Министерство образования и науки РФ
ФГБОУ ВО «Тверской государственный университет»
Факультет прикладной математики и кибернетики
Кафедра информационных технологий
Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»
Профиль «Инженерия программного обеспечения»

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Исследование алгоритмических проблем для клеточных автоматов

Автор:
Орехов Павел Сергеевич

Научный руководитель:
д.ф.-м.н., доцент,
Дудаков Сергей Михайлович

Допущен (а) к защите:

Руководитель ООП:

_____/Язенин А.В./

(подпись, дата)

Заведующий кафедрой: _____

(наименование)

_____/ФИО/

(подпись, дата)

Тверь 2016

Задание на выпускную квалификационную работу.

Исследовать алгоритмические проблемы для клеточных автоматов с ограничением алфавита.

Содержание

Введение

Актуальность и обзор литературы

В данной работе рассмотрены некоторые алгоритмические проблемы, касающиеся одномерных клеточных автоматов. Идея клеточного автомата восходит к работам фон Неймана в 1940х годах[4]. Наибольший вклад в теорию клеточных автоматов был внесен Стивеном Вольфрамом и оформлен в его книге «A new kind of science»[1], однако Вольфрам в основном заостряет свое внимание на элементарных клеточных автоматах, с алфавитом $\Sigma = \{0,1\}$. Здесь, помимо элементарных, также рассмотрены и автоматы с большим алфавитом, а именно 3х и более символьные автоматы.

Цели и задачи работы

Основной мотивацией данной работы является исследование алгоритмической проблемы, называемой «проблема остановки»[6] которая при подаче входного слова w, вычислительному устройству, спрашивает: «Остановится ли вычислительное устройство на данном входном слове?»

В некоторых статьях Вольфрама и людей, связанных с ним, приведены доказательства универсальности некоторых клеточных автоматов и других вычислительных устройств[1][3][5], однако они используют немного другое понятие универсальности, отличное от основного, неформально оно звучит так: «Для каждой вычислимой функции f существует программа P, такая, что моделируя f на P мы получим интересующий нас ответ на каком-то такте работы вычислительного устройства»[3]. Заметим, что четкого понятия остановки здесь нет, то есть Вольфрама в основном интересует можем ли мы получить интересующий нас результат на каком-то такте работы автомата, а дальше этот автомат может вычислять что угодно, то есть главная мотивации состоит в том, что ответ там есть. Однако в данной работе критерий остановки считается обязательным и универсальность определена следующим образом: «Для каждой

вычислимой функции f существует программа P, такая, что моделируя f на P мы всегда останавливаемся и получаем интересующий нас результат».

Итак, главными целями моей работы являются:

1) Изучение клеточных автоматов.
2) Изучение методов алгоритмического решения различных проблем.
3) Исследование проблемы остановки.
4) Исследование других проблем.

Структура работы

Глава 1

В данной главе введены основные понятия, использованные в работе, такие, как клеточный автомат, проблема остановки, конфигурация ленты автомата, этим самым выполнен пункт 1 из списка целей моей работы.

Глава 2

Тут сначала рассмотрены основные способы доказательства разрешимости/неразрешимости проблемы остановки для какого-либо вычислительного устройства (пункт 2 из списка целей моей работы). Затем на примере показано, почему проблема остановки является разрешимой для двухсимвольных клеточных автоматов и некоторых трехсимвольных. Затем рассмотрены регулярные структуры, порождаемые некоторыми двухсимвольными клеточными автоматами (пункт 4 из списка целей моей работы), всего рассмотрено 4 автомата, но также приведены номера похожих автоматов, для которых доказательства эквивалентны.

В пункте 2.3 данной главы получен наиболее важный результат о том, что проблема остановки неразрешима для пятнадцатисимвольных клеточных автоматов. Сделано это путем сведения минимальной универсальной машины Тьюринга к данному семейству клеточных автоматов (пункт 3 из списка целей моей работы).

Глава 3

Здесь представлена программа, которая была мной написана и которую я использовал при изучении клеточных автоматов, и сделан обзор технологий, использованных для написания данной программы. Также указаны ссылки, где можно скачать программный код и «собрать» данную программу.

1. Основные понятия

Определение 1. Пусть $\Sigma -$некоторое непустое конечное множество (алфавит), будем называть $s \in \Sigma^*$ строкой или словом, а так же считать, что $\lambda \in \Sigma^*$ и называть его пустым словом. Тогда клеточный автомат $-$ это пара $\langle \Sigma, \phi \rangle$, где $\phi: \Sigma^3 \to \Sigma -$ функция, называемая программой автомата, которая работает в соответствии с правилами автомата, количество которых $|\Sigma|^3$.

Определение 2. Конфигурация ленты автомата $-$это функция $f: \mathbb{N} \to \Sigma$, которая говорит, что написано на ленте.

Определение 3. Пусть $f_i -$ это конфигурация ленты автомата после i шагов. Тогда будем говорить, что автомат остановился в конфигурации f_i, если $(\forall j \in \mathbb{N}) j > i \to (\forall k \in \mathbb{N}) f_i(k) = f_j(k)$.

Определение 4. Основание – строка, написанная на ленте автомата, после какого-то количества проделанных шагов (автомат по сути рисует какую-то фигуру).

Определение 5. Коэффициент масштабирования – число x из функции $f(x) = a * x^n + c$. Показывает во сколько раз увеличился узор, порожденный автоматом, в размере, при переходе от шага $f(n)$ к шагу $f(n+1)$.

Также иногда будут использованы следующие сокращения:

1) КА $-$ клеточный автомат.
2) МТ$-$ машина Тьюринга.
3) NKA $-$ N-символьный клеточный автомат.

2. Проблема остановки для клеточных автоматов

2.1. Проблема остановки для двух и трехсимвольных клеточных автоматов

Основным способом доказательства неразрешимости проблемы остановки является сведение. Когда мы сводим вычислительное устройство A к вычислительному устройству B, мы показываем способ моделирования работы A, на B. Если мы, например, сведем, машину Тьюринга к какому-то вычислительному устройству, то мы докажем, что для него проблема остановки неразрешима.

Доказать разрешимость проблемы остановки для вычислительного устройства A можно, например, путем сведения его к устройству B, для которого проблема остановки разрешима. То есть, нужно показать способ моделирования A на B.

Конечно же существуют и другие способы доказательства разрешимости/неразрешимости этой проблемы. Например, для машины Тьюринга это доказательство приводится через самоприменимость.

Далее, как пример, рассмотрим, почему проблема остановки разрешима для двухсимвольных клеточных автоматов.

Теорема 2.1. Проблема остановки разрешима для элементарных (двухсимвольных) КА.

Доказательство.

Будем использовать «0» для обозначения белой клетки и «1», для обозначения черной.

Если в автомате присутствуют правила двух видов: $1) 000 \to 1$ и $111 \to 0, 2) 001 \to 1$ или $100 \to 1$, мы знаем, что автомат не остановится.

В первом случае, так как лента бесконечна в оба края, всегда найдутся 3 белые клетки, которые произведут черную, по правилу $000 \to 1$, после чего сработает правило $111 \to 0$, выводя, опять же, белые клетки под тройками чёрных.

Во втором случае, слово будет расти в каждый бок на единицу, и автомат никогда не остановится.

В оставшихся случаях можно дать автомату проделать 2^n шагов (т.к. существует 2^n бинарных чисел длины n) и посмотреть, остановится он или нет. Если он остановится, то всё хорошо, он остановился. Если он не остановится, тогда он попал в цикл и выводит повторяющиеся конфигурации, более формально: начальная конфигурация выводит конфигурацию x, x выводит y, а y опять выводит x, поэтому автомат не остановится.

Теорема 2.2. Проблема остановки разрешима для трехсимвольных клеточных автоматов, в которых отсутствуют правила следующих видов: $1) 002 \to 1$ и $001 \to 0, 2) 001 \to 2$ и $002 \to 0$.

Доказательство.

Очевидно, что КА не остановится если имеются правила $000 \to 1$ и $111 \to 0$ или $000 \to 2$ и $222 \to 0$. Но правила вида $00x \to y$ и $x00 \to y$ анализировать намного сложнее, как видно из текста теоремы. Можно рассмотреть следующие комбинации таких правил:

$$1.\, 001 \to 0 \text{ и } 002 \to 0$$

$$2.\, 001 \to 0 \text{ и } 002 \to 1$$

$$3.\, 001 \to 0 \text{ и } 002 \to 2$$

$$4.\, 001 \to 1 \text{ и } 002 \to 0$$

$$5.\, 001 \to 1 \text{ и } 002 \to 1$$

$$6.\, 001 \to 1 \text{ и } 002 \to 2$$

$$7.\, 001 \to 2 \text{ и } 002 \to 0$$

$$8.\, 001 \to 2 \text{ и } 002 \to 1$$

$$9.\ 001 \to 2 \text{ и } 002 \to 2$$

Симметричные правила вида $x00 \to y$ были опущены.

В первом случае очевидно, что слово не будет разрастаться по бокам, потому что данные правила порождают «0».

В случаях 5, 6, 8, 9 очевидно, что автомат не остановится, потому что слово будет разрастаться в бока.

Случаи 2, 3, 4, 7 являются более интересными. Заметим, что случай 2 «похож» на случай 7, и случай 3 «похож» на случай 4. Поэтому целесообразно рассмотреть только, например, случаи 2 и 3, для краткости. Для начала, рассмотрим случай 3, потому что он более лёгкий. Итак, имеются правила $001 \to 0$ и $002 \to 2$. Очевидно, что если первый и последний символы входного слова «2», то явно, что КА не остановится. Но если они «1», то нужно рассмотреть более подробно.

Рассмотрим, когда первые и последние символы входного слова могут вывести «2» по его краям, т.к. если появляется «2» то заключаем, что автомат не остановится.

Комбинации, которые надо рассмотреть:

$$010 \to 0, 011 \to 0, 012 \to 0$$

$$010 \to 0, 011 \to 0, 012 \to 1$$

$$010 \to 0, 011 \to 0, 012 \to 2$$

$$010 \to 1, 011 \to 1, 012 \to 0$$

итд.

Рассмотрим, что происходит, если имеется первая тройка, сверху. Пусть на ленте написано слово w. Первый и последний символы «1». Имеем правила $010 \to 0, 011 \to 0, 012 \to 0$ (первая тройка). Тут очевидно, что при каждом такте работы

автомата слово, написанное на ленте, будет сужаться на 2 символа, потому что данные правила стирают первый и последний символы. Но если в какой-то момент времени мы получим «2», тогда сработает правило $002 \to 2$, и мы заключим, что КА не остановится. В итоге можно дать автомату сделать $|w|/2$ шага, и если слово становится пустым, то автомат остановится, если нет, то не остановится.

Теперь рассмотрим более обобщенный случай 3. Нетрудно заметить, что в данном случае можно дать автомату проделать 3^n шагов, и если на хотя бы одном из них, на краю слова написана «2», то автомат не остановится. Если этого не происходит, то автомат попал в цикл и так же не остановится. Иначе автомат остановится.

Рассмотрим случай 2. Имеем правила $001 \to 0$ и $002 \to 1$. На данный момент не известно как с этим быть. Рассмотрим следующие правила:

$$010 \to 0, 011 \to 0, 012 \to 0$$

$$010 \to 0, 011 \to 0, 012 \to 1$$

$$010 \to 0, 011 \to 0, 012 \to 2$$

$$010 \to 0, 011 \to 1, 012 \to 0$$

итд.

Данные правила были выписаны так как даже если у входного слова на краю написана «2», то очевидно, что на следующем шаге она станет «1», поэтому целесообразно рассматривать именно эти правила.

Также очевидно, что можно не волноваться о тройках, которые не порождают «2», потому что слово не будет разрастаться (может быть за исключением первого

шага), и можно дать автомату проработать $3^n + 1$ шагов и посмотреть остановится он или нет.

Рассмотрим способы доказательства разрешимости для случая 2. Рассматривать все комбинации правил для каждой тройки из таблицы нецелесообразно, так как, даже при одной первой интересующей нас тройке $\langle 0,0,2 \rangle$, нужно будет рассмотреть $3^{27-3} = 3^{24} = 282{,}429{,}536{,}481$ комбинацию, и то непонятно что это даст, потому что входные слова могут быть разными.

На данный момент более вероятным является тот факт, что здесь легче всего свести что-то, для чего проблема остановки неразрешима к 3КА, и заключить, что проблема остановки неразрешима. Аналогично можно попробовать свести 3КА к чему-то, для чего проблема остановки разрешима.

2.2. Доказательства фрактального поведения некоторых элементарных клеточных автоматов

Для каждого автомата будет доказано следующее утверждение:

(Главное) утверждение. Автомат A, на входном слове w, будет порождать «хороший» фрактальный узор после $f(n)$ шагов. «Хороший» в том смысле, что будет возможно построить подобие между узором, полученным после $f(n)$ шагов, где $n \in \mathbb{N}$, и узором, полученным после $f(k)$ шагов, где $k \leq n$. Здесь $f(n) = a * x^n + c$.

Автомат 18, входное слово «1»

Похожие автоматы: 22, 26, 60, 82, 90, 94, 122, 126, 129, 133, 146, 151, 153, 154, 161, 167, 181, 182, 183, 195, 210, 218.

Правила автомата:

$$1)\ 000 \rightarrow 0$$

$$2)\ 001 \rightarrow 1$$

$$3)\ 010 \rightarrow 0$$

4) $011 \rightarrow 0$

5) $100 \rightarrow 1$

6) $101 \rightarrow 0$

7) $110 \rightarrow 0$

8) $111 \rightarrow 0$

Рассмотрим следующий рисунок (64 шага автомата):

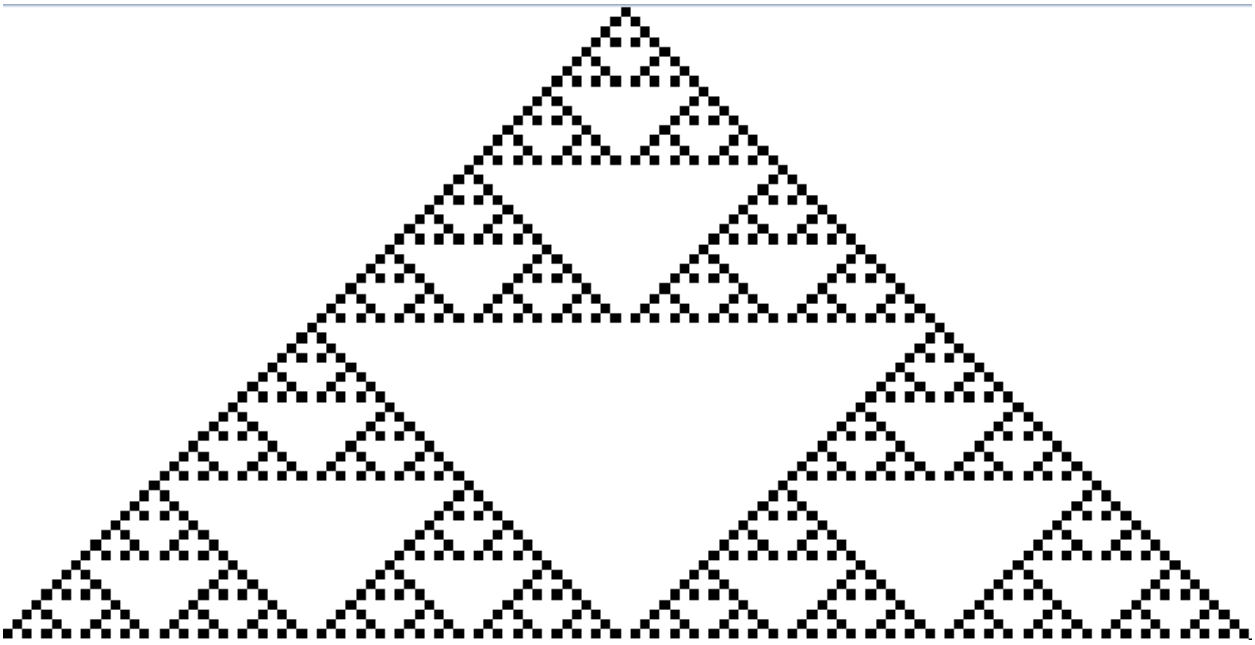

Рис. 1. 64 шага автомата 18

На данном рисунке показан фрактальный узор после 2^{n} шагов автомата, где $\mathrm{n} \in \{1,2,3,4,5,6\}$. Коэффициент масштабирования данного фрактала равен 2, соответственно отсюда следует, что $f(n)$, из главного утверждения, будет равно 2^n.

Автомат, исходя из рисунка 1, устроен следующим образом: сначала, после $f(k)$ шагов, мы имеем треугольник, на шаге $f(k+1)$ выводятся два таких же треугольника по его краям.

Замечание об основании: основание каждого из описанных треугольников является числами Мерсенна, то есть $2^{\mathrm{p}} - 1$.

Лемма 2.1. Автомат 18 превращает последовательность вида $1(01)^{l}$ в последовательность вида $1(0)^{2l+1}1$.

Доказательство. очевидно из правил 2,3,5,6. Заметим, что обе единицы новой строки написаны в позициях, находящихся строго по краям старой строки.

Лемма 2.2. Длина основания треугольника, после 2^{k} шагов, будет равна $2^{k+1} - 1$.

Доказательство. Следует из того факта, что входное слово «1», а из-за правил 2 и 5 узор будет расти в левый и правый бок на один, при каждом шаге.

Теперь докажем главное утверждение по индукции.

Доказательство(по индукции по n).

Базис. $\mathrm{n} = 1$, как видно из рисунка 1, для этого случая утверждение верно.

Индукционный шаг. предположим, что $\mathrm{n} = \mathrm{k}$ верно, докажем, что утверждение верно для $\mathrm{n} = \mathrm{k} + 1$:

Из леммы 2.1 и правил 2 и 5 следует, что после того, как автомат сделает 2^{k} шагов, на следующем шаге, на ленте автомата, будут написаны две единицы, на расстоянии 2^{k+1}.

Теперь рассмотрим почему узор, выводимый из левой единицы, не «столкнется» с узором из правой единицы до шага $2^{\mathrm{k+1}}$.

Так как из-за правил «$001 \rightarrow 1$» и «$100 \rightarrow 1$» узор, выводимый из «1» будет расти в любой бок на одну единицу, при каждом шаге, то через 2^{k} шагов, он вырастет в бок на 2^{k}. Так как на шаге $2^{k} + 1$ единицы разделены $2^{k+1} - 1$

нулевыми клетками, то под основание они зайдут на $2 * (2^k - 1) = 2^{k+1} - 2$, потому что начальные единицы находятся строго по краям основания.

Отсюда следует, что между основаниями новых треугольников будет одна нулевая клетка, и они не «столкнутся». А так как были выведены 2 новых треугольника, идентичных предыдущему, на расстоянии 1 друг от друга, то суммарная длина нового основания будет $2 * (2^{k+1} - 1) + 1 = 2^{k+2} - 1$, что означает, что мы получим новое основание, которое будет соответствовать треугольнику, полученному после 2^{k+1} шагов по лемме 2.2. Конец.

Автомат 150, входное слово «1»

Похожие автоматы: 105.

Автомат имеет следующие правила:

$$1)\ 000 \rightarrow 0$$
$$2)\ 001 \rightarrow 1$$
$$3)\ 010 \rightarrow 1$$
$$4)\ 011 \rightarrow 0$$
$$5)\ 100 \rightarrow 1$$
$$6)\ 101 \rightarrow 0$$
$$7)\ 110 \rightarrow 0$$
$$8)\ 111 \rightarrow 1$$

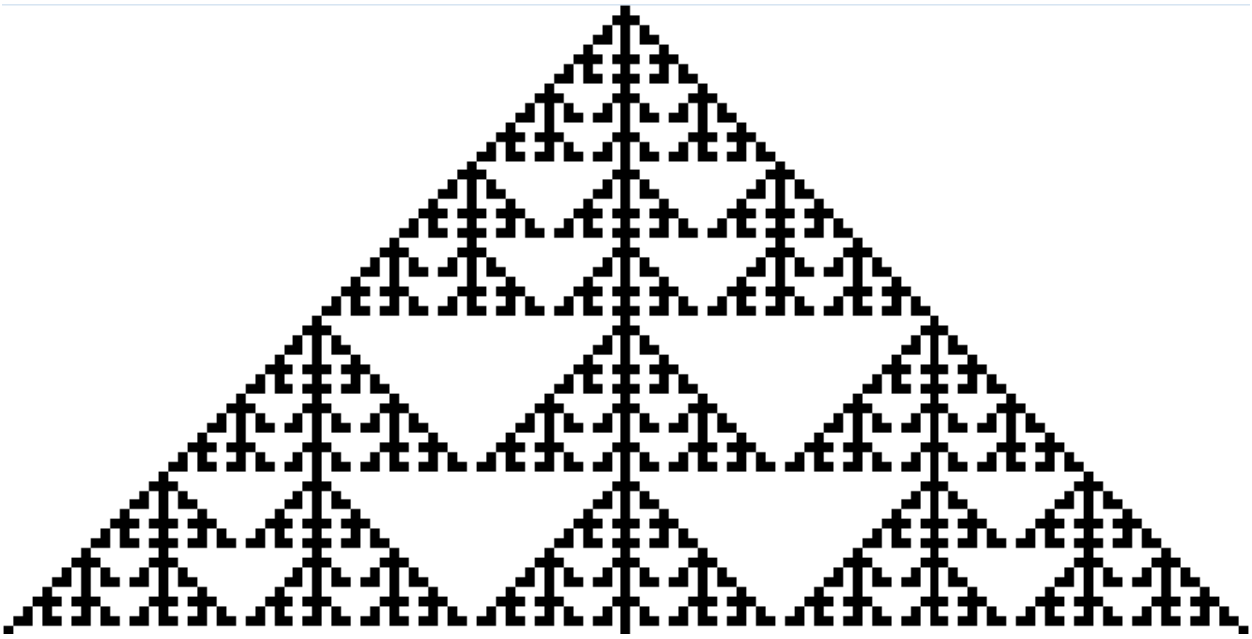

Рис. 2. 64 шага автомата 150

На рис. 2 имеем 64 шага автомата. Так как коэффициент масштабирования данного фрактала равен 2, то целесообразно рассматривать количество шагов, равное степени числа 2, отсюда $f(n) = 2^n$.

Докажем главное утверждение (тут будет рассмотрено следующее: через $2^n + 1$ шагов, после 2^n-го шага, на ленте появится слово «$10^{(2^{n+2}-2)/2}10^{(2^{n+2}-2)/2}1$», длины $2^{n+2} + 1$).

Замечание об основаниях: при нечетных n, у основания будет следующее регулярное выражение: $(110)^{(2^n-2)/3}111(011)^{(2^n-2)/3}$, а при чётных n: $(110)^{(2^n-1)/3}1(011)^{(2^n-1)/3}$. Длина основания, после 2^n шагов, равна $2^{n+1} - 1$. А на шаге $2^n + 1$ выводятся слова «$10^{2^n-1}10^{2^n-1}1$», длины $2^{n+1} + 1$.

Доказательство(по индукции по n).

Базис. $n = 1$, как видно из рисунка 2, для этого случая утверждение верно.

Индукционный шаг. предположим, что $n = k$ верно, докажем, что утверждение верно для $n = k + 1$:

Теперь, для удобства, разобьём предыдущую картинку следующим образом:

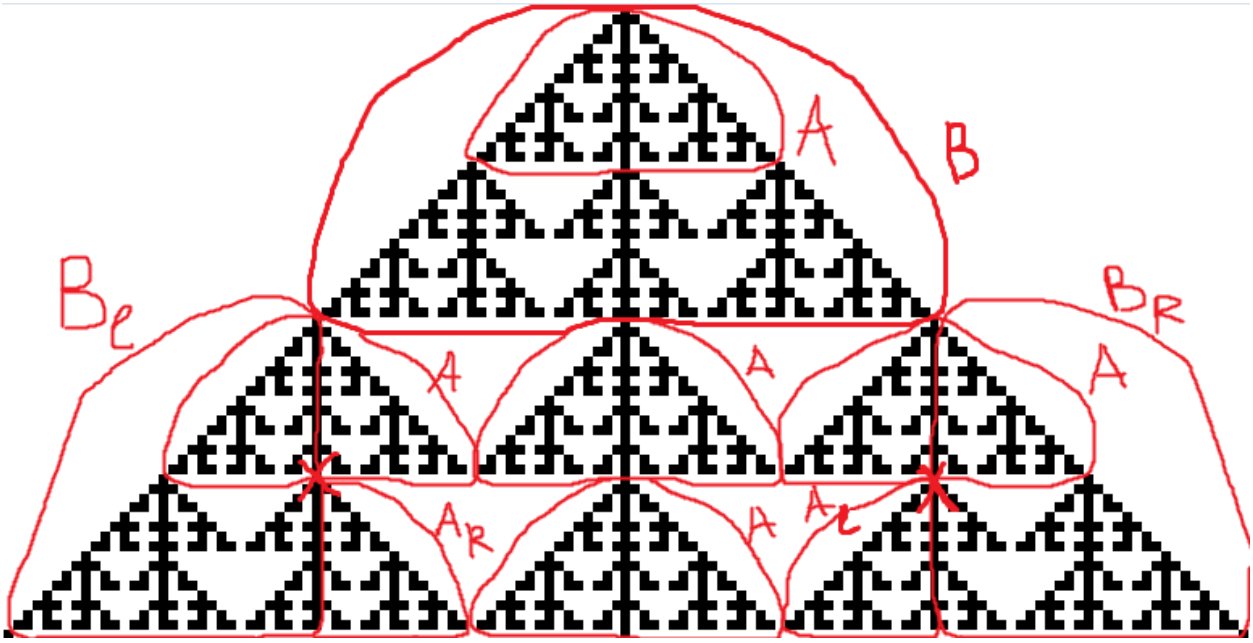

Рис. 3. Выделенные области рис. 2

Предположим, что мы сделали $2^k + 1$ шаг, на ленте написано слово «$10^{2^k-1}10^{2^k-1}1$». Тогда, так как мы имеем 3 единицы, наш рисунок начнет заново себя повторять сначала, в трёх местах. Более конкретно он будет повторять часть «A».

Утверждение. после того, как было сделано 2^k шагов, после еще 2^{k-1} шагов, автомат выведет часть «A» предыдущего узора 3 раза (см. рисунок 3).

Доказательство. пусть на $2^k + 1$м шаге на ленте написана строка с регулярным выражением, описанным в замечании. Тогда автомат будет выводить верхнюю часть, так как входным словом была строка «1». Далее мы сможем проделать $2^{k-1} - 1$ шагов, без столкновения трёх верхних частей, так как расстояние между единицами данной строки $2^k - 1$. Столкновения не произойдет,

потому что части «A» приблизятся друг к другу на $2^{k-1} - 1$ ($2 * (2^{k-1} - 1) = 2^k - 2$, соответственно между ними будет одна свободная клетка). Конец.

Теперь рассмотрим, что происходит далее. До этого момента, после $2^{\mathrm{k}} + 2^{\mathrm{k-1}}$ шагов автомата на ленте написана строка длины $3 * 2^{\mathrm{k-1}} + 2$, так как в местах сочленения треугольников образуется последовательность «101», и наш автомат имеет правило «$101 \rightarrow 0$», то в тех местах появятся нули. Места основания треугольника, которые могут порождать «1» на следующем шаге автомата, это левый конец, середина и правый конец, так как треугольника 3, точек сочленения 2 (в которых участвует 4 конца треугольника), то всего на следующем шаге будет написано 5 единиц.

Далее, из центральной единицы выводится новый треугольник, а пары боковых единиц будут соответствовать продолжению правой части предыдущего треугольника, справа, и левой части предыдущего треугольника, слева (B_l и B_r на рисунке 3). Заметим, что центральная часть, части «A», каждого треугольника, соответствует центральной части, нижней части B, вплоть до последнего уровня (что также ясно из замечания о регулярных выражениях оснований). Далее, после еще $2^{\mathrm{k-1}}$ шагов справа и слева будут выводиться правая и левая части предыдущего треугольника, начиная с точек, помеченных «X», а внутри будут выводиться по середине часть «A» предыдущего треугольника, и правая и левая ее части от точек «X» (A_l и A_r на рисунке 3).

Так как центральная часть оснований чередуется, то в месте сочленения A_l и B_r (так же B_l и A_r) появится последовательность «110» или «011», в зависимости от четности степени двойки проделанных шагов, что выведется в «0», из-за правил «$110 \rightarrow 0$» и «$011 \rightarrow 0$». И так мы проделали $2^{\mathrm{k+1}} + 1$ шагов, и у нас получилось новое слово, «10…010…01» длины $2^{\mathrm{k+2}} + 1$, что соответствует основанию треугольника, после $2^{\mathrm{k+1}} + 1$ шагов. Конец.

Автомат 106, входное слово «11»

Похожие автоматы: 120,169,225.

Автомат имеет следующие правила:

$$1)\ 000 \rightarrow 0$$
$$2)\ 001 \rightarrow 1$$
$$3)\ 010 \rightarrow 0$$
$$4)\ 011 \rightarrow 1$$
$$5)\ 100 \rightarrow 0$$
$$6)\ 101 \rightarrow 1$$
$$7)\ 110 \rightarrow 1$$
$$8)\ 111 \rightarrow 0$$

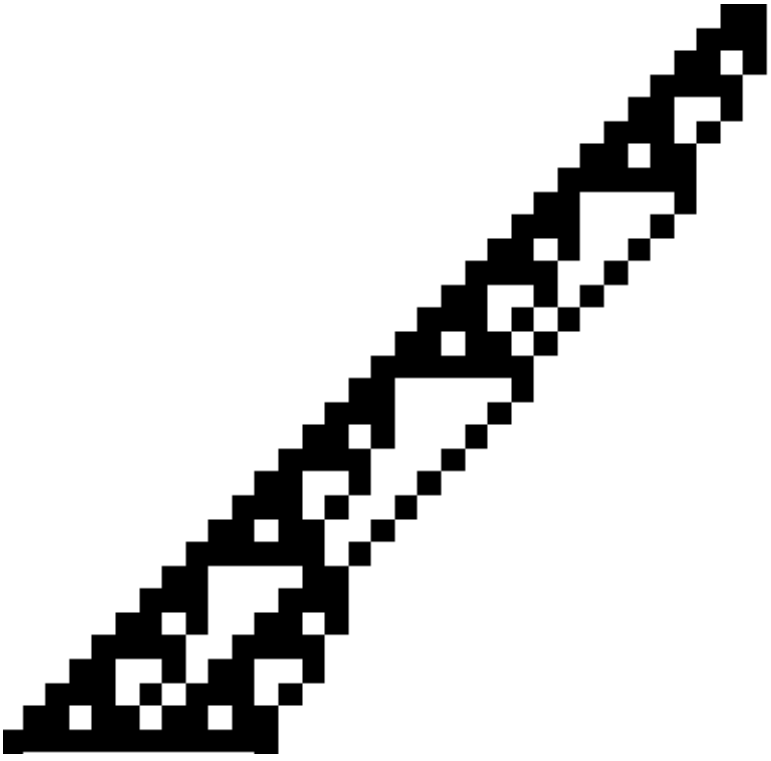

Рис. 4. 32 шага автомата 106

На рис. 4 показаны 32 шага автомата. Также видно, что, при высоте 32, его основание равно 12.

Рассмотрим автоматы данного типа с основанием $6 * 2^n$ и высотой $f(n) = 8 * 4^n$ и докажем главное утверждение.

Как автомат порождает фрактал:

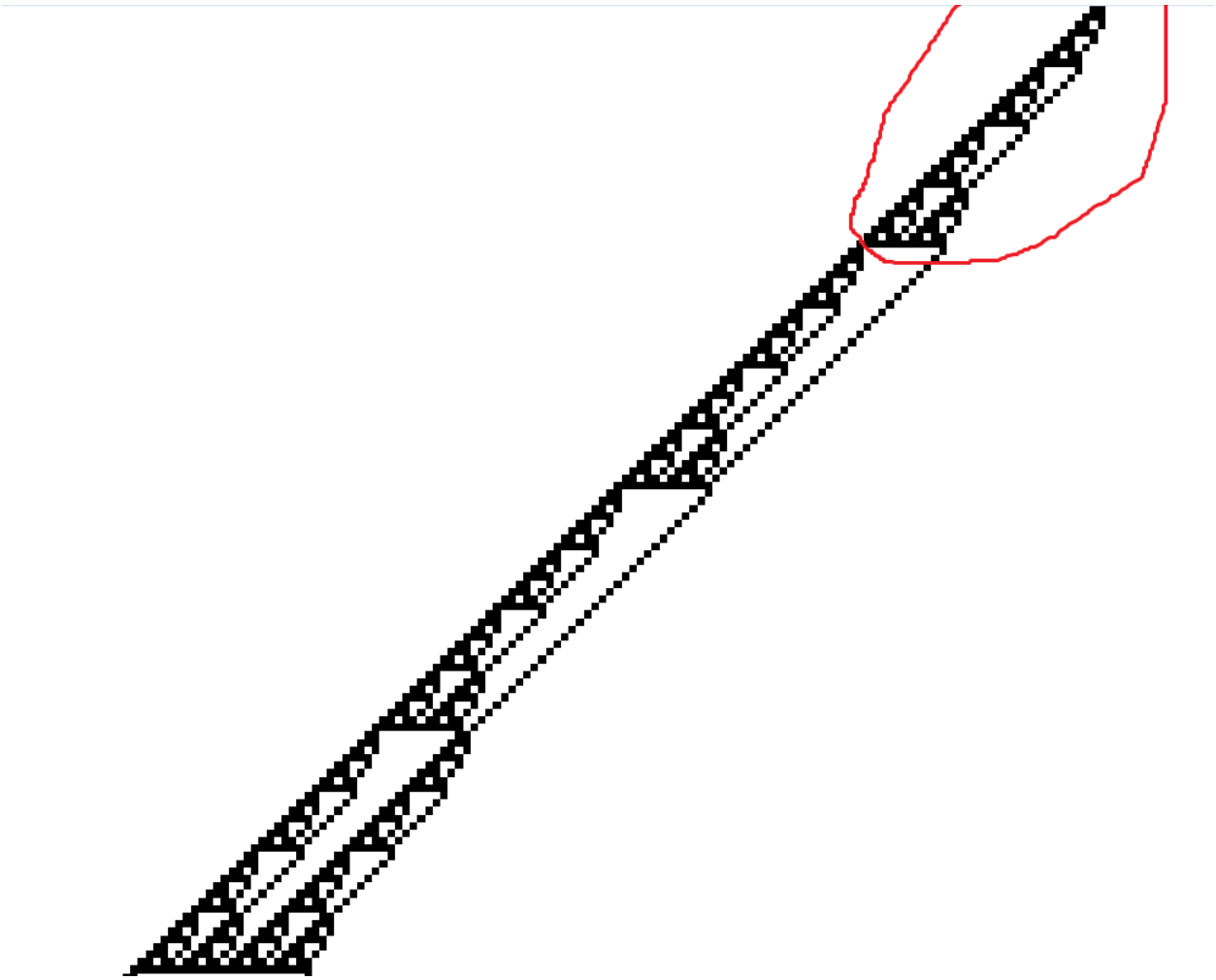

Рис. 5. Большее количество шагов автомата 106

Из рис. 5 видно, что автомат копирует выделенную часть 4 раза.

Лемма 2.3. При описанном переходе от шага $f(n)$ к шагу $f(n+1)$ основание узора будет самой широкой его частью.

Доказательство. Предположим, что для $f(n)$ оно таковым является (это видно из рисунка 5, для какого-то n). Теперь выводим две копии предыдущего узора рядом друг с другом. Так как основание предыдущей фигуры было самым

широким то явно, что основание умноженное на два так же будет самым широким основанием новой фигуры.

Замечание об основании: после $f(n)$ шагов, основание будет строкой 1^{6*2^n}.

Доказательство(по индукции по n).

Базис. $n = 1$, как видно из рисунка 4, для этого случая утверждение верно.

Индукционный шаг. предположим, что $n = k$ верно, докажем, что утверждение верно для $n = k + 1$:

Рассмотрим более подробно, как автомат копирует верхнюю часть узора 4 раза. Во-первых, обозначим позицию левой точки основания через «x», соответственно, если основание равно $6 * 2^k$, то позиция правой точки будет равна $x + 6 * 2^k - 1$.

Далее, рассмотрим правила «$001 \rightarrow 1$», «$011 \rightarrow 1$» и «$110 \rightarrow 1$». Первые 2 правила оставляют «11» под левой частью основания, начиная с позиции x-1, третье правило оставляет «1» под правой частью основания, начиная с позиции $x + 6 * 2^k - 1$.

Так как слева имеем «11», что и было входным словом автомата, то предыдущий узор начнет повторяться заново. Так как справа «1», то правило «$001 \rightarrow 1$», справа, будет выводить прямую линию. Через $8 * 4^k$ шага, левая часть узора, будет в точке $x - 8 * 4^k$, а правая в $x - 8 * 4^k + 6 * 2^k - 1$. Прямая линия, выводимая из правой части, после $8 * 4^k$ шагов, будет в точке $x + 6 * 2^k - 1 - 8 * 4^k + 1 = x + 6 * 2^k + 8 * 4^k$, что будет находиться точно справа, от правой части узора, выводимого из левой стороны. Столкновения между левой и правой частью не произойдет, потому что основание узора всегда шире всех остальных частей, по лемме 2.3, хотя бы на 1, поэтому оно максимум может приблизиться на 1 к

остальным частям узора, но так как есть правило «$101 \to 1$», то это ни на что не повлияет.

Теперь мы получили основание на 1 больше предыдущего. Из-за этого, после следующих $8 * 4^k$ шагов, мы получим основание «1…101», так как есть правила «$101 \to 1$» и «$110 \to 1$», то справа мы получим «11», вместо «1», и поэтому автомат параллельно и слева и справа начнет выводить одинаковые узоры. Они не столкнуться по тому же аргументу, как и в предыдущем абзаце. А так как левая часть будет выводиться из позиции $x - 2 * 8 * 4^k$, а правая из $x - 2 * 8 * 4^k + 6 * 2^k - 1$, то итоговое новое основание будет ограничено точками $x - 3 * 8 * 4^k$ и $x - 3 * 8 * 4^k + 6 * 2^k - 1 + 6 * 2^k$ и равно $6 * 2^{k+1}$, что соответствует высоте $8 * 4^{k+1}$. Конец.

Автомат 94, входное слово «111101111»

Автомат имеет следующие правила:

1) $000 \to 0$
2) $001 \to 1$
3) $010 \to 1$
4) $011 \to 1$
5) $100 \to 1$
6) $101 \to 0$
7) $110 \to 1$
8) $111 \to 0$

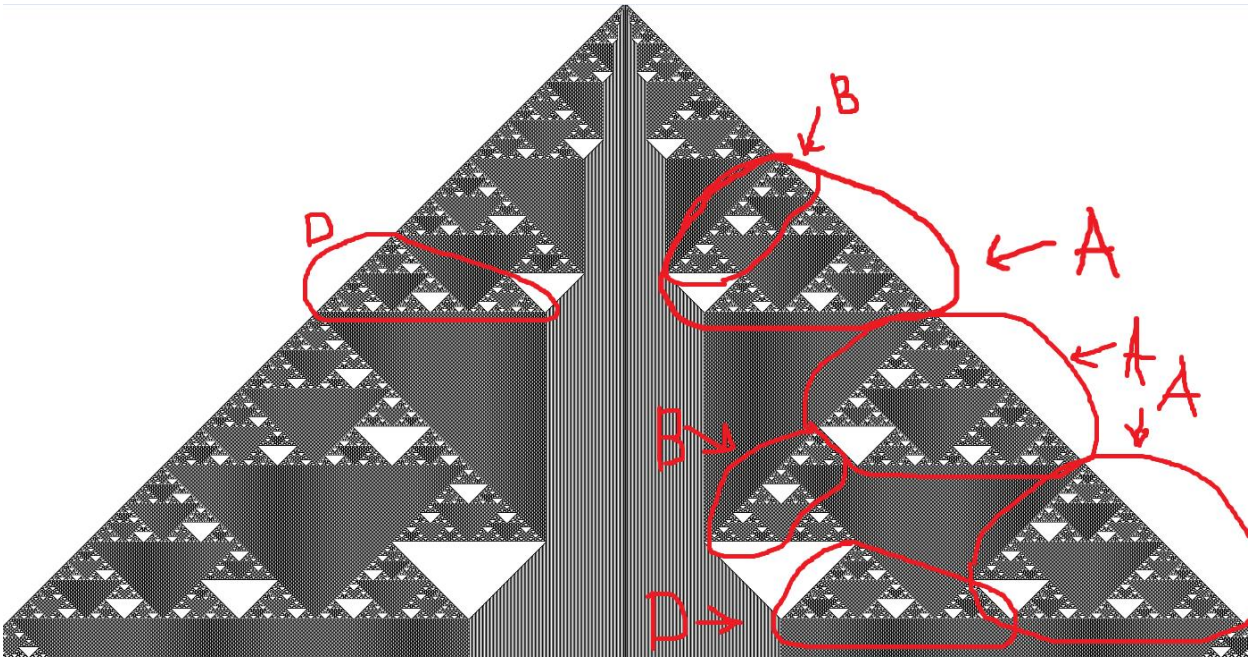

Рис. 6. Узор автомата 94

Обозначим через $f(n)$ функцию, вычисляемую по формуле $2^{n+3}+2$. Далее будем рассматривать содержимое ленты за $f(n)$ шагов, где $n \in \mathbb{N}$.

Так же введем несколько других функций:

$CTr_b(n) = 2^{n+1}+4 -$ длина основания углового треугольника (белый треугольник под частью "B", на картинке)

$CTr_s(n) = 2^n+1 -$ длина стороны углового треугольника

$MTr_h(n) = 2^{n+2} -$ высота основного треугольника

$MTr_b(n) = 2^n * 6 + 2 -$ ширина основного треугольника

$D_l(n) = 2^n * 3 - 4 -$ ширина части D.

Лемма 2.4. После шага $f(n+1)$ автомат скопирует часть «A» два раза.

Доказательство. Заметим, что конфигурации правой части ленты, после шага $f(n+1)$, и после шага $f(n)$ имеют вид $(111100)^*11$. Отсюда и следует, что сначала скопируется одна часть «A», а затем вторая.

Лемма 2.5. Последовательность вида $(111100)^*$ переходит в последовательность вида $(100111)^*$.

Доказательство. Очевидно из правил 2,4,5,7,8.

Лемма 2.6. После вывода основания углового треугольника части «A», через 2 шага, по левую сторону этого треугольника будет написано слово, вида $(111100)^*11$ (такое же, как и в начале части «A»), что указано стрелкой, на рисунке 6.

Доказательство. Заметим, что граница части «A» (и её продолжение) будет идти своеобразным зигзагом:

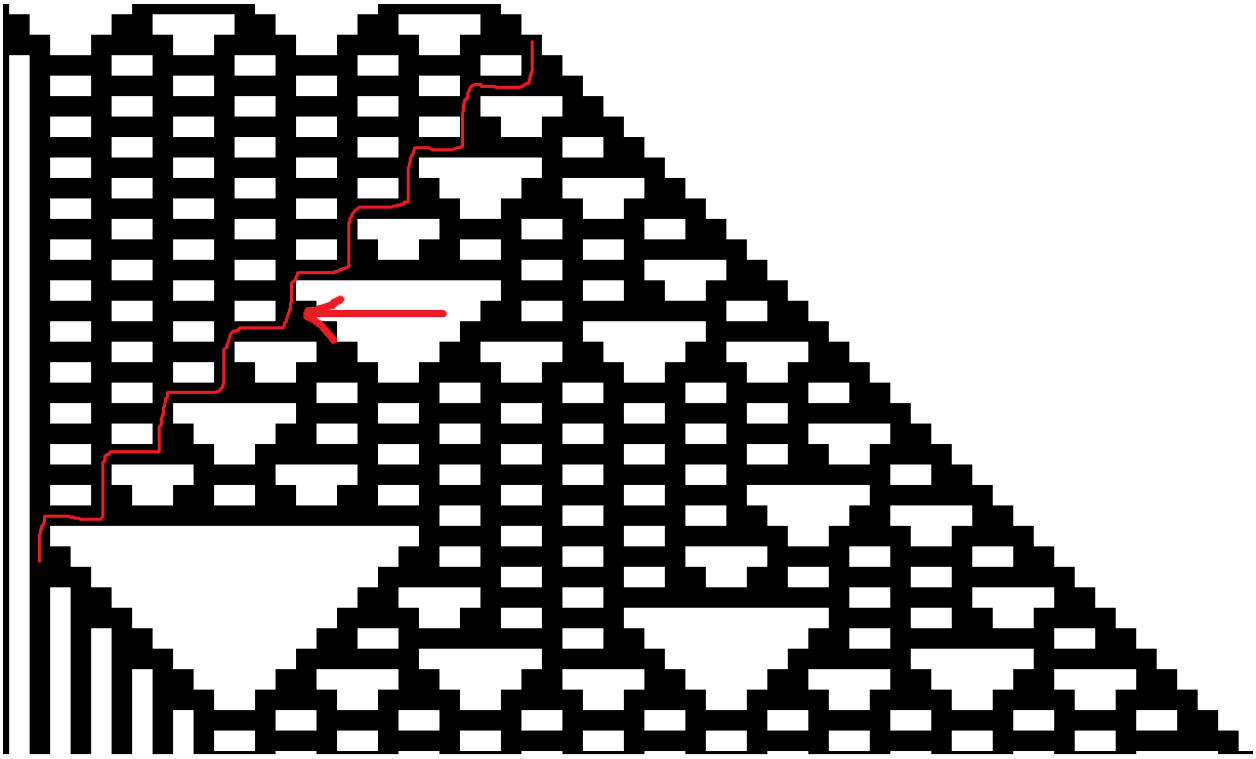

Рис. 7. Граница части «A» автомата 94

И так как левая часть будет идти вниз, а правая внутрь, то через 2 шага мы получим «11».

Лемма 2.7. От первой части «A», слева, будет отходить часть «B» (см. рисунок 7).

Доказательство. Так как левый катет углового треугольника части «A» будет равен $\frac{1}{4}$длины правой стороны части «A» $-$ 1, и конфигурации ленты в верхней четверти части «A» и по левую сторону углового треугольника скопированной части «A» имеют вид «$(111100)^{*}11$» (по лемме 2.7), то часть «B» будет повторяться заново, как и верхняя четверть части «A». Также, слева будет выведена в точности часть «B», потому что она устроена следующим образом:

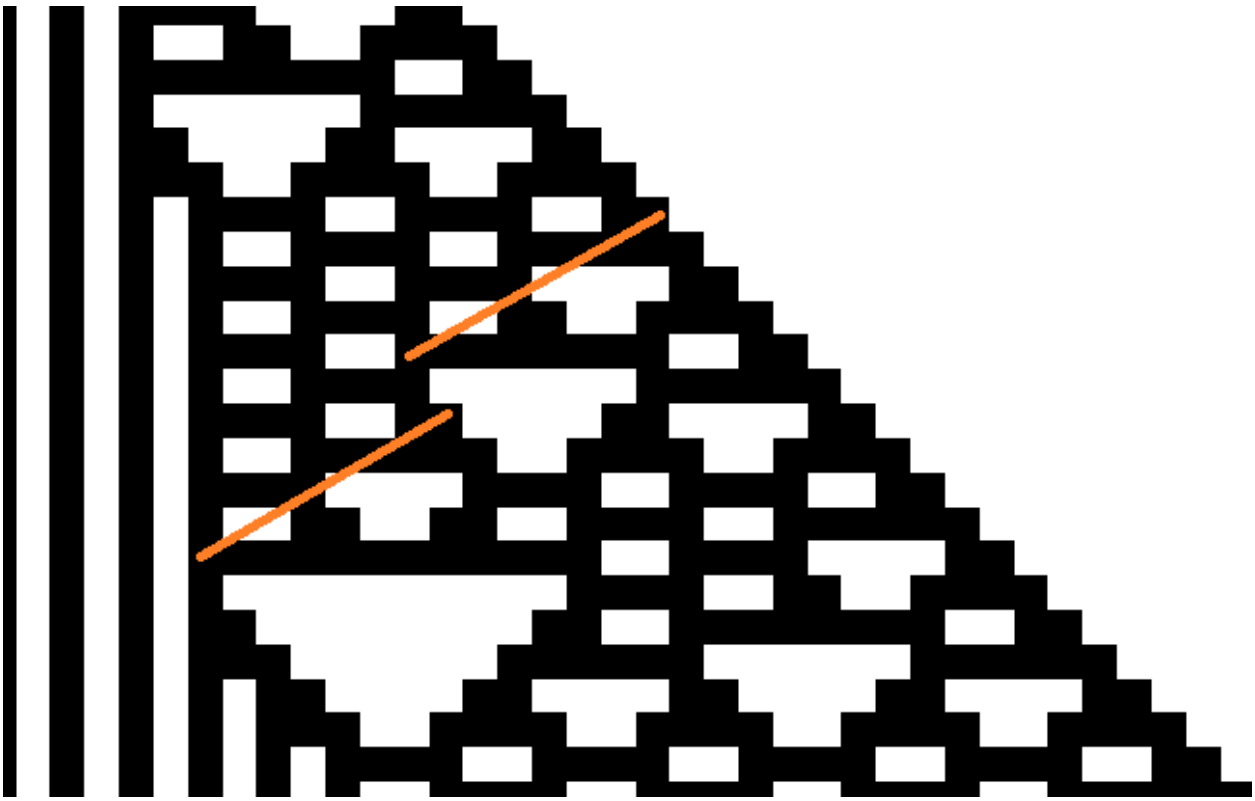

Рис. 8. Часть «B» автомата 94

Как видно из рисунка 8, линии, измеряющие длину одинакового размера.

Лемма 2.8. Вторая часть «A», и новый угловой треугольник соединены частью «D».

Доказательство. Очевидно из рисунка 8. Также заметим, что длина основания части «D» равна $D_l(n) = 2^n * 3 - 4$ и в основном доказательстве проверим, что эта формула сохраняется на следующем шаге.

Лемма 2.9. После некоторого количества шагов, слева, рядом друг с другом, будут выведены 2 треугольника, основание которых будет равно $CTr_b(n-1) = 2^n + 4$. Так как их два, и у них есть 4 общих клетки, то суммарно их основание будет равно $2^n + 4 + 2^n + 4 - 4 = 2^{n+1} + 4$. Из-за того, что эти два треугольника копии предыдущего треугольника, на этом же месте, то мы получим новое основание для углового треугольника.

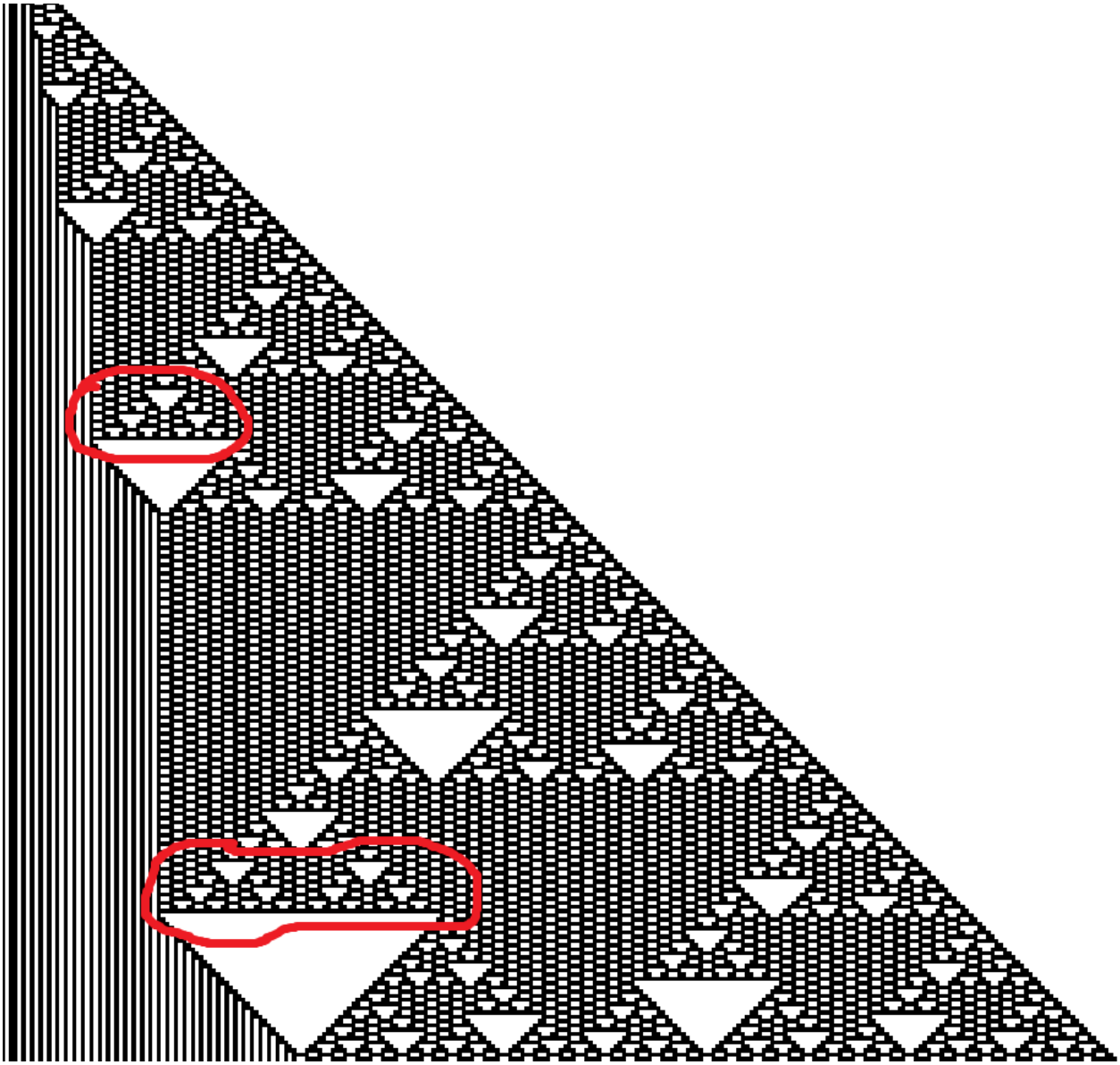

Рис. 9. Доказательство леммы 2.9

Доказательство. Очевидно из рисунка 9.

Доказательство главного утверждения (по индукции по n).

Базис. $n = 1$, как видно из рисунка 6, для этого случая утверждение верно.

Индукционный шаг. Предположим, что $n = k$ верно, докажем, что утверждение верно для $n = k + 1$:

В предыдущих леммах было сказано, какие части предыдущего узора копируются в новый. Тут будет доказано, почему все формулы истинны при n = k + 1.

Рассмотрим формулы CTr_b и CTr_h. То, что они истинны, показано в лемме 2.9.

MTr_h остаётся истинной, потому что мы скопировали 2 части «A».

Далее докажем истинность формулы для D_l.

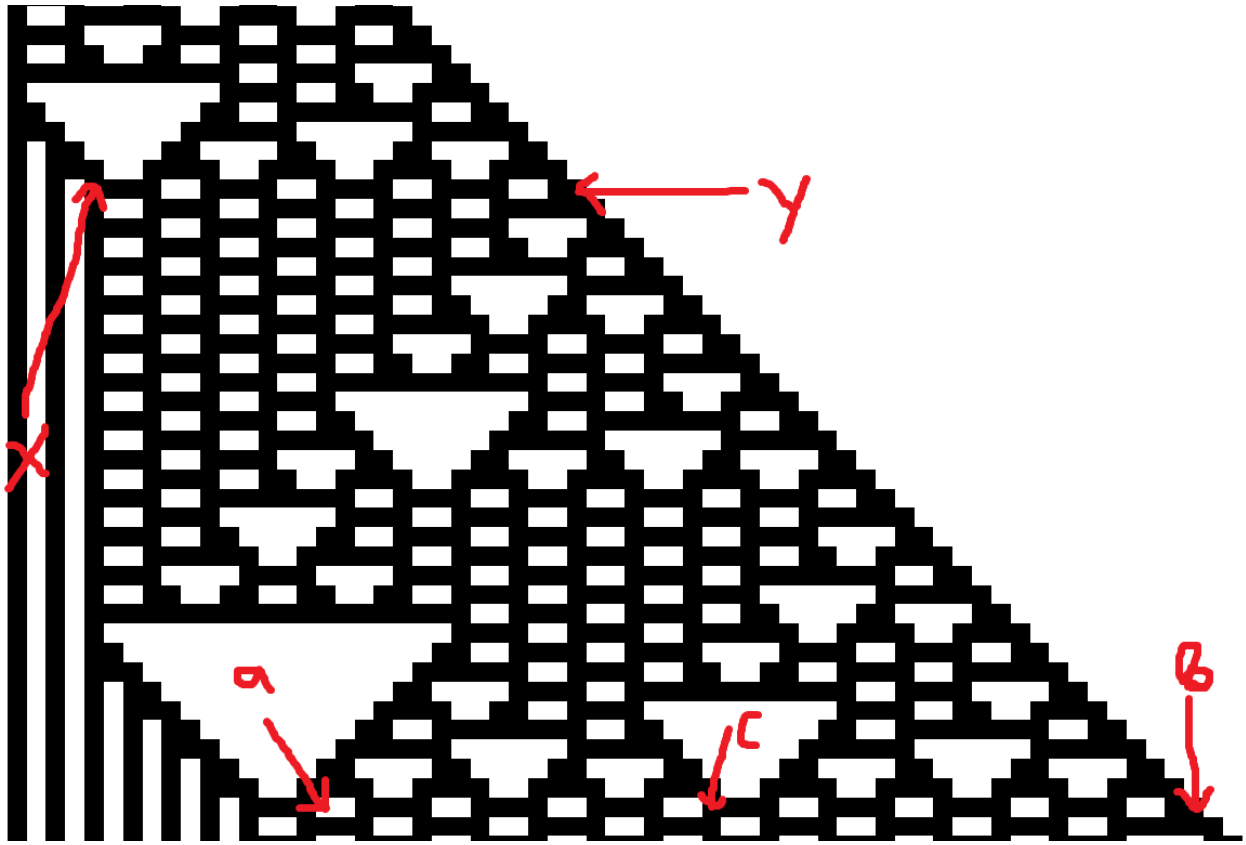

Рис. 10. Представление точек x, y, a, b, c

Зафиксируем точки x и y, где x – левая часть старого фрактала, а y – правая. Причем ясно, что $y = x + Mr_b(k)$.

Теперь зафиксируем точки a, b, c,где $a-$ правый конец нижней части нового углового треугольника + 1, $b-$ правый конец нового фрактала, а c −левый конец углового треугольника второй копии части «A».

Эти точки можно вычислить по формулам:

$$a = x + CTr_s(k+1) + 3$$
$$b = y + 2 * MTr_b(k)$$
$$c = b - MTr_b(k) + 1$$

Из рисунка 10 видно, что $D_l(k+1) = c - a$. Докажем это:

$c - a = b - MTr_b(k) + 1 - x - CTr_s(k+1) - 3 = y + 2 * MTr_h(k) - MTr_b(k) + 1 - x - CTr_s(k+1) - 3 = x + MTr_b(k) - 1 + 2 * MTr_h(k) - MTr_b(k) + 1 - x - CTr_s(k+1) - 3 = 2 * MTr_h(k) - CTr_s(k+1) - 3 = 2 * 2^{k+2} - 2^{k+1} - 1 - 3 = 4 * 2^{k+1} - 2^{k+1} - 1 - 3 = 3 * 2^{k+1} - 4.$

Отсюда следует, что мы вывели новую часть «D».

Далее, если сложить $3 * 2^{k+1} - 4 + 4 + MTr_b(k) = 3 * 2^{k+1} + 2^k * 6 + 2 = 6 * 2^{k+1} + 2$. То получаем новую длину основания основного треугольника, что соответствует формуле MTr_b. Конец.

2.3. Доказательства универсальности некоторых клеточных автоматов

Далее будет рассмотрено сведение универсальной машины Тьюринга $U_{m,n}$, где m−количество состояний, а n−количество символов, к соответствующим n+n*m клеточным автоматам.

В статье «Four small universal Turing machines» авторов Woods и Neary доказана универсальность машин Тьюринга для состояние-символ пар (5,5), (6,4), (9,3) и (15,2)[2]. Возьмем, например, пару (9,3) и покажем что ее можно промоделировать с помощью 30-символьного клеточного автомата. Остальные пары можно моделировать абсолютно аналогичным образом.

Теорема 2.3. 30-символьный клеточный автомат может моделировать универсальную машину Тьюринга $U_{9,3}$.

Доказательство.

Рассмотрим таблицу переходов для универсальной машины Тьюринга $U_{9,3}$.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
c	bRu_1	cLu_3	cLu_3	bLu_9	cRu_6	bLu_4	δLu_4	cRu_7	bLu_5
b	cLu_2	cLu_2	bLu_4	bLu_4		bRu_6	bRu_7	cRu_9	cRu_8
δ	δRu_3	δLu_2	δRu_1	δLu_4	δLu_8	δRu_6	δRu_7	δRu_8	cRu_1

Рис. 11. Таблица переходов для $U_{9,3}$

$U_{9,3}$ начинает работу из состояния u_1 и заканчивает ее, читая символ «b», находясь в состоянии u_5. Входные слова также описаны в статье вышеуказанных авторов, поэтому о них не стоит задумываться и требуется лишь запрограммировать переходы из состояния в состояние по символу.

Обозначим алфавит нашего автомата следующим образом: $A = \{c, b, \delta\} \cup \{\frac{y}{z} | y \in \{c, b, \delta\}, z \in \{u_1, \dots, u_9\}\}$. Таким образом, всего получается 3 + 3 * 9 = 30 символов.

С помощью символов вида $\frac{y}{z}$ будем обозначать позицию головки автомата. Заметим, что при каждом такте работы клеточного автомата на ленте может быть написан один и только один символ вида $\frac{y}{z}$.

Теперь рассмотрим правила машины Тьюринга, и как они транслируются в правила клеточного автомата. Всего может быть 3 следующих вида правил:

1) $q, x \to p, y, R$

2) $q, x \to p, y, L$

3) $q, x \to p, y, S$

Первый вид говорит: «Читай символ «x» в состоянии q, печатай там «y» и двигайся вправо». Остальные работают по аналогии, только второй двигается влево, а третий стоит на месте.

Первое правило.

Чтобы его промоделировать нам потребуются правила (клеточного автомата) вида:

$\alpha \frac{x}{q} \beta \to y$ (печатаем «y» вместо «x»)

$\frac{x}{q} \alpha\beta \to \frac{\alpha}{p}$ (двигаемся вправо)

Второе правило.

Его моделируют правила вида:

$\alpha \frac{x}{q} \beta \to y$ (печатаем «y»)

$\alpha\beta \frac{x}{q} \to \frac{\beta}{p}$ (двигаемся влево)

Третье правило. (необязательное для $U_{9,3}$)

Его моделирует правило вида:

$\alpha \frac{x}{q} \beta \to \frac{y}{p}$ (печатаем новый символ и остаёмся на месте)

Остальные правила будут иметь следующий вид:

$xyz \to y \; \forall x, y, z \in \{c, b, \delta\}$.

Стоит заметить, что из состояния u_5, по символу «b» автомат никуда не переходит и ничего не печатает, рассмотрим в следующей теореме как запрограммировать этот случай, чтобы клеточный автомат останавливался тогда и только тогда, когда-бы остановилась $U_{9,3}$.

Теорема 2.4. Клеточный автомат остановится тогда и только тогда, когда-бы остановилась $U_{9,3}$.

Доказательство.

Вспомним определение остановки для клеточных автоматов. Оно звучит следующим образом: «Пусть $f_i -$ это конфигурация ленты автомата после i шагов. Тогда будем говорить, что автомат остановился в конфигурации f_i, если $(\forall j \in \mathbb{N}) j > i \to (\forall k \in \mathbb{N}) f_i(k) = f_j(k)$».

Так как состояние u_5 по символу «b» никуда не переходит, то введем следующее правило для этого случая:

$$\alpha \frac{b}{u_5} \beta \to b.$$

Далее, очевидно, что мы избавились от символов вида $\frac{y}{z}$, а так правила вида $xyz \to y$ их не выводят, то дальше, очевидно, что автомат остановился по определению остановки.

В заключение, ясно, что проблема остановки является неразрешимой для $\min(5 + 5 * 5{,}4 + 6 * 4{,}3 + 9 * 3{,}2 + 15 * 2) = 28$-символьного клеточного автомата.

Теперь рассмотрим несколько другой способ моделирования машины Тьюринга на клеточном автомате. Пусть множество состояний МТ это U, алфавит это A, а R – это множество правил. Тогда алфавитом ленты КА будет следующее множество:

$$A \cup \{p_{u_i} | u_i \in U\} \cup \{p' \mid \exists (u_j, x \to p, a, L) \in R\}$$

Докажем следующую теорему:

Теорема 2.5. Проблема остановки разрешима для 15-символьных клеточных автоматов.

Доказательство.

Пусть на ленте при каждом такте работы имеется только один символ p или p'. Символ p будет моделировать головку автомата и обозреваемый ею символ будет находиться справа от него.

Рассмотрим как моделировать сдвиг вправо. Пусть на ленте автомата написана строка $s =$ " $\dots xypzk \dots$", состояние автомата “p” обозреваемый символ “z” и в машине Тьюринга имеется правило $p, z \to q, a, R$. Тогда его будут моделировать следующие правила автомата:

$$pz\beta \to q, \forall \beta \in A$$

$$\beta pz \to a, \forall \beta \in A$$

Отсюда очевидно, что s превратится в " $\dots xyaqk \dots$".

Теперь рассмотрим как моделировать левое правило. Пусть на ленте та же строка $s =$ " $\dots xypzk \dots$". И в МТ имеется правило $p, z \to q, a, L$. Тогда мы сделаем сдвиг влево в 2 такта.

Первый такт будут моделировать следующие правила:

$$pz\alpha \to a, \forall \alpha \in A$$

$$\alpha pz \to q', \forall \alpha \in A$$

А второй так следующие:

$$\alpha\beta q' \to q, \forall \alpha, \beta \in A$$

$$\alpha q'\beta \to \alpha, \forall \alpha, \beta \in A$$

Отсюда очевидно, что после этих двух тактов на ленте будет $s =$ " $\ldots xqyak \ldots$ ".

С таким способом моделирования минимальное количество символов даёт универсальная машина Тьюринга $U_{6,4}$ показанная на рис. 12.

	u_1	u_2	u_3	u_4	u_5	u_6
g	bLu_1	gRu_1	bLu_3	bRu_2	bLu_6	bLu_4
b	gLu_1	gRu_2	bLu_5	gRu_4	gRu_6	gRu_5
δ	cRu_2	cRu_2	δLu_5	cRu_4	δRu_5	gRu_1
c	δLu_1	gRu_5	δLu_3	cRu_5	bLu_3	

Рис. 12. Таблица переходов для $U_{6,4}$

На рис. 12 видно, что нам потребуется 4 символа из основного алфавита, плюс 6 символов из алфавита состояний, плюс $|\{u_1, u_3, u_4, u_5, u_6\}| = 5$, так как только для символов состояния u_1, u_3, u_4, u_5, u_6 существует какое-то правило вида $*,* \rightarrow u_i,*, L$, итого 15 символов.

3. Программная реализация клеточных автоматов и использованные для неё технологии

3.1. Использованные технологии

В данной работе была написана программа для генерации картинок двух и трехсимвольных клеточных автоматов с использованием технологий Oracle Java и Netbeans, была выбрана именно эта технология, потому что она является кроссплатформенной.

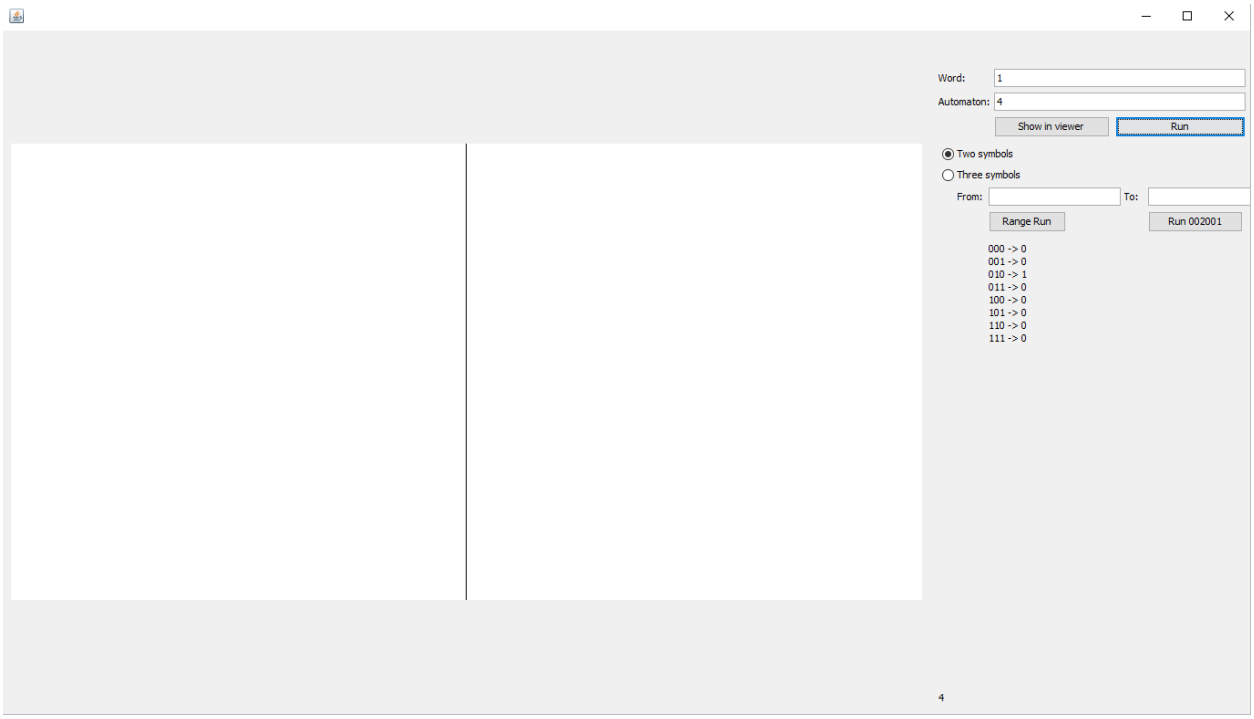

Рис. 13. Двухсимвольный автомат «4», слово «1»

На рис. 13 приведен пример картинки, генерируемой двухсимвольным клеточным автоматом «4». Заметим, что по определению остановки клеточного автомата, автомат «4» остановился, потому что единственное правило, которое выводит «1» – «$010 \rightarrow 1$». На данной картинке используются черные пиксели, для символа «1», и белые для символа «0». Более подробное объяснение того, как работают правила будет дано в обзоре проблемы остановки для двух и трех символьных клеточных автоматов.

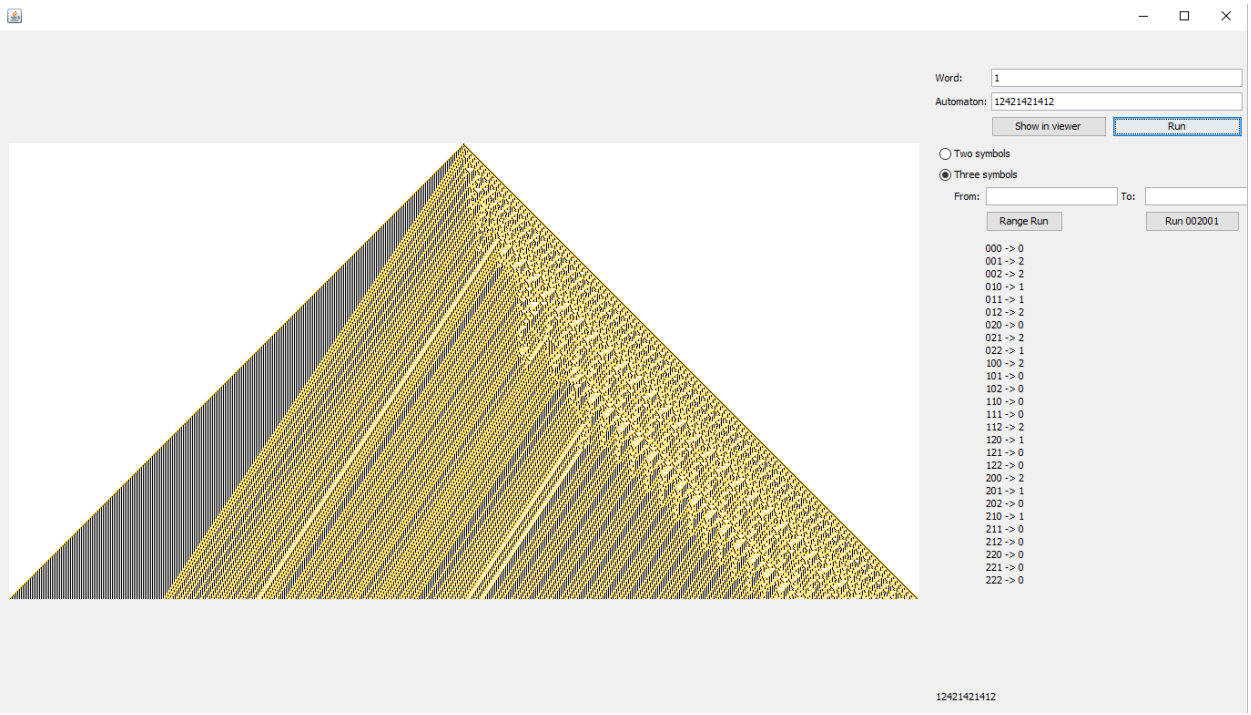

Рис. 14. Трехсимвольный автомат 12421421412, входное слово «1»

На рис. 14 приведен пример картинки, генерируемой трех символьным клеточным автоматом. Картинки являются цветными, поэтому этот документ лучше рассматривать в цветном формате, а не в черно-белом.

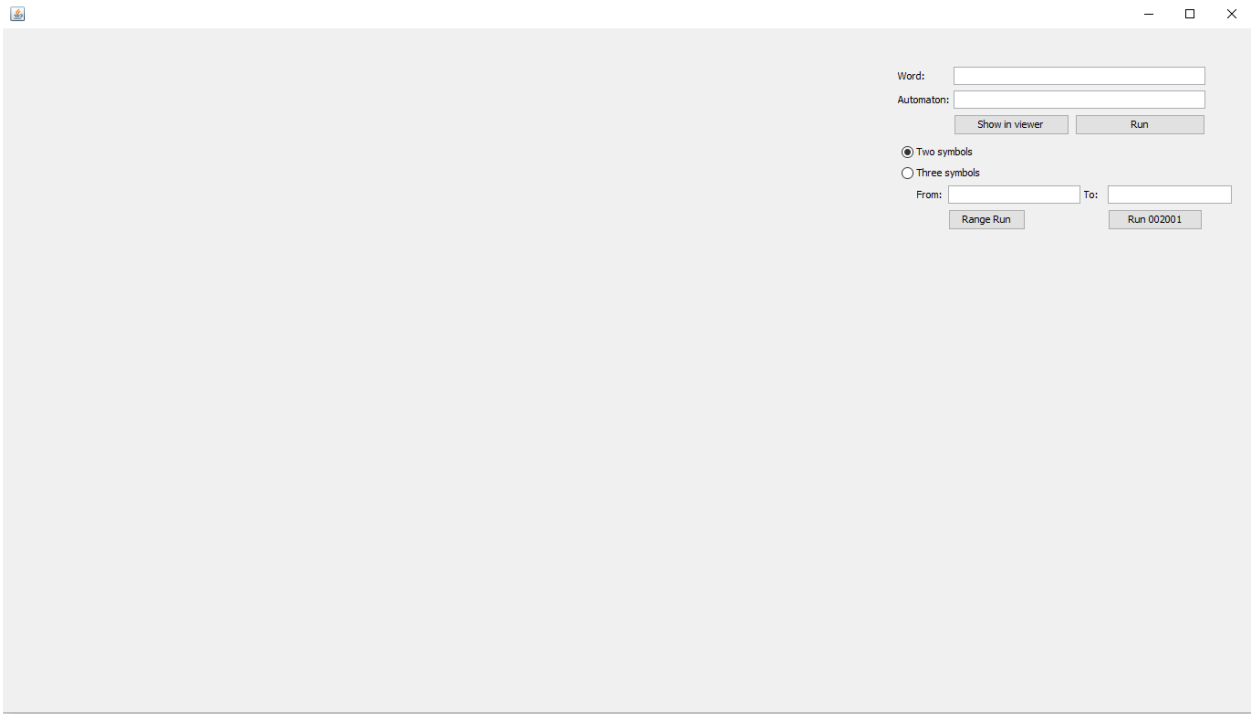

Рис. 15. Интерфейс программы

3.2. Программная реализация

Программа может порождать узоры любых двух и трехсимвольных автоматов. В программе есть поле «Word» (входное слово), «Automaton»(номер автомата), в которые записываются входные данные.

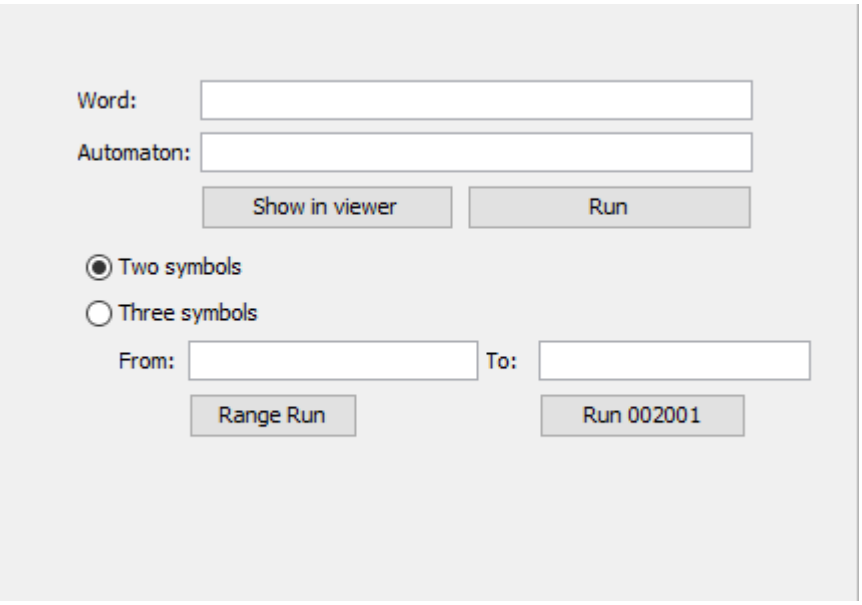

Рис. 16. Поля и кнопки программы

Как видно из рисунка 16, перед вводом данных мы также должны выбрать тип автомата трехсимвольный он или двухсимвольный.

Также присутствуют кнопки для «слайд-шоу», в поля «From» и «To» мы вводим от какого до какого автомата мы ходим просмотреть узоры. Кнопка «Range Run» выводит автоматы последовательно. А кнопка «Run 002001» выводит только те автоматы, в которых есть правила «$002 \to 0$» и «$001 \to 2$», или «$001 \to 0$» и «$002 \to 1$», а также им симметричные. Конечно же для кнопок со «слайд-шоу» нужно в поле «Word» ввести входное слово.

Программу можно скачать по ссылке https://github.com/paulpaul1076/CellularAutomataViewer и собрать в среде Netbeans.

Заключение

В рамках данной работы исследовались алгоритмические проблемы для клеточных автоматов.

Сначала были приведены все термины, использующиеся в данной работе (см. главу 1). Затем были рассмотрены наиболее частые способы доказательства неразрешимости какой-то проблемы для определенного вычислительного устройства (см. главу 2), и представлены доказательства разрешимости проблемы остановки для двухсимвольных клеточных автоматов, и для некоторых трехсимвольных. Также были исследованы регулярные структуры, порождаемые двухсимвольными клеточными автоматами и приведены доказательства их регулярности (см главу 2).

Далее был представлен наиболее важный результат о том, что проблема остановки неразрешима для пятнадцатисимвольных клеточных автоматов (см. главу 2, пункт 3). Мой научный руководитель рассказал мне хороший способ моделирования машины Тьюринга с помощью клеточных автоматов, представленный в разделе 2.3, и используя результат из статьи «Four small universal Turing machines»[2], я показал, что проблема остановки неразрешима для пятнадцатисимвольных клеточных автоматов.

В конце работы представлена программа, которая использовалась для изучения структур, порождаемых автоматами, и описаны технологии, которые использовались для написания данной программы (см. главу 3).

В качестве дальнейших исследований, планируется более подробно ознакомиться с м-теговыми системами (англ. m-tag system), и попробовать промоделировать их на трехсимвольном клеточном автомате. Открытым вопросом остаётся разрешимость проблемы остановки для 3, 4, …, 14-символьных автоматов.

Список литературы

1. Wolfram, Stephen. A New Kind of Science**.** −Wolfram Media, 2002.−1197 с.
2. Turlough Neary and Damien Woods. Four small universal Turing machines.[Электронный ресурс].−149 кб. Режим доступа: http://www.dna.caltech.edu/~woods/download/NearyWoodsMCU07.pdf
3. Matthew Cook. Universality in Elementary Cellular Automata. [Электронный ресурс].−916 кб. Режим доступа: http://www.complex-systems.com/pdf/15-1-1.pdf
4. Ilachinski, Andrew. Cellular Automata: A Discrete Universe. –World Scientific, 2001.-808 с.
5. Alex Smith. Universality of Wolfram's 2,3 Turing Machine.[Электронный ресурс].-741 кб. Режим доступа: https://www.wolframscience.com/prizes/tm23/TM23Proof.pdf
6. Michael Sipser. Introduction to the Theory of Computation (Third ed.).−PWS Publishing, 2012.−480 с.
7. Adamatzky, Andrew. Game of Life Cellular Automata.−Springer, 2010.−579 с.
8. Chopard, Bastien; Droz, Michel. Cellular Automata Modeling of Physical Systems.−Cambridge University Press, 2005.−356 с.
9. Smith, Alvy Ray. Introduction to and Survey of Cellular Automata or Polyautomata Theory.[Электронный ресурс].−107 кб. Режим доступа: http://alvyray.com/papers/CA/PolyCA76.pdf
10. Вопрос «Is the halting problem decidable for 3 symbol one dimensional cellular automata?».−Режим доступа: http://cs.stackexchange.com/questions/52655/is-the-halting-problem-decidable-for-3-symbol-one-dimensional-cellular-automata
11. Вопрос «Разрешима ли проблема остановки для 3х символьных одномерных клеточных автоматов?».−Режим доступа:

http://math.hashcode.ru/questions/88548/высшая-математика-разрешима-ли-проблема-остановки-для-3х-символьных-клеточных-автоматов

Приложение
Программный код

```
package caviewer;


import java.awt.Color;

import java.awt.Desktop;

import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.SwingWorker;


/**

 *

 * @author Paul

 */

public class Form extends javax.swing.JFrame {


   /**

    * Creates new form Form

    */

   public Form() {

      initComponents();

      this.jRadioButton1.setSelected(true);

   }


   /**

    * This method is called from within the constructor to initialize the form.

    * WARNING: Do NOT modify this code. The content of this method is always

    * regenerated by the Form Editor.
```

```
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
private void initComponents() {

    buttonGroup1 = new javax.swing.ButtonGroup();
    jButton1 = new javax.swing.JButton();
    jTextField1 = new javax.swing.JTextField();
    jTextField2 = new javax.swing.JTextField();
    jLabel1 = new javax.swing.JLabel();
    jButton2 = new javax.swing.JButton();
    jRadioButton1 = new javax.swing.JRadioButton();
    jRadioButton2 = new javax.swing.JRadioButton();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jTextField3 = new javax.swing.JTextField();
    jTextField4 = new javax.swing.JTextField();
    jLabel5 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    jButton3 = new javax.swing.JButton();
    jLabel7 = new javax.swing.JLabel();
    jButton4 = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jButton1.setText("Run");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
```

```
    }
});

jButton2.setText("Show in viewer");
jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
    }
});

buttonGroup1.add(jRadioButton1);
jRadioButton1.setText("Two symbols");

buttonGroup1.add(jRadioButton2);
jRadioButton2.setSelected(true);
jRadioButton2.setText("Three symbols");

jLabel3.setText("Word:");

jLabel4.setText("Automaton:");

jLabel5.setText("To:");

jLabel6.setText("From:");

jButton3.setText("Range Run");
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton3ActionPerformed(evt);
    }
```

```
        });

        jButton4.setText("Run 002001");
        jButton4.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton4ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE, 953, Short.MAX_VALUE)
                .addGap(18, 18, 18)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jRadioButton1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addGroup(layout.createSequentialGroup()
                                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addGroup(layout.createSequentialGroup()
                                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                            .addComponent(jLabel4)
                                            .addComponent(jLabel3))
                                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
```

```
                        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                            .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 127,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jButton1, javax.swing.GroupLayout.DEFAULT_SIZE, 143,
Short.MAX_VALUE))
                        .addComponent(jTextField1, javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jTextField2, javax.swing.GroupLayout.Alignment.LEADING)))
                    .addComponent(jLabel2)
                    .addComponent(jRadioButton2))
                .addGap(0, 0, Short.MAX_VALUE)))
            .addGap(31, 31, 31))
        .addGroup(layout.createSequentialGroup()
            .addGap(21, 21, 21)
            .addComponent(jLabel6)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE, 145,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jLabel5))
                        .addComponent(jButton3))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE, 136,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jButton4, javax.swing.GroupLayout.PREFERRED_SIZE, 104,
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                    .addGap(0, 23, Short.MAX_VALUE))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jLabel7)
                    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(31, 31, 31)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel3))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel4))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jButton1)
                        .addComponent(jButton2))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(jRadioButton1)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jRadioButton2)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
                    .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel5)
                    .addComponent(jLabel6))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jButton3)
                    .addComponent(jButton4))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jLabel7)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 506,
Short.MAX_VALUE)
                .addComponent(jLabel2))
            .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addContainerGap())
    );

    pack();
  }// </editor-fold>//GEN-END:initComponents

  private boolean checkTwoSymbolWord(String word) {
    for (int i = 0; i < word.length(); i++) {
      if (word.charAt(i) != '0' && word.charAt(i) != '1') {
        return false;
      }
    }
    return true;
  }
```

```
private boolean checkTwoSymbolRule(long ruleNumber) {
    final long maxNumOfTwoSymbolAutomata = 256; // 2 ^ 8
    if (!(ruleNumber >= 0 && ruleNumber < maxNumOfTwoSymbolAutomata)) {
        return false;
    }
    return true;
}

private boolean checkThreeSymbolWord(String word) {
    for (int i = 0; i < word.length(); i++) {
        if (word.charAt(i) != '0' && word.charAt(i) != '1' && word.charAt(i) != '2') {
            return false;
        }
    }
    return true;
}

private boolean checkThreeSymbolRule(long automatonNumber) {
    final long maxNumOfThreeSymbolAutomata = 7_625_597_484_987L; // 3 ^ 27
    if (!(automatonNumber >= 0 && automatonNumber < maxNumOfThreeSymbolAutomata)) {
        return false;
    }
    return true;
}

private void showPic(long automatonNumber) {
    this.jLabel2.setText("");

    final int GRID_WIDTH = 2000;
```

```
final int GRID_HEIGHT = 500;

String word = this.jTextField1.getText();

if (word.length() > GRID_WIDTH) {
   this.jLabel2.setText("Error: input word too long");
   return;
}

int base = 0;

byte[][] grid = new byte[GRID_HEIGHT][GRID_WIDTH];
BufferedImage image = new BufferedImage(GRID_WIDTH / 2, GRID_HEIGHT,
BufferedImage.TYPE_INT_ARGB);
CA automaton = null;

if (this.jRadioButton1.isSelected() && checkTwoSymbolWord(word) &&
checkTwoSymbolRule(automatonNumber)) {
   automaton = new TwoSymbolAutomaton(automatonNumber);
   base = 2;
} else if (this.jRadioButton2.isSelected() && checkThreeSymbolWord(word) &&
checkThreeSymbolRule(automatonNumber)) {
   automaton = new ThreeSymbolAutomaton(automatonNumber);
   base = 3;
} else {
   this.jLabel2.setText("Error: bad word/automaton number or no radio button enabled");
   return;
}

// rule label
StringBuilder sb = new StringBuilder();
```

```
for (int i = 0; i < base * base * base; ++i) {
   String lhs = "";
   int lhsInt = i;
   lhs += (lhsInt % base);
   lhsInt /= base;
   lhs = "" + (lhsInt % base) + lhs;
   lhsInt /= base;
   lhs = "" + (lhsInt % base) + lhs;
   lhs += " -> ";
   lhs += automaton.getRhs(i);
   sb.append(lhs).append("<br>");
}

this.jLabel7.setText("<html>" + sb.toString() + "</html>");

int offset = GRID_WIDTH / 2 - word.length() / 2;

for (char ch : word.toCharArray()) {
   grid[0][offset++] = (byte) (ch - '0');
}

for (int i = 1; i < GRID_HEIGHT; i++) {
   for (int j = 0; j < GRID_WIDTH; j++) {
      int left = 0, middle = 0, right = 0;
      if (j > 0) {
         left = grid[i - 1][j - 1];
      }
      middle = grid[i - 1][j];
      if (j < GRID_WIDTH - 1) {
         right = grid[i - 1][j + 1];
```

```
        }
        int rule = left * base * base + middle * base + right;
        grid[i][j] = automaton.getRhs(rule);
    }
}
offset = GRID_WIDTH / 4 + 1;
for (int i = 0; i < image.getHeight(); i++) {
    for (int j = 0; j < image.getWidth(); j++) {
        assert grid[i][j + offset] == 0 || grid[i][j + offset] == 1 || grid[i][j + offset] == 2;
        switch (grid[i][j + offset]) {
            case 0:
                image.setRGB(j, i, Color.WHITE.getRGB());
                break;
            case 1:
                image.setRGB(j, i, Color.BLACK.getRGB());
                break;
            case 2:
                image.setRGB(j, i, Color.ORANGE.getRGB());
                break;
        }
    }
}

// update image
this.jLabel1.setIcon(new ImageIcon(image));
this.jLabel1.repaint();
this.jLabel1.revalidate();

// save image
try {
```

```
            File f = new File(System.getProperty("user.dir") + "\\" + picName);
            ImageIO.write(image, "png", f);
        } catch (Exception e) {
            this.jLabel2.setText("Error: pic wasn't saved");
        }
        this.jLabel2.setText(Long.toString(automatonNumber));
    }

    // RUN BUTTON
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
        try {
            long automatonNumber = Long.parseLong(this.jTextField2.getText());
            showPic(automatonNumber);
        } catch (Exception e) {
            this.jLabel2.setText("Error: couldn't parse automaton number");
        }
    }//GEN-LAST:event_jButton1ActionPerformed

    // SHOW IN VIEWER BUTTON
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton2ActionPerformed
        try {
            File f = new File(System.getProperty("user.dir") + "\\" + picName);
            Desktop dt = Desktop.getDesktop();
            dt.open(f);
        } catch (Exception e) {
            this.jLabel2.setText("Error: couldn't open image");
        }
    }//GEN-LAST:event_jButton2ActionPerformed
```

```
    private RangeRunWorker rangeRun = null;

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton3ActionPerformed

        try {

            if (this.jButton3.getText().equals("Range Run")) {

                this.jButton3.setText("Stop");

                long from = Long.parseLong(this.jTextField3.getText());

                long to = Long.parseLong(this.jTextField4.getText());

                rangeRun = new RangeRunWorker(from, to);

                rangeRun.execute();

            } else if (this.jButton3.getText().equals("Stop")) {

                this.jButton3.setText("Range Run");

                rangeRun.cancel(true);

                rangeRun = null;

            }

        } catch (Exception e) {

            this.jLabel2.setText("Error: bad range input");

        }

    }//GEN-LAST:event_jButton3ActionPerformed


    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton4ActionPerformed


    }//GEN-LAST:event_jButton4ActionPerformed


    private final String picName = "temppic.png";


    class RangeRunWorker extends SwingWorker<Void, Void> {


        private final long from, to;
```

```
    public RangeRunWorker(long from, long to) {
        this.from = from;
        this.to = to;
    }

    @Override
    protected Void doInBackground() throws Exception {
        for (long i = from; i <= to; i++) {
            showPic(i);
            Thread.sleep(1000);
        }
        return null;
    }

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Windows".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
```

```
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    }
    //</editor-fold>
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Form().setVisible(true);
        }
    });
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
private javax.swing.JButton jButton4;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
```

```
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JRadioButton jRadioButton1;
    private javax.swing.JRadioButton jRadioButton2;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    // End of variables declaration//GEN-END:variables
}

class TwoSymbolAutomaton implements CA {

    private final byte[] rhs = new byte[8];

    public TwoSymbolAutomaton(long number) {
        for (int i = 0; i < 8; i++) {
            rhs[i] = (byte) (number % 2);
            number /= 2;
        }
    }

    @Override
    public byte getRhs(int ptr) {
        return rhs[ptr];
    }
}
```

```
class ThreeSymbolAutomaton implements CA {

    private final byte[] rhs = new byte[27];

    public ThreeSymbolAutomaton(long number) {
        for (int i = 0; i < 27; i++) {
            rhs[i] = (byte) (number % 3);
            number /= 3;
        }
    }

    @Override
    public byte getRhs(int ptr) {
        return rhs[ptr];
    }
}

interface CA {

    byte getRhs(int ptr);
}
```