УДК 681.3.06

РЕАЛИЗАЦИЯ АНИМАЦИИ ПРИ ПОСТРОЕНИИ ВИЗУАЛИЗАТОРОВ АЛГОРИТМОВ НА ОСНОВЕ АВТОМАТНОГО ПОДХОДА

М. А. Казаков,
аспирант
А. А. Шалыто,
доктор техн. наук, профессор
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В статье предлагается подход к реализации анимации алгоритмов, который расширяет технологию построения визуализаторов алгоритмов на основе автоматного подхода.

Article describes proposed approach for algorithm animation extension above existing technology for algorithm visualizations implementation based on the automaton programming approach.

Введение

Визуализаторы алгоритмов широко используются в процессе преподавания дискретной математики и теории классических вычислительных алгоритмов [1 – 3].

Визуализатор – программа, иллюстрирующая выполнение алгоритма при определенных исходных данных [4].

В работе [5] рассматривался метод построения визуализаторов на основе автомата Мили, а также предлагалась технология формального преобразования вычислительного алгоритма в визуализатор. Аналогичным образом логика визуализаторов может строиться на основе автомата Мура.

Ранее визуализатор рассматривался как дискретная последовательность статических иллюстраций, что в большинстве случаев достаточно для обучения. Однако в некоторых алгоритмах статических иллюстраций недостаточно. Примером такого алгоритма может служить обход дерева [6], так как именно процесс движения указателя по вершинам дерева является наиболее интересным и важным для понимания этого алгоритма.

В настоящей работе предлагается расширение технологии построения визуализаторов с целью включения в нее возможности анимации требуемых шагов визуализации. При этом рассматривается такая разновидность анимации, при которой изображается переход от предыдущих значений визуализируемых переменных к последующим. Поскольку каждая статическая иллюстрация является функцией от визуализируемых переменных, то указанная анимация и будет визуализировать шаг алгоритма в динамике. Предлагаемый подход иллюстрируется на примере традиционной реализации алгоритма обхода двоичного дерева [7].

Автоматная технология построения визуализаторов без анимации

В работе [5] описана автоматная технология построения визуализаторов алгоритмов, состоящая из следующих этапов.

1. *Постановка задачи, которую решает визуализируемый алгоритм.*
2. *Решение задачи в словесно-математической форме.*
3. *Выбор визуализируемых переменных.*
4. *Анализ алгоритма для визуализации.* Анализируется решение с целью определения того, что и как отображать на экране.
5. *Алгоритм решения задачи.*
6. *Реализация алгоритма на выбранном языке программирования.* На этом шаге производится реализация алгоритма, его отладка и проверка работоспособности.
7. *Построение схемы алгоритма по программе.*

8. *Преобразование схемы алгоритма в граф переходов автомата, реализующего алгоритм*, который может быть как автоматом Мили, так и автоматом Мура [8, 9].

9. *Преобразование автомата реализующего алгоритм в автомат визуализации.*

10. *Выбор интерфейса визуализатора.*

11. *Сопоставление иллюстраций и комментариев с состояниями автомата* (иллюстрации формируются компонентом программы, называемым «рисовальщик»).

12. *Архитектура программы визуализатора.*

13. *Программная реализация визуализатора.*

С целью обеспечения возможности анимации в последовательность действий вводятся дополнительные шаги. Отметим следующее: поскольку изменения визуализируемых переменных в используемых в настоящей работе автоматах Мура производятся в состояниях, что весьма удобно, то будем рассматривать вопрос о введении анимации в визуализаторы, построенные на основе автоматов Мура.

Анимация в визуализаторах на основе автоматов Мура

Для построения визуализатора с анимацией предлагается применять четыре типа автоматов:

1) *автомат, реализующий алгоритм* (формируется на восьмом этапе);

2) *автомат визуализации* (формируется на девятом этапе);

3) *преобразованный автомат визуализации*;

4) *автомат анимации.*

Автомат визуализации отображает последовательно *статические иллюстрации* в каждом состоянии, что приводит к *статической (пошаговой) визуализации.* Здесь под статической иллюстрацией понимается фиксированный кадр, не изменяющийся с течением времени.

Преобразованный автомат визуализации кроме статических иллюстраций отображает также и *динамические иллюстрации* в дополнительно вводимых анимационных состояниях. Это позволяет говорить о *динамической визуализации* (*анимации*).

Динамическая иллюстрация состоит из набора промежуточных статических иллюстраций, отображаемых на экране автоматом анимации через определенные промежутки времени. Это приводит к динамическому изменению иллюстрации в анимационном состоянии.

Для автоматов Мура анимацию (также как и формирование статических изображений и комментариев) естественно проводить в состояниях. Для введения анимации в визуализатор расширим технологию за счет введения следующих шагов:

a) *выбор состояний автомата визуализации, в которых выполняется анимация* (такие состояния будем называть анимационными);

b) *построение преобразованного автомата визуализации путем замены каждого из анимационных состояний тремя состояниями*, в первом из которых производятся преобразования данных, во втором – анимация, соответствующая этому состоянию, при переходе в третье состояние анимация заканчивается, а непосредственно в третьем состоянии отображается статическая иллюстрация, соответствующая состоянию исходного автомата;

c) *разработка анимационного автомата* (выполняется один раз для всех визуализаторов, проектируемых по этой технологии);

d) *обеспечение взаимодействия между преобразованным автоматом визуализации и анимационным автоматом;*

e) *разработка и реализация функции вывода каждой динамической иллюстрации, зависящей от состояния автомата визуализации и шага анимации.*

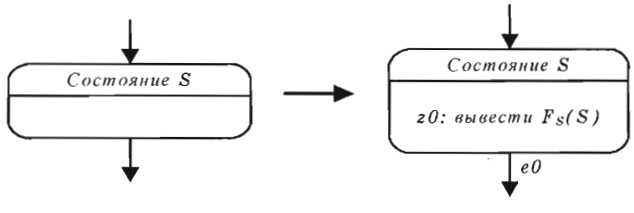
До перехода к более подробному рассмотрению новых этапов изложим, в чем состоит девятый этап исходной технологии (рис. 1).

На этом этапе вводится выходное воздействие $z0$, осуществляющее формирование статического изображения и комментария в рассматриваемом состоянии. Различия в изображениях и комментариях формализуются с помощью функции $F_S(S)$, параметром которой является номер состояния S. В каждом состоянии, формирующем выходное воздействие $z0$, переход к следующему состоянию осуществляется по событию $e0$ (нажатие клавиши «шаг» в интерфейсе визуализатора).

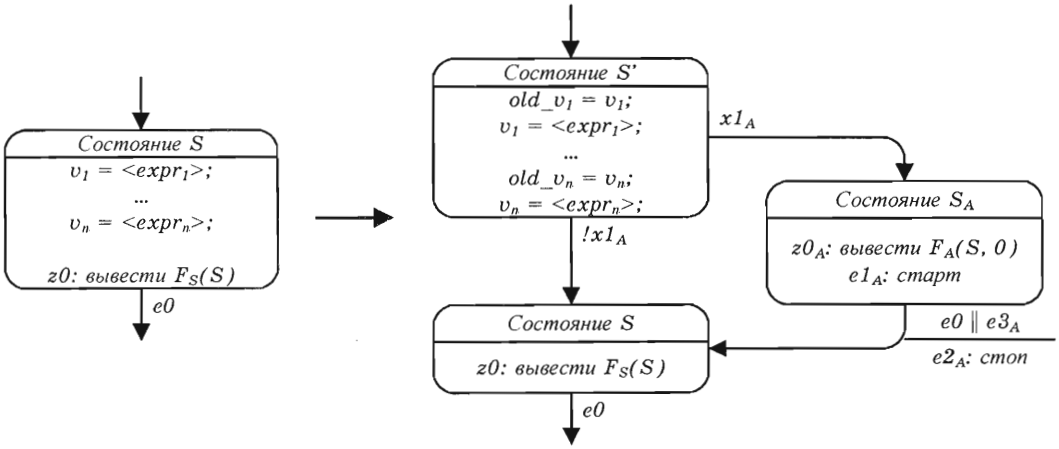
Поясним каждый из вновь вводимых этапов $a - e$.

На этапе a выполняется выбор анимационных состояний в соответствии с особенностями алгоритма и принятыми решениями по анимации на четвертом этапе исходной технологии (анализ алгоритма для визуализации). Так как в алгоритме обхода двоичного дерева наибольший интерес представляет процесс перехода между вершинами, то состояния, в которых происходит переход, и будут выбраны в качестве анимационных.

На этапе b в автомате визуализации для получения преобразованного автомата этого типа



■ ***Рис. 1.*** *Формирование состояний автомата визуализатора*



■ *Рис. 2. Преобразование анимационного состояния*

каждое анимационное состояние формально заменяется тремя состояниями (рис. 2).

Опишем это преобразование. Предположим, что состояние S выбрано для визуализации. Предположим также, что в этом состоянии изменяются значения визуализируемых переменных $v_1, ..., v_n$. При этом переменной v_i присваивается значение выражения $< expr_i >$. Преобразование состоит в замене выбранного состояния на три состояния – S', S и S_A.

В состоянии S' значения переменных $v_1, ..., v_n$ предварительно сохраняются в переменных $old\_v_1, ..., old\_v_n$, а также проводятся все действия, выполняемые в состоянии S автомата визуализации.

В состоянии S' преобразованного автомата визуализации в отличие от состояния S исходного автомата этого типа не происходит ожидания события, а выполняется переход в одно из состояний – S_A или S.

Переход в состояние S_A производится, если переменная $x1_A$ принимает значение `true` (анимация включена). Анимация осуществляется, пока преобразованный автомат визуализации находится в состоянии S_A. При входе в это состояние осуществляется запуск автомата анимации при помощи события $e1_A$: *старт* и отображается иллюстрация $F_A(S, 0)$, соответствующая началу анимации.

Выход из состояния анимации и переход в состояние S происходит, как по событию $e0$ (нажатие клавиши), так и по событию $e3_A$ (окончание анимации), и сопровождается сигналом $e2_A$: *стоп*. Таким образом, обеспечивается как автоматическое, так и ручное завершение анимации.

Состояние S является завершающим. В этом состоянии осуществляется отображение статической иллюстрации $F_S(S)$. Отметим, что поскольку в преобразованном анимационном автомате присутствуют действия на дугах, то этот автомат является смешанным автоматом.

Также следует отметить, что при выключении анимации ($x1_A$ отсутствует) преобразованный автомат визуализации становится изоморфным исходному автомату визуализации. Поэтому включение анимации может рассматриваться именно как *расширение* исходного визуализатора, а не его преобразование.

На этапе c строится автомат анимации (рис. 3). Он является смешанным автоматом, поскольку содержит действия как на дугах, так и в вершинах графа перехода.

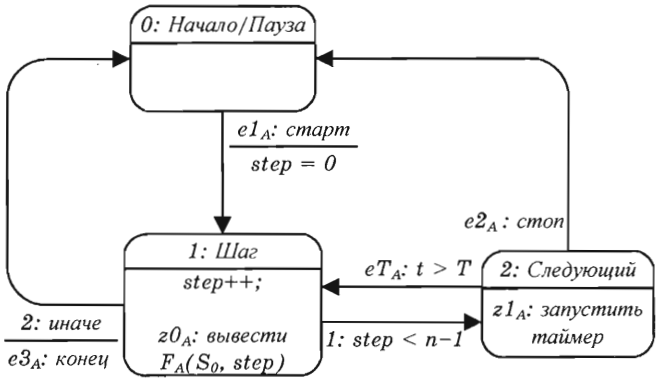
Из рассмотрения рис. 3 следует, что автомат предназначен для изменения значения переменной *step*, которая хранит номер шага анимации.

Автомат анимации формирует следующие выходные воздействия и события:

- $z1_A$: *запустить таймер* – запускает таймер, который через период времени T генерирует событие eT_A;
- $z0_A$: *вывести* $F_A(S_0, step)$ – информирует «рисовальщик» о необходимости перерисовать иллюстрацию;
- $e3_A$: *конец анимации* – сигнализирует о том, что анимация закончилась.

Анимационный автомат реагирует на следующие события:

- $e1_A$: *старт* – автомат визуализации перешел в состояние, в котором должна начаться анимация;



■ *Рис. 3. Автомат анимации*

• $e2_A$: *стоп* – автомат визуализации перешел в состояние, в котором анимация должна закончиться;

• eT_A – событие от таймера, по которому осуществляется переход к следующему шагу анимации.

Описанный автомат является универсальным и может быть использован в любом визуализаторе.

На этапе *d* автомат визуализации и автомат анимации связываются посредством событий и выходных воздействий, как показано на рис. 4.

На этапе *e* стандартная функциональность «рисовальщика» сопоставления номера состояния с текстовым комментарием и статической иллюстрацией расширяется. В режиме анимации «рисовальщик» принимает на вход значение номера шага. Другими словами, если функция F_S реализует статические иллюстрации, а функция F_A – динамические (анимационные) иллюстрации, то при переходе от статики к анимации осуществляется преобразование вида

$$F(state) \to \begin{cases} F_S(state); \\ F_A(state, step). \end{cases}$$

При этом справедливы следующие соотношения:

$$F_S(state) = F(state); \quad (1)$$

$$F_A(state, 0) = F_S(state - 1); \quad (2)$$

$$F_A(state, N) = F_S(state). \quad (3)$$

Соотношение (1) отражает тот факт, что преобразованный автомат визуализации формирует статические иллюстрации, совпадающие с иллюстрациями исходного автомата. Соотношения (2), (3) показывают, что формирование динамических (анимационных) иллюстраций в состоянии *state* происходит таким образом, что начальная анимационная иллюстрация совпадает со статической иллюстрацией в предыдущем состоянии автомата, а конечная – со статической иллюстрацией в состоянии *state*.

Из изложенного следует, что рисовальщик в визуализаторе в общем случае будет реализовываться посредством не одного, а двух операторов *switch*. При этом первый оператор *switch* будет соответствовать статическим иллюстрациям, а второй – динамическим.

Таким образом, первые четыре перечисленных шага вводятся в указанную выше технологию между девятым и десятым этапами, а пятый шаг – между одиннадцатым и двенадцатым этапами. При этом этапы технологии с первого по девятый не изменяются, а дальнейшие шаги преобразуются следующим образом.

10. *Выбор состояний, в которых будет выполняться анимация.*

11. *Замена каждого из выбранных состояний тремя состояниями.*

12. *Использование автомата анимации.*

13. *Обеспечение взаимодействия между преобразованным и анимационным автоматами.*

14. *Выбор интерфейса визуализатора.*

15. *Сопоставление иллюстраций и комментариев с состояниями автомата.*

16. *Обеспечение выбора в каждом анимационном состоянии статического либо динамического изображения*, зависящего не только от состояния основного автомата, но и от номера шага анимации.

17. *Архитектура программы визуализатора.*

18. *Программная реализация визуализатора.*

Построение визуализатора для алгоритма обхода двоичного дерева

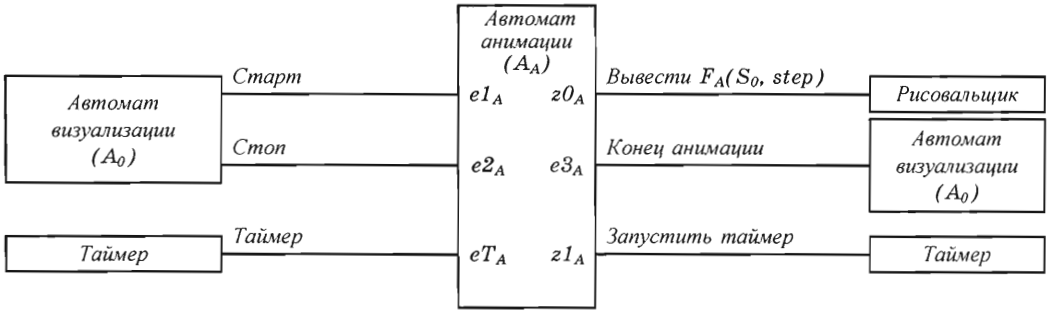
Продемонстрируем технологию на примере построения визуализатора обхода двоичного дерева.

Постановка задачи обхода двоичного дерева. Задано двоичное дерево, состоящее из *n* вершин, каждая из которых имеет ссылки на правого и левого потомка. Требуется реализовать алгоритм, который выводит по одному разу все вершины этого дерева [7].

Решение задачи. Приведем словесную формулировку решения этой задачи.

Для описания вершины дерева введем следующую структуру данных:

```
public class Node {
   private int left;  // левое поддерево
   private int right; // правое поддерево
```



■ *Рис. 4. Схема связей автомата анимации*

```
    private int index; // индекс в массиве
    ...
}
```

Введем массив `visited[0..n-1]`, в котором будем отмечать пройденные вершины.

Для решения этой задачи предлагается создать вспомогательный стек вершин (*stack*). Далее, двигаясь от корня к листьям, предлагается совершить следующее действие в цикле:

- если в текущей вершине (*node*) левое поддерево не пройдено (`node.left > 0 && !visited[node.left]`), то записать ее в стек и двигаться к левому потомку;
- иначе, если правое поддерево не пройдено (`node.right > 0 && !visited[node.right]`), то записать текущую вершину в стек и двигаться к правому потомку;
- иначе, если в стеке находится хотя бы одно значение, то «снять» с вершины стека предыдущую вершину и двигаться к ней;
- иначе завершить алгоритм.

Выбор визуализируемых переменных. Предлагается визуализировать следующие переменные:

- `stack` – стек вершин от корня до текущей вершины;
- `visited` – массив флагов для идентификации пройденных вершин;
- `result` – результирующий массив;
- `node` – текущая вершина.

Выбор алгоритма визуализации. Предлагается визуализировать следующие особенности алгоритма для обеспечения возможности наилучшего разъяснения его действия:

- процесс занесения вершины в стек;
- текущая вершина;
- процесс перехода из предыдущей в следующую вершину;
- результат выполнения алгоритма;
- пройденные вершины (отмечать цветом).

Алгоритм решения задачи. Приведем алгоритм решения задачи на псевдокоде [10]:

```
1  node Я nodes[0]
2  добавить node в результат
3  while true
4    do
5       отметить node.left как пройденную
6       if node.left ≥ 0 and node.left не пройдена
7            then
8                 добавить node в стек
9                 node ← node.left
10                добавить node в результат
11      else if node.right ≥ 0 and node.right не пройдена
12           then
13                добавить node в стек
14                node ← node.left
15                добавить node в результат
16      else if стек не пуст
17           then node ← с вершины стека
18      else break
19 return результат
```

Реализация алгоритма на языке *Java*. Перепишем программу, записанную на псевдокоде, с помощью языка *Java*:

```
/**
 * @param nodes массив вершин дерева
 * @return список всех вершин, выведенных в порядке обхода
 */
private static List traverse(Node[] nodes) {
    // Содержит результат
    List result = new ArrayList();
    // Стек для обеспечения возврата
    LinkedList stack = new LinkedList();
    // Хранят флаги того, что вершина пройдена
    boolean[] visited = new boolean[nodes.length];
    // Заполняем исходными значениями
    Arrays.fill(visited, false);
    // Выбираем корневую вершину
    Node node = nodes[0];
    result.add(node);
    while (true) {
        // Отмечаем текущую вершину, как пройденную
        visited[node.index()] = true;
        if (node.left() >= 0 && !visited[node.left()]) {
            // Переход к левому поддереву
            stack.addLast(node);
            node = nodes[node.left()];
            result.add(node);
        } else if (node.right() >= 0 && !visited[node.right()]) {
            // Переход к правому поддереву
            stack.addLast(node);
            node = nodes[node.right()];
            result.add(node);
        } else {
            // Возврат
            if (stack.isEmpty()) {
                break;
            }
            node = (Node)stack.removeLast();
        }
    }
    return result;
}
```

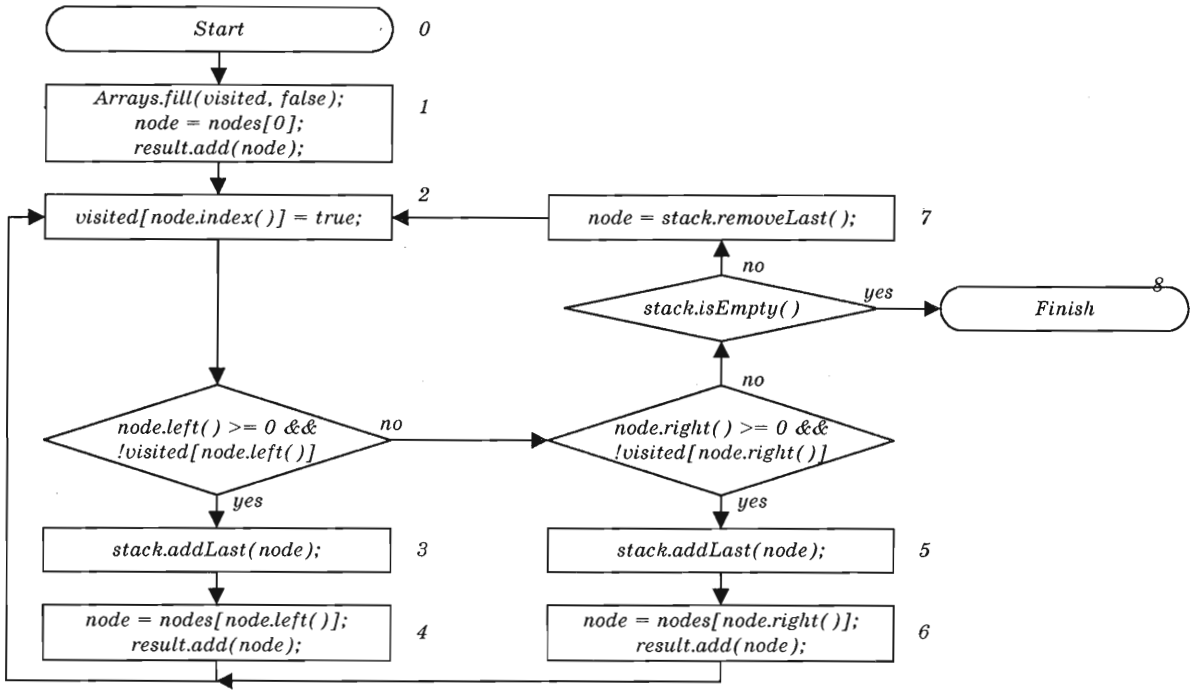
В этой программе операции ввода/вывода не приведены, так как их будет выполнять визуализатор.

Построение схемы алгоритма по программе. Построим по тексту программы схему алгоритма (рис. 5).

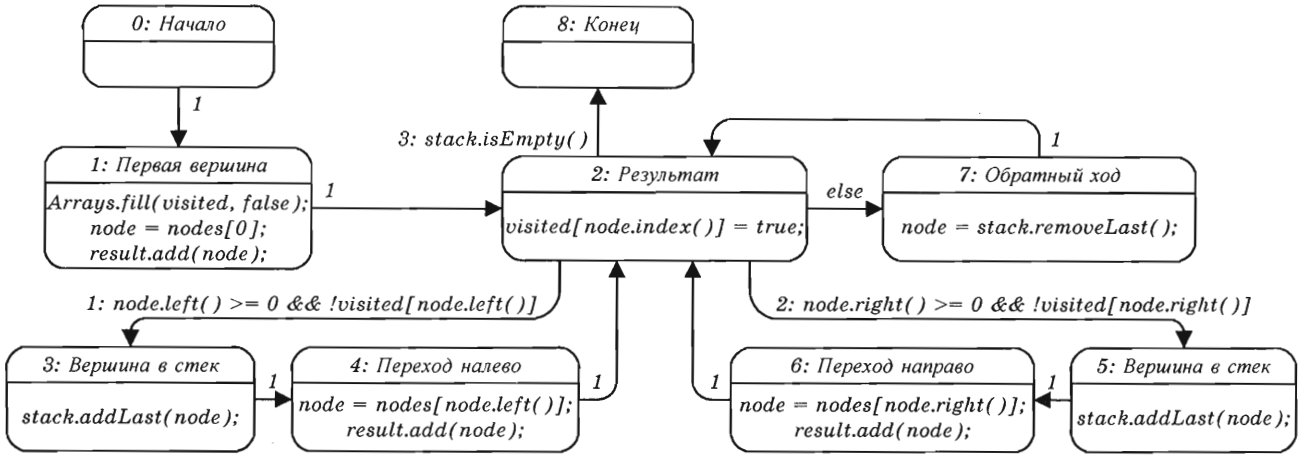
Преобразование схемы алгоритма в автомат Мура. Следуя методу, приведенному в работе [8], построим автомат Мура, соответствующий приведенной выше схеме алгоритма (рис. 6).

Исходя из схемы алгоритма на рис. 4, выделим восемь состояний. В состоянии 1 осуществляется инициализация алгоритма. В состоянии 2 текущая вершина отмечается как пройденная. В состояниях 3 и 5 текущая вершина помещается в стек, а в состояниях 4 и 6 осуществляется переход. Состояние 7 отвечает за обратный ход. Состояние 8 является конечным.

Преобразование автомата, реализующего алгоритм в автомат визуализации. В соответствии с методом, схема которого представлена на рис. 1,



■ *Рис. 5. Схема алгоритма двоичного обхода дерева*



■ *Рис. 6. Автомат Мура, реализующий обход двоичного дерева*

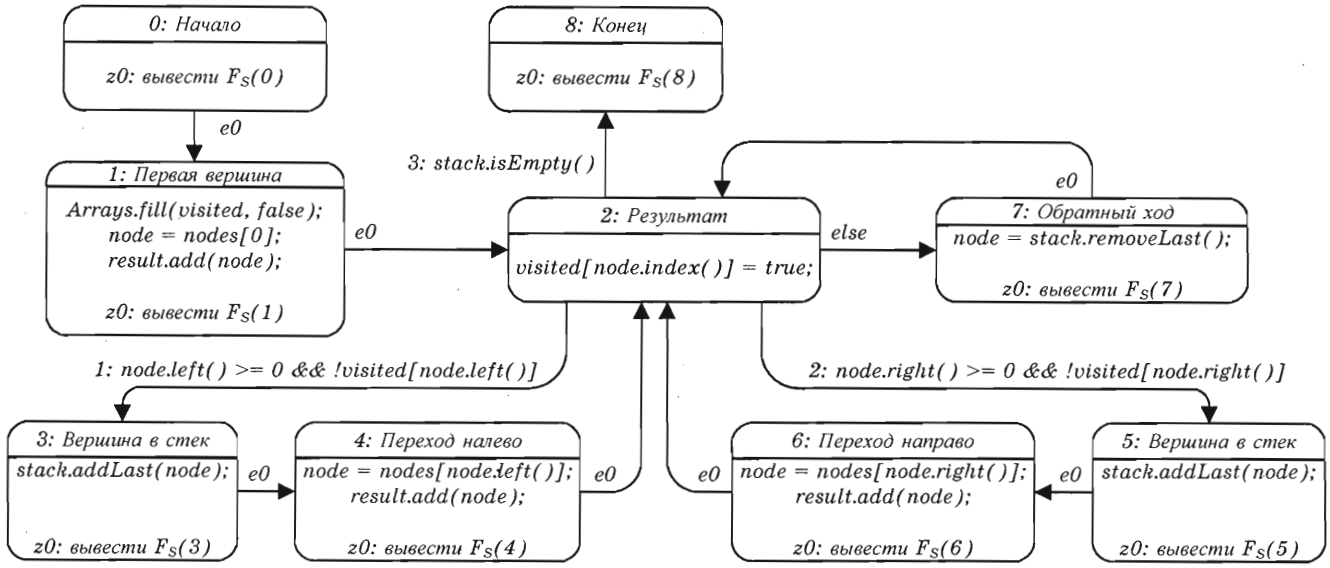
автомат, изображенный на рис. 6, преобразуется в автомат визуализации (рис. 7).

Выбор состояний, в которых будет выполняться анимация. В алгоритме обхода дерева наиболее важным для отображения является процесс передвижения по вершинам, поэтому анимационными являются состояния, в которых изменяется номер текущей вершины. Такими состояниями являются состояния с номерами 4, 6, 7.

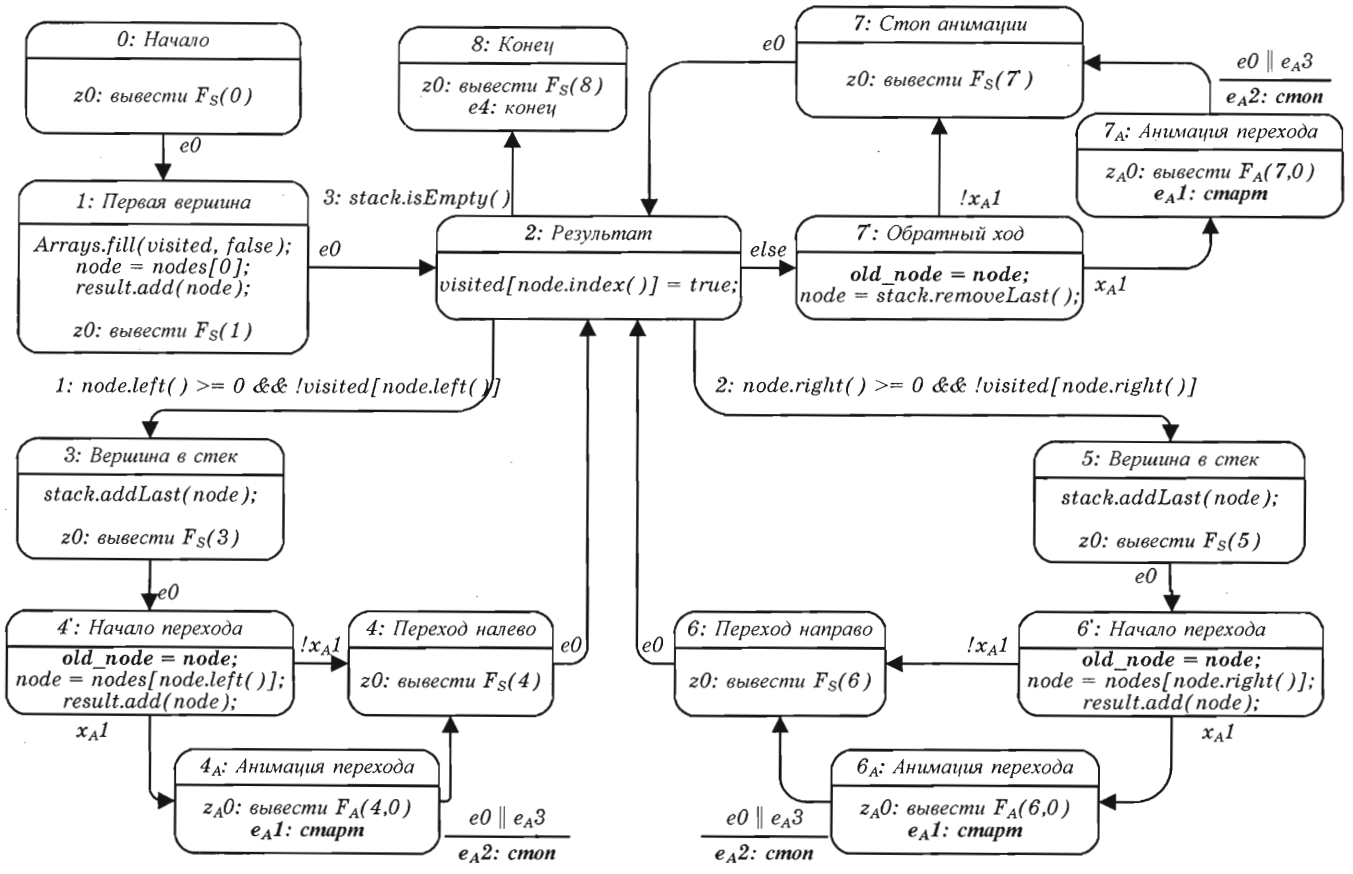
Замена каждого из анимационных состояний тремя состояниями. В соответствии с методом, изложенным выше (см. рис. 2), заменим состояния 4, 6, 7 тройками состояний, в первом из которых запоминается номер предыдущей вершины, во втором формируется событие начала анимации, а в третьем завершается шаг визуализатора и создается конечная иллюстрация в состоянии (рис. 8). При этом событие окончания анимации формируется на переходе между анимационным и конечным состояниями шага визуализации.

Использование автомата анимации. Как отмечалось выше, для анимации используется стандартный автомат, приведенный на рис. 3.

Обеспечение взаимодействия между преобразованным и анимационным автоматами. Для обеспечения взаимодействия между автоматом визуализации и автоматом анимации вводятся связи, представленные на рис. 4.



■ ***Рис. 7.*** *Автомат визуализации*

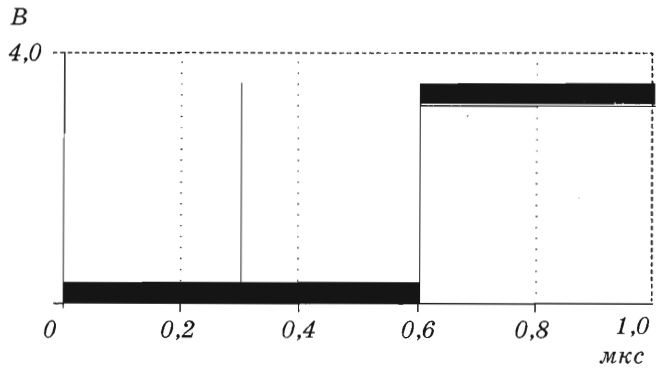


■ ***Рис. 8.*** *Преобразованный автомат визуализации*

Выбор интерфейса визуализатора. В верхней части визуализатора (рис. 9) приводится следующая информация:

• дерево, изображенное графически. Серым цветом отмечены пройденные вершины, белым цветом – непройденные. Текущая вершина отмечена темно-серым цветом;

• в правой части иллюстрации изображен стек, а в нижней части – порядок обхода дерева (массив пройденных вершин).



■ ***Рис. 15.*** *Результаты Монте-Карло SPICE-моделирования для устройства на рис. 14 (200 прогонов)*

В схеме на рис. 14 транзистор М3 соответствует переменной $a_0 = z_a$ с единичным весом. Пара транзисторов М10, М11 (М14, М15) соответствует переменным $z_b(z_c)$, причем открытый транзистор М11 (М15) моделирует набор $I(z_b) = 1(I(z_c) = 1)$, а открытая пара транзисторов М10, М11 (М14, М15) моделирует $I(z_b) = 15$ $(I(z_c) = 15)$. В эксперименте была реализована следующая последовательность состояний $[x, I(z_a), I(z_b), I(z_c)]$:

$$[0,0,15,15] \underset{\substack{t=0\div300\text{ нс}\\ Y=0}}{\Rightarrow} [0,0,1,1\} \underset{\substack{t=300\div600\text{ нс}\\ Y=0}}{\Rightarrow} [0,1,1,1] \underset{\substack{t=600\div1000\text{ нс}\\ Y=1}}{\quad} . \quad (30)$$

На рис. 15 приведены результаты 200 прогонов Монте-Карло SPICE-моделирования схемы на рис. 14 для 10 %-го разброса параметров транзисторов:

– TOX (тонкий окисел) Dev 2,5 %, Lot 2,5 %,[1]

– VTO (порог) Dev 2,5 %, Lot 2,5 %.

Из рис. 15 следует, что реализуемость рассмотренного устройства вполне удовлетворительна[2]. Как показано в работе [10], для 10 %-го разброса параметров устойчивая работа β-управляемого порогового элемента гарантирована при значениях порога $\eta \le \min(3, \sum_j \omega_j - 3)$. Это условие, как видно из результатов моделирования, сохраняется и для пороговых элементов с функциональными входами.

Результаты эксперимента также позволяют надеяться, что аналогично обычному β-управляемому пороговому элементу использование процедуры обучения для формирования V_{ref1} [13–16] позволит поднять критическое значение порога с 3 до более чем 100. Это предположение основывается на том, что при использовании процедуры обучения все технологические вариации параметров компенсируются в процессе обучения и на критическую величину порога начинают влиять только вариации эксплуатационных характеристик (температура, напряжение источника питания и т. д.) и точность стабилизации тока в функциональном входе.

[1] Dev – разброс внутри одного lot, Lot – разброс между lots.

[2] Всплеск в момент времени $t = 300$ нс – помеха, связанная с одновременным переключением транзисторов М10 и М14, что эквивалентно одновременному изменению значений 30 переменных.

Заключение

Приведенное выше ограничение на значение порога может вызвать разочарование в перспективности использования β-управляемого порогового элемента как такового и β-управляемого порогового элемента с функциональными входами в частности. Природу невозможно обмануть, и асимптотические оценки сложности логических схем являются их очевидным свойством. С другой стороны, на практике мы имеем дело не со схемами вообще, а с практически необходимыми схемами, которые, именно в силу их практической ориентации, обладают, как правило, некоторой внутренней организацией. Если эта организация толерантна пороговой логике, то можно надеяться на эффективность применения пороговых элементов. Для подтверждения сказанного обратимся к примеру.

Во многих применениях весьма эффективным оказывается использование равновесных кодов «k из n» или «k и только k из n» – это и асинхронная передача данных (самосинхронные коды) [21–22], и задачи тестирования (равновесное кодирование) [22], и многое другое. В этих задачах необходимо синтезировать индикаторы (чекеры), распознающие такие коды.

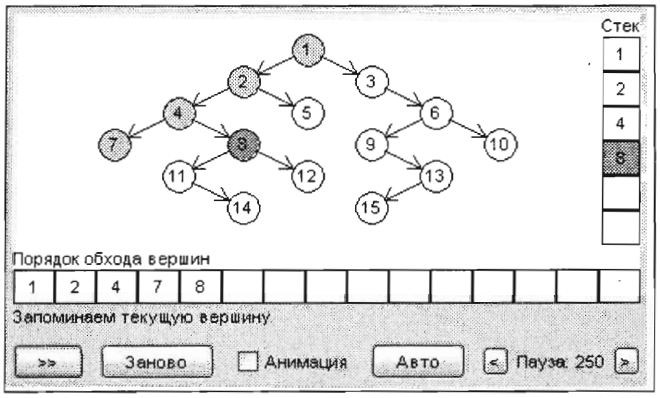
Пусть нам необходимо реализовать чекер кода «6 и только 6 из 12». Функция, реализуемая чекером, при этом определяется, как

$$Ch_{12}^{6}(x_0, x_1, \ldots, x_{11}) = \begin{cases} 1, \text{ если } \sum_{j=0}^{11} x_j = 6 \\ 0, \text{ если } \sum_{j=0}^{11} x_j \ne 6 \end{cases} . \quad (31)$$

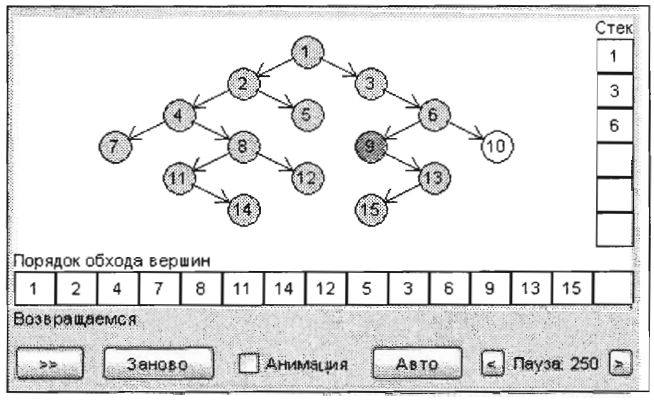
Минимальная дизъюнктивная форма этой функции содержит 924 терма ранга 12. Использование декомпозиции снижает сложность реализации этой функции в булевом базисе, однако это снижение не принципиально и сохраняет порядок сложности.

Рассмотрим реализацию чекера (31) в пороговом базисе с функциональными входами при ограничении $\min(\eta, \sum_j \omega_j - \eta) \le 3$.

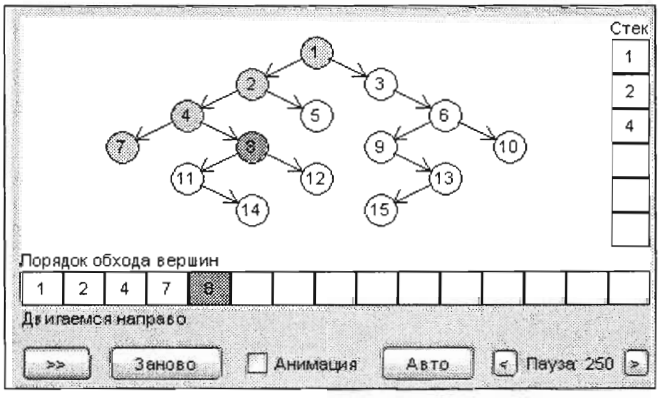
Функция чекера (31) немонотонна, однако она достаточно просто выражается через монотонные (пороговые):



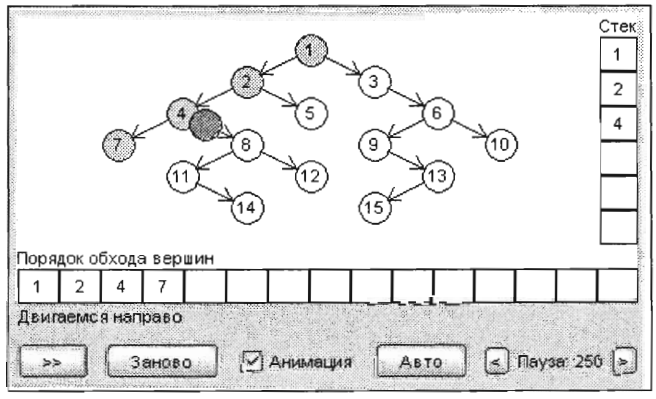
■ *Рис. 9. Иллюстрация в состояниях 3 и 5*



■ *Рис. 11. Иллюстрация в состоянии 7*



■ *Рис. 10. Интерфейс визуализатора*



■ *Рис. 12. Динамическая иллюстрация*

Под массивом пройденных вершин динамически отображаются текстовые комментарии, соответствующие текущим состояниям автомата визуализации. Например, на рис. 10 изображен комментарий «Двигаемся направо», соответствующий состоянию 6. Под комментарием расположены элементы управления. Следует отметить, что по сравнению с обычным визуализатором появляется флаг управления анимацией:

«>>» – сделать шаг алгоритма;

«Заново» – начать алгоритм сначала;

«Анимация» – отображать или не отображать анимацию;

«Авто» – перейти в автоматический режим;

«<», «>» – изменить паузу между автоматическими шагами алгоритма.

Сопоставление иллюстраций и комментариев с состояниями автомата. Для наглядности визуализации алгоритма предлагается акцентировать внимание на следующих элементах.

В состояниях 4 и 6 (см. рис. 10) статическая иллюстрация отображает: текущую вершину; пройденные вершины; последнюю вершину в результирующем массиве.

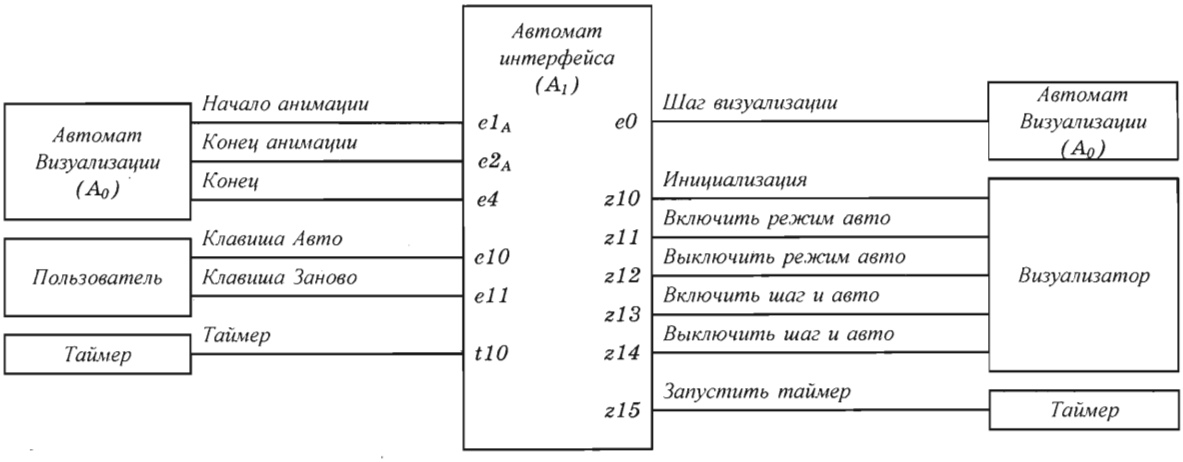
В состояниях 3 и 5 (см. рис. 9) отмечается вершина, добавленная в стек.

В состоянии 7 (рис. 11) отображается обратный ход. При этом отмечается только текущая вершина, поскольку данные в стек и массив пройденных вершин не добавляются.

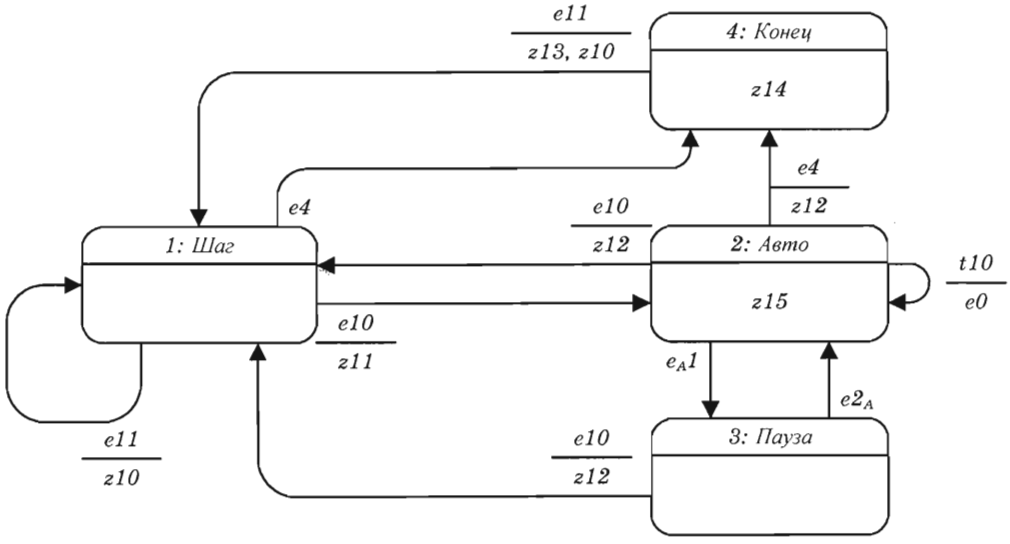
Обеспечение выбора в каждом анимационном состоянии статического либо динамического изображения. Для отображения динамических иллюстраций используются статические иллюстрации с наложенным изображением движущегося указателя (рис. 12).

Архитектура программы визуализатора. Для реализации пользовательского интерфейса сформирован еще один автомат, реализующий поведение интерфейса визуализатора. Схема взаимодействия этого автомата приведена на рис. 13.

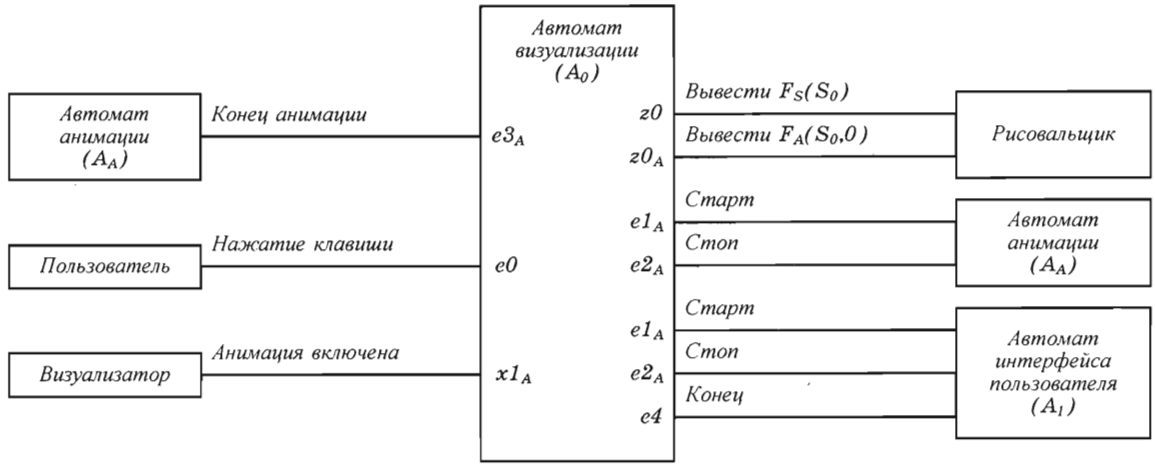
Как следует из этого рисунка, автомат взаимодействует с автоматом визуализации и управляет интерфейсом визуализатора. Таймер используется для реализации функции *Авто*. Диаграмма переходов интерфейсного автомата приведена на рис. 14.



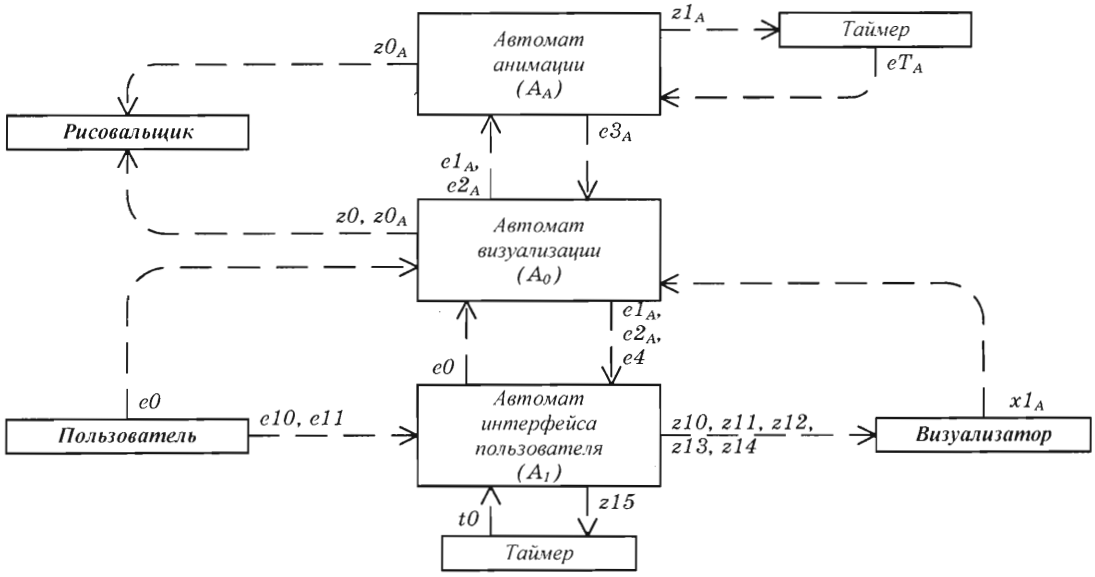
■ ***Рис. 13.*** *Схема взаимодействия автомата интерфейса визуализатора*



■ ***Рис. 14.*** *Диаграмма переходов автомата интерфейса визуализатора*



■ ***Рис. 15.*** *Схема взаимодействия преобразованного автомата визуализации*



■ ***Рис. 16.*** *Схема взаимодействия компонент визуализатора*

Схема взаимодействий преобразованного автомата визуализации изображена на рис. 15.

Схема взаимодействия всех компонент визуализатора представлена на рис. 16.

Заключение

В статье предложен подход к реализации визуализаторов алгоритмов с анимацией, улучшающий восприятие алгоритмов обучающимися, что особенно важно при дистанционном обучении. Подход расширяет технологию построения визуализаторов алгоритмов на основе автоматного подхода, предложенную в работе [5].

Литература

1. **Интернет-школа** программирования. http://ips.ifmo.ru
2. **Казаков М. А., Мельничук О. П., Парфенов В. Г.** Интернет школа программирования в СПбГИТМО (ТУ). Реализация и внедрение // Материалы Всероссийск. научно-методич. конф. «Телематика'2002». – СПб., 2002. – С. 308–309. http://tm.ifmo.ru/tm2002/db/doc/get_thes.php?id=170
3. **Столяр С. Е., Осипова Т. Г., Крылов И. П., Столяр С. С.** Об инструментальных средствах для курса информатики // Труды II Всероссийской конференции «Компьютеры в образовании». – СПб., 1994. – С.18–19.
4. **Казаков М. А., Столяр С. Е.** Визуализаторы алгоритмов как элемент технологии преподавания дискретной математики и программирования // Материалы международн. научно-методич. конф. «Телематика'2000». – СПб., 2000. – С.189–191.
5. **Казаков М. А., Шалыто А. А.** Использование автоматного программирования для реализации визуализаторов // Компьютерные инструменты в образовании. 2004. – № 2. – С.19–33. http://is.ifmo.ru, раздел «Статьи».
6. **Корнеев Г. А., Шамгунов Н. Н., Шалыто А. А.** Обход деревьев на основе автоматного подхода // Компьютерные инструменты в образовании. – 2004. – № 3. – С. 32–37. http://is.ifmo.ru, раздел «Статьи».
7. **Шень А.** Программирование: теоремы и задачи. – М.: МЦНМО, 2004. – 296 с.
8. **Шалыто А. А., Туккель Н. И.** Преобразование итеративных алгоритмов в автоматные // Программирование. – 2002. – № 5. – С. 12–26. http://is.ifmo.ru, раздел «Статьи».
9. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. – СПб.: Наука, 1998. – 628 с.
10. **Кормен Т., Лайзерсон Ч., Ривест Р.** Алгоритмы. Построение и анализ. – М.: МЦНМО, 2000. – 960 с.