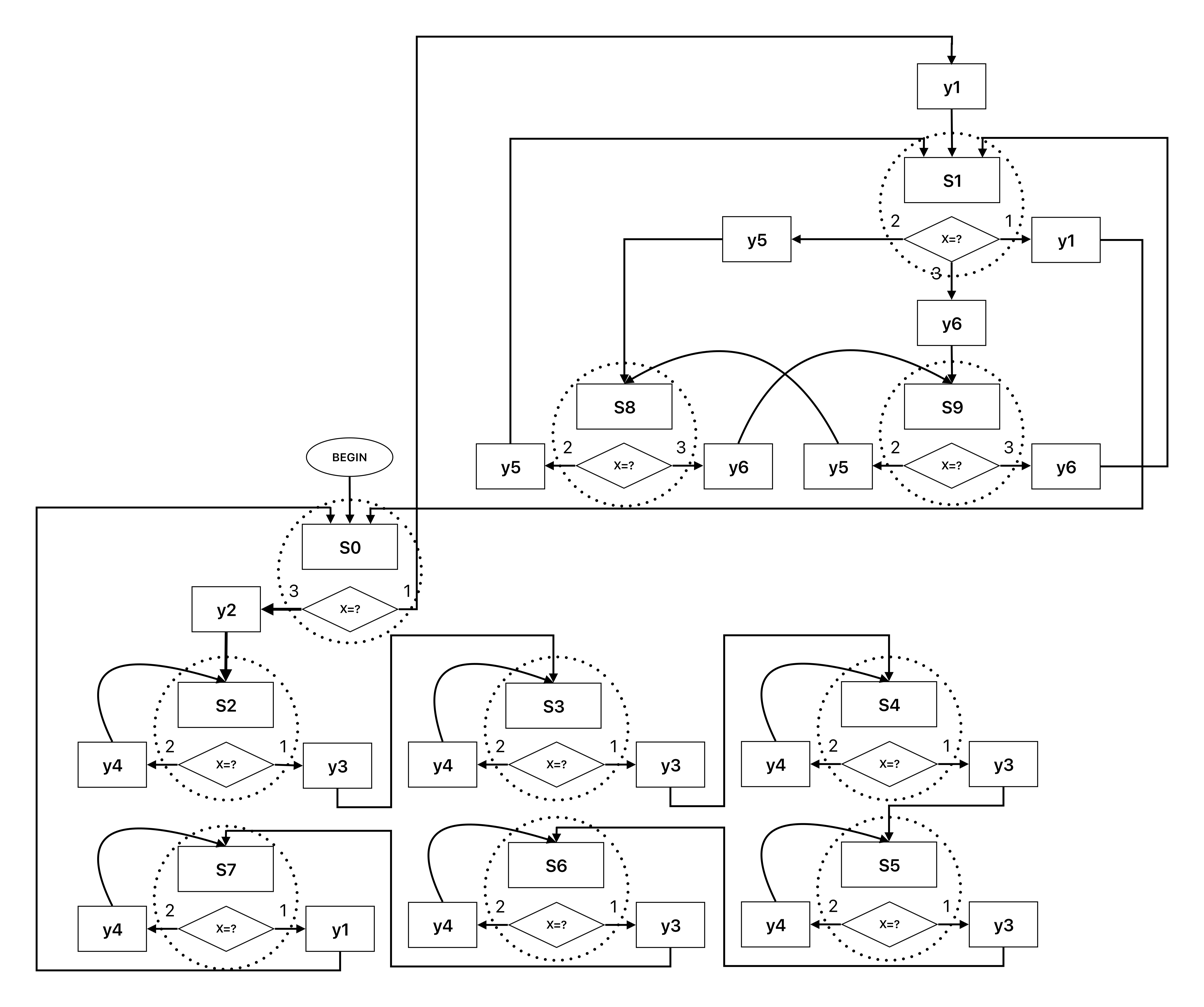


Рис. 3. Структурная схема программы

1. При использовании установленной в табл. 3 последовательности программа добавляет в таблицы переходов и выходов соответствующие данные и ниже строит идентичную табл. 3 таблицу протокола работы, что означает что программа работает правильно (рис. 4).

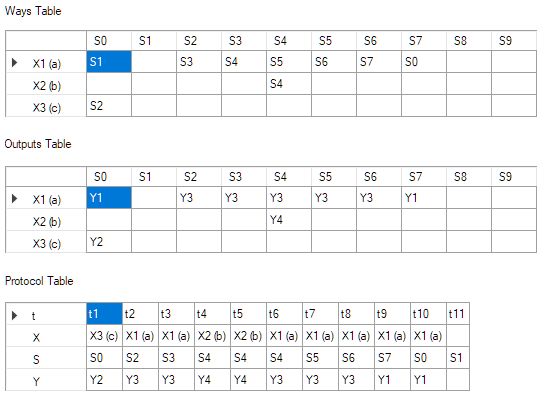


Рис. 4. Таблицы по заданному протоколу

1. При задании последовательности *CCBACBACBACBACBACBABABACABBCCA*, которая бы обошла все возможные состояния и реакции, программа строит таблицы со следующим результатом (рис. 5):

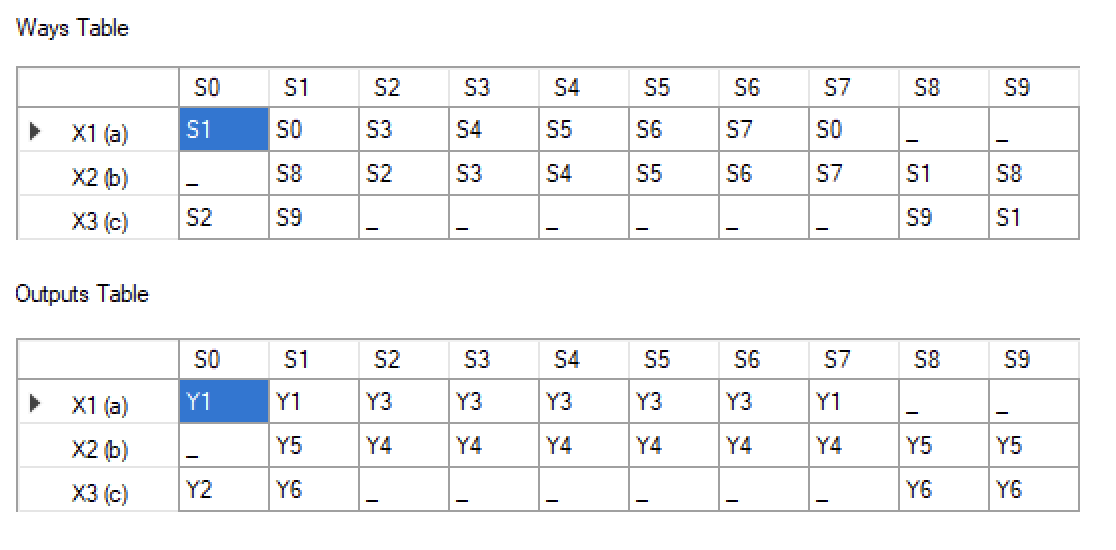


Рис. 5. Таблицы с полным протоколом работы

Полученные таблицы полностью совпадают с таблицами переходов и выходов, определённых для автомата ранее, что означает что программа работает правильно.

1. При установке последовательности с неверным входным символом программа выполнит команды, идущие до него, а затем отобразит сообщение об ошибке (рис. 6), после чего программа перейдёт в изначальное состояние при этом очистив таблицу протокола.

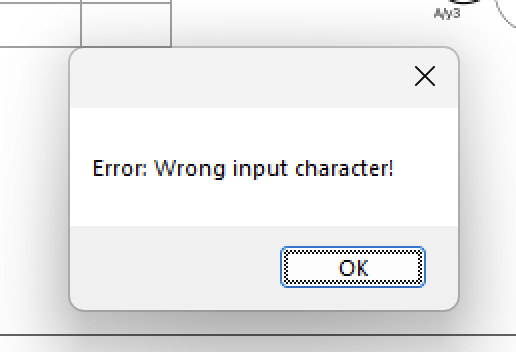
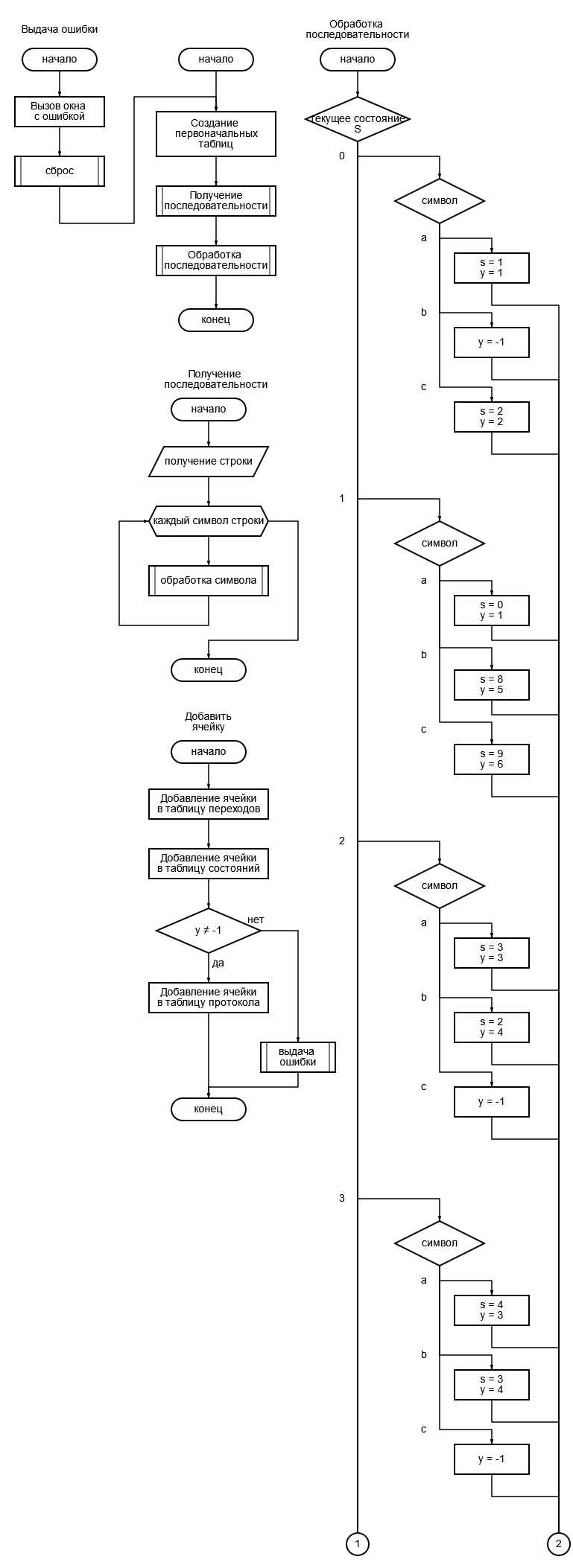
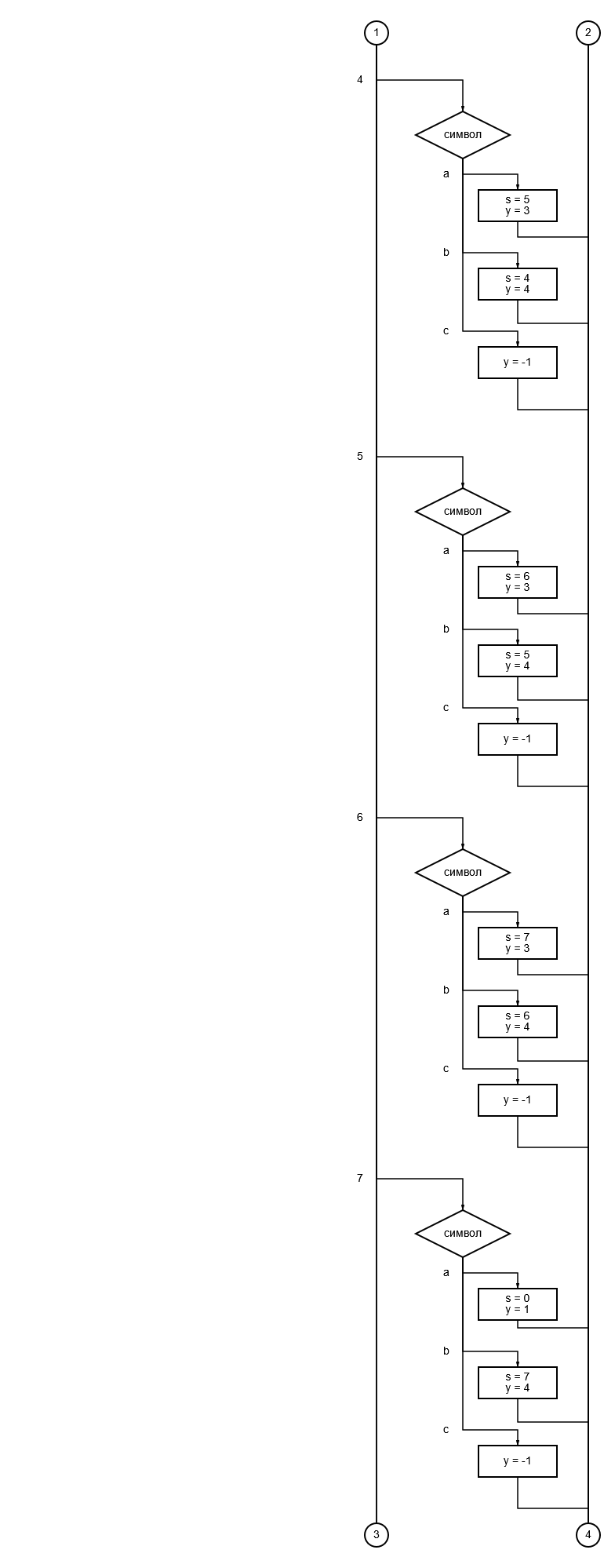


Рис. 6. Сообщение об ошибке

1. Для алгоритма написанной программы составлена следующая блок-схема (рис. 7):





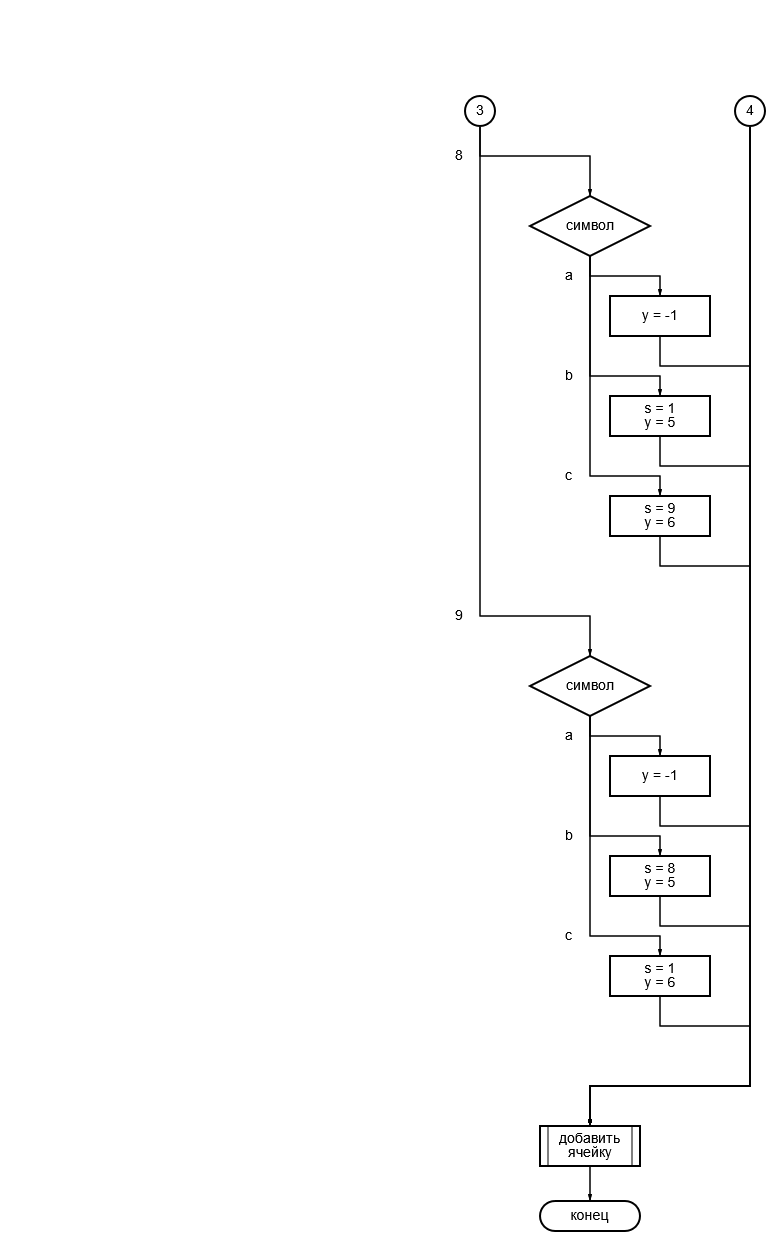


Рис. 7. Блок-схема программы

РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

В результате выполнения лабораторной работы получено представление частичного автомата «Электронные часы» в виде автомата Мили и аналогичного ему автомата Мура. Также была спроектирована и разработана программа (рис. 8), моделирующая работу автомата.

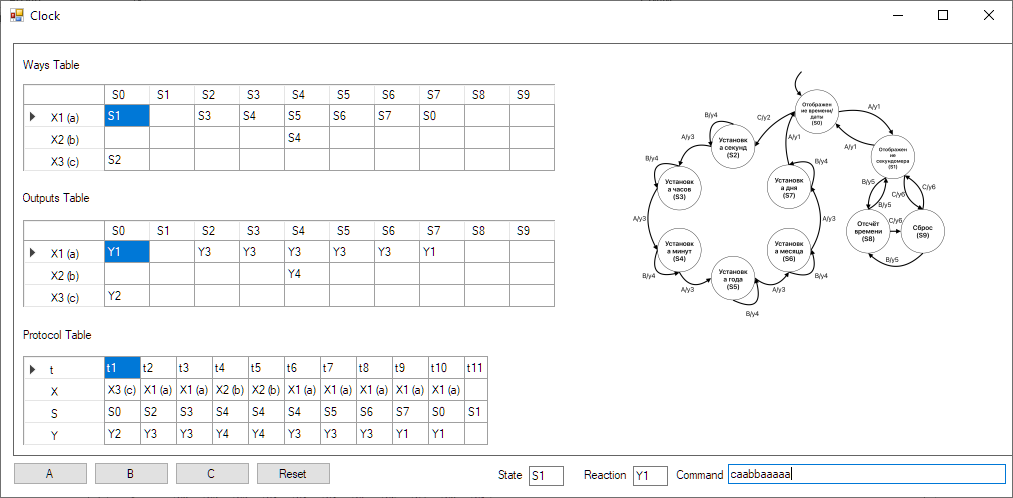
**

Рис. 8. Окно программы

Часть программного кода спроектированной программы:

private:

int s = 0;

int y = 0;

int t = 0;

System::String^ get\_y() {

if (y != -1) {

return "Y" + y;

}

return "\_";

}

System::String^ get\_s() {

if (y != -1) {

return "S" + s;

}

return "\_";

}

System::Void add\_cell(char command, int state, int proto\_len) {

/\*

\* Функция, добавляющая новую ячейку в каждую таблицу по команде

\*/

switch (command) {

case 'a': {

this->dataGridViewWays->Rows[0]->Cells[state]->Value = get\_s();

this->dataGridViewOut->Rows[0]->Cells[state]->Value = get\_y();

this->dataGridViewProtocol->Rows[1]->Cells[proto\_len-1]->Value = "X1 (a)";

break;

}

case 'b': {

this->dataGridViewWays->Rows[1]->Cells[state]->Value = get\_s();

this->dataGridViewOut->Rows[1]->Cells[state]->Value = get\_y();

this->dataGridViewProtocol->Rows[1]->Cells[proto\_len-1]->Value = "X2 (b)";

break;

}

case 'c': {

this->dataGridViewWays->Rows[2]->Cells[state]->Value = get\_s();

this->dataGridViewOut->Rows[2]->Cells[state]->Value = get\_y();

this->dataGridViewProtocol->Rows[1]->Cells[proto\_len-1]->Value = "X3 (c)";

break;

}

}

if (y != -1) {

// Выполняется если если символ допустимый

this->dataGridViewProtocol->Rows[0]->Cells[proto\_len - 1]->Value = "t" + t;

this->dataGridViewProtocol->Rows[2]->Cells[proto\_len - 1]->Value = "S"+state;

this->dataGridViewProtocol->Rows[3]->Cells[proto\_len - 1]->Value = get\_y();

this->dataGridViewProtocol->Rows[0]->Cells[proto\_len]->Value = "t" + (t+1);

this->dataGridViewProtocol->Rows[2]->Cells[proto\_len]->Value = get\_s();

} else {

throw\_error();

}

}

System::Void make\_table(DataGridView^ table) {

/\*

\* Функция инициализирующая пустую таблицу

\*/

table->ColumnCount = 10;

table->Rows->Add(3);

table->Rows[0]->HeaderCell->Value = "X1 (a)";

table->Rows[1]->HeaderCell->Value = "X2 (b)";

table->Rows[2]->HeaderCell->Value = "X3 (c)";

for (int i = 0; i < table->ColumnCount; i++)

table->Columns[i]->Name = "S" + i;

}

System::Void clear\_table(DataGridView^ table) {

/\*

\* Функция, очищающая ячейки таблицы

\*/

for (int i = 0; i < table->RowCount; i++)

for (int j = 0; j < table->ColumnCount; j++)

table->Rows[i]->Cells[j]->Value = "";

}

System::Void reset() {

/\*

\* Функция, возвращающая программу в первоначальное состояние

\*/

s = 0;

y = 0;

t = 0;

stateBox->Text = "S0";

yBox->Text = "\_";

commandBox->Text = "";

clear\_table(this->dataGridViewProtocol);

this->dataGridViewProtocol->ColumnCount = 1;

}

System::Void throw\_error() {

/\*

\* Функция, открывающая окно с ошибкой и вызывающая reset()

\*/

MessageBox::Show("Error: Wrong input character!");

reset();

}

System::Void process\_command(char command) {

/\*

\* Функция, обрабатывающая следующее состояние автомата по команде

\*/

int prev\_s = s;

switch (s) {

case 0: {

switch (command) {

case 'a': { s = 1; y = 1; break; };

case 'b': { y = -1; break; };

case 'c': { s = 2; y = 2; break; };

}

break;

}

case 1: {

switch (command) {

case 'a': { s = 0; y = 1; break; };

case 'b': { s = 8; y = 5; break; };

case 'c': { s = 9; y = 6; break; };

}

break;

}

case 2: {

switch (command) {

case 'a': { s = 3; y = 3; break; };

case 'b': { s = 2; y = 4; break; };

case 'c': { y = -1; break; };

}

break;

}

case 3: {

switch (command) {

case 'a': { s = 4; y = 3; break; };

case 'b': { s = 3; y = 4; break; };

case 'c': { y = -1; break; };

}

break;

}

case 4: {

switch (command) {

case 'a': { s = 5; y = 3; break; };

case 'b': { s = 4; y = 4; break; };

case 'c': { y = -1; break; };

}

break;

}

case 5: {

switch (command) {

case 'a': { s = 6; y = 3; break; };

case 'b': { s = 5; y = 4; break; };

case 'c': { y = -1; break; };

}

break;

}

case 6: {

switch (command) {

case 'a': { s = 7; y = 3; break; };

case 'b': { s = 6; y = 4; break; };

case 'c': { y = -1; break; };

}

break;

}

case 7: {

switch (command) {

case 'a': { s = 0; y = 1; break; };

case 'b': { s = 7; y = 4; break; };

case 'c': { y = -1; break; };

}

break;

}

case 8: {

switch (command) {

case 'a': { y = -1; break; };

case 'b': { s = 1; y = 5; break; };

case 'c': { s = 9; y = 6; break; };

}

break;

}

case 9: {

switch (command) {

case 'a': { y = -1; break; };

case 'b': { s = 8; y = 5; break; };

case 'c': { s = 1; y = 6; break; };

}

break;

}

}

t++;

int proto\_len = this->dataGridViewProtocol->ColumnCount++;

add\_cell(command, prev\_s, proto\_len);

stateBox->Text = "S" + s;

yBox->Text = get\_y();

}

System::Void Clock\_Load(System::Object^ sender, System::EventArgs^ e) {

/\*

\* Функция, вызывающаяся при запуске программы

\*/

this->pictureBox2->Image = Image::FromFile("scheme.png");

make\_table(this->dataGridViewWays);

make\_table(this->dataGridViewOut);

this->dataGridViewProtocol->ColumnCount = 1;

this->dataGridViewProtocol->Rows->Add(4);

this->dataGridViewProtocol->Rows[0]->HeaderCell->Value = "t";

this->dataGridViewProtocol->Rows[1]->HeaderCell->Value = "X";

this->dataGridViewProtocol->Rows[2]->HeaderCell->Value = "S";

this->dataGridViewProtocol->Rows[3]->HeaderCell->Value = "Y";

}

bool error = false;

System::Void commandBox\_Enter(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e) {

/\*

\* Функция, вызывающаяся при нажатии клавиши Enter/Return на поле ввода.

\*/

if (error) {

error = false;

this->commandBox->Text = "";

}

if (e->KeyChar == (char)Keys::Return) {

for (int i = 0; i < this->commandBox->Text->Length; i++) {

if (System::Char::ToLower(this->commandBox->Text[i]) == 'a' ||

System::Char::ToLower(this->commandBox->Text[i]) == 'b' ||

System::Char::ToLower(this->commandBox->Text[i]) == 'c') {

// Вызов функции обработки каждого символа последовательности

process\_command(System::Char::ToLower(this->commandBox->Text[i]));

}

else

{

this->commandBox->Text = "ERROR";

error = true;

break;

}

}

}

}

System::Void buttonReset\_Click(System::Object^ sender, System::EventArgs^ e) {

// Нажатие кнопки Reset

reset();

}

System::Void buttonA\_Click(System::Object^ sender, System::EventArgs^ e) {

// Нажатие кнопки задания символа A

process\_command('a');

}

System::Void buttonB\_Click(System::Object^ sender, System::EventArgs^ e) {

// Нажатие кнопки задания символа B

process\_command('b');

}

System::Void buttonC\_Click(System::Object^ sender, System::EventArgs^ e) {

// Нажатие кнопки задания символа C

process\_command('c');

}

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было выполнено построение частичного автомата «Электронные часы», при этом был рассмотрен как вариант автомата Мили, так и аналогичный ему вариант автомата Мура. Алфавит построенного автомата состоит из трёх команд: нажатие на кнопку A, нажатие на кнопку B, нажатие на кнопку C. Для автомата было определено 10 состояний, включая начальное (S0-S9) и 6 возможных выходных реакций (y1-y6).

Для автомата Мили были построены таблицы переходов, выходов, а также определена одна из возможных входных последовательностей, которая бы не приводила к пустой реакции, после чего для этой последовательности была построена таблица с протоколом работы автомата.

После для описанного автомата была спроектирована и написана программа, имитирующая работу автомата: нахождение в нескольких состояниях и переходы из каждого из них. Также программа строит три таблицы: таблицу переходов, таблицу выходов и таблицу с протоколом работы для заданной в программу последовательности входных сигналов.

Характеристики автомата Мили:

* Абстрактный – т.к. имеет некоторые допущения и не описывает реальную работу автомата, а представляет собой лишь математическую модель;
* Частичный – т.к. имеются неопределённые выходные сигналы;
* Асинхронный – т.к. время перехода зависит от событий *x1*, *x2*, *x3*, между которыми различный промежуток времени.
* Детерминированный – т.к. автомат из состояния *Si* может перейти лишь в одно состояние *Sj*.