Санкт-Петербургский государственный университет информационных технологий, механики и оптики

На правах рукописи

Наумов Лев Александрович

Метод введения обобщенных координат и инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов

Специальность 05.13.12. Системы автоматизации проектирования (приборостроение)

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель – доктор технических наук, профессор Шалыто А.А.

Санкт-Петербург

2007

Содержание

Введение

Актуальность проблемы. В настоящее время большинство вычислительных задач требует для своего решения ресурсов, превосходящих возможности однопроцессорных компьютеров. Появляется все больше алгоритмов, использующих параллельные или распределенные вычисления для решения задач. Кроме того, возникают новые задачи, которые требуют параллельной или распределенной архитектуры для своего решения.

Принимая во внимание то обстоятельство, что сейчас при разработке сложных программно-аппаратных комплексов значительная доля затрат идёт на разработку программного обеспечения и эта доля неуклонно возрастает, задача создания инструментального средства для автоматизации проектирования программного обеспечения систем с параллельной и распределенной вычислительными архитектурами становится актуальной.

Существует ряд математических моделей параллельных и распределенных вычислений. Одной из них являются клеточные автоматы [1, 2], применению которых и посвящена настоящая диссертация.

Клеточные автоматы [3] – это дискретные динамические системы, поведение которых может быть полностью описано в терминах локальных зависимостей [1]. Эти автоматы представляют собой модели параллельных вычислений, которые обладают сколь угодно большой степенью параллелизма. Можно считать, что они представляют собой дискретный эквивалент понятия "поле". Клеточные автоматы применяются для проведения различных вычислительных экспериментов, так как они удобны, например, для численного решения дифференциальных уравнений и уравнений в частных производных. Они также широко используются для моделирования поведения сложных систем [2, 4, 5].

Существует ряд средств автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов таких, как, например, средство *SIMP/STEP* (реализующее набор функций для решения задач физического моделирования), *CAGE* (инструмент для образовательных целей) или *Mirek's Cellebration* (*MCell*) (средство для визуализации) и *CDL* (инструмент, позволяющий использовать *FPGA*-системы для выполнения вычислительных экспериментов). Однако все эти средства имеют ограничения, которые накладываются на предметные области решаемых задач, реализуемые клеточные автоматы или используемые вычислительные архитектуры. Не существует единого средства, пригодного, как для образовательных, так и для научных целей. Поэтому для автоматизации проектирования программного обеспечения вычислительных экспериментов на основе клеточных автоматов необходимо разработать универсальное, расширяемое инструментальное средство, обладающее широким спектром функциональных возможностей, так как существующие инструменты не позволяют решать разнообразные задачи из различных предметных областей с помощью широкого спектра клеточных автоматов. Так, например, далеко не каждое средство позволяет моделировать автоматы, как с двумерными, так и с трехмерными решетками. Малая их часть поддерживает использование разнообразных окрестностей. Например, окрестность Марголуса поддерживают всего несколько специализированных средств. Не одно из них не допускает хранения строк в клетках, что необходимо при реализации задач параллельного синтаксического анализа. Только некоторые позволяли описывать структуры данных, хранимые в клетках.

Отсутствие универсальных средств во многом связано с тем, что до настоящего времени не был предложен метод введения обобщенных

координат, который позволял бы обеспечить единообразие представления данных о состояниях клеток автоматов с различными решетками (в том числе, и с решетками разной размерности).

Настоящая диссертация посвящена созданию такого метода, а также универсального инструментального средства на его базе. Такое средство позволит использовать естественный параллелизм клеточных автоматов и может быть применено для реализации, визуализации и изучения параллельных и распределенных алгоритмов, которые, например, используются при создании и применении систем автоматизации проектирования приборов. Оно заменит дорогостоящие экспериментальные установки для проведения экспериментов. Кроме того, необходимо отметить, что сложность современного параллельного программного обеспечения такова, что средство автоматизации его проектирования необходимо.

Из изложенного следует, что работа по созданию метода введения обобщенных координат, обеспечивающего единообразие представления данных о состояниях клеток разнообразных решеток клеточных автоматов, а также удобного, универсального, расширяемого средства автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов на его основе является весьма актуальной.

Целью диссертационной работы является разработка метода введения обобщенных координат и создание на его основе инструментального средства автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов, которое позволит проводить эксперименты на различных вычислительных архитектурах.

Основные задачи диссертационной работы состоят в следующем:

1. Разработка метода введения обобщенных координат для решеток клеточных автоматов.
2. Разработка на основе метода введения обобщенных координат инструментального средства, включающего в себя среду выполнения и библиотеку для создания клеточных автоматов, которое предназначено для автоматизации проектирования программного обеспечения вычислительных экспериментов с их использованием.
3. Применение инструментального средства для автоматизации проектирования программного обеспечения ряда вычислительных экспериментов с использованием клеточных автоматов на различных вычислительных архитектурах.
4. Классификация всех структур, которые порождаются одномерными клеточными автоматами из точечного зародыша с целью анализа простейших клеточных автоматов, демонстрирующих сложное поведение (в том числе, самовоспроизведение).

Научная новизна состоит в том, что впервые предложен метод введения обобщенных координат, обеспечивающий единый подход для хранения данных из разнообразных решеток клеточных автоматов, на базе которого создано инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов.

Положения, **выносимые на защиту**:

1. Метод введения обобщенных координат для клеточных автоматов обеспечивает единообразие представления данных для автоматов с различными решетками, в том числе, и разной размерности.
2. Классификация всех структур, порождаемых одномерными клеточными автоматами из точечного зародыша, группирует все типы поведения,

порождаемые простейшими клеточными автоматами (в том числе, самовоспроизведение) в классы эквивалентности.

Методы исследования. В работе использованы методы теории автоматов, теории множеств, теории параллельных вычислений, теории алгоритмов, вычислительной математики, основы объектно-ориентированного программирования.

Достоверность и обоснованность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается доказательствами, обоснованием постановок задач, точной формулировкой критериев, компьютерным моделированием, а также результатами использования методов, предложенных в диссертации, в ходе выполнения вычислительных экспериментов.

Практическое значение работы состоит в том, что разработано универсальное и расширяемое инструментальное средство для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием широкого спектра клеточных автоматов, которое позволяет проводить вычислительные эксперименты на однопроцессорных, многопроцессорных и кластерных платформах.

Реализация результатов работы (имеются акты внедрения). Для образовательных целей результаты используются в Санкт-Петербургском государственном университете информационных технологий, механики и оптики (СПбГУ ИТМО), в Санкт-Петербургском государственном университете и в Университете Амстердама (Нидерланды). Для научно-исследовательских целей результаты используются в СПбГУ ИТМО, в Университете Амстердама и в Политехническом университете Бухареста (Румыния).

Научно-исследовательские работы. Результаты диссертации получены в ходе выполнения в СПбГУ ИТМО научно-исследовательских работ по теме "Разработка технологии автоматного программирования", выполненной по грату РФФИ № 02-07-90114 в 2002-2003 годах; по теме "Разработка технологии программного обеспечения систем управления на основе автоматного подхода", выполняемой по заказу Министерства образования РФ в 2002-2005 годах; по гранту конкурса персональных грантов для студентов и аспирантов вузов Санкт-Петербурга в 2004 году; по теме "Разработка среды и библиотеки "*CAME&L*" для организации параллельных и распределенных вычислений на основе клеточных автоматов", выполненной по гранту РФФИ № 05-07-90086 в 2005-2006 годах. При выполнении работ по этому гранту автор был ответственным исполнителем. Научно-исследовательская деятельность автора дважды отмечена стипендией Президента РФ.

Апробация результатов работы. Основные положения диссертационной работы докладывались на международной научной конференции "*International Conference of Computational Sciences*" (Санкт-Петербург – Мельбурн, 2003 г.), на международной научной конференции "*Cellular Automata for Research and Industry*" (Амстердам, 2004 г.), на научной школе "*Joint Advanced Students School*" (Санкт-Петербург, 2004 г.), на научно-методических конференциях "Телематика-2004", "Телематика-2005", "Телематика-2006" (Санкт-Петербург), на научной школе "*Ferienakademie*" (Южный Тироль, 2005 г.), на всероссийской научной конференции "Научный сервис в сети Интернет: технологии распределенных вычислений" (Дюрсо, 2005 г.), на международной научной конференции "Высокопроизводительные параллельные вычисления на кластерных системах" (Санкт-Петербург, 2006 г.).

Публикации. По теме диссертации опубликовано 14 печатных работ, в том числе, в журналах "Известия РАН. Теория и системы управления" (входит в перечень ВАК), "Информационно-управляющие системы", "Телекоммуникации и информатизация образования", а также в материалах конференций "*International Conference of Computational Sciences*", "*Cellular Automata for Research and Industry*", "Телематика" и "Научный сервис в сети Интернет: технологии распределенных вычислений".

Структура диссертации. Диссертация изложена на 282 страницах и состоит из оглавления, введения, четырех глав, заключения, списка терминов, предметного указателя, списка литературы и двух приложений. Список литературы состоит из 94 наименований. Работа содержит 44 рисунка и 13 таблиц.

Глава 1 содержит обзор истории развития теории клеточных автоматов. В ней вводятся основные понятия этой теории, а также математический аппарат и формализм. Доказывается корректность утверждаемых свойств и вводимых понятий. Предлагается математическая база для дальнейших рассуждений.

Кроме того, приводятся и анализируются требования к инструментальным средствам автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов. Перечисляются и описываются более двадцати существующих продуктов, разработанных для этих целей, аргументируется потребность в расширении этого списка, приводятся основные свойства, возможности и преимущества использования предлагаемого в настоящей работе инструментального средства *CAME&L*.

В главе 2 разрабатывается метод введения обобщенных координат для решеток клеточных автоматов, позволяющий обеспечить универсальный

подход к хранению информации о состояниях клеток для различных решеток. Предлагаемые координаты имеют ряд важных свойств. Введение обобщенных координат фактически представляет собой нумерацию неотрицательными целыми числами элементов многомерного дискретного пространства. Метод иллюстрируется набором примеров для двумерных клеточных автоматов, для которых вводятся:

- спиральные обобщенные координаты для решеток из треугольников;
- спиральные обобщенные координаты для решеток из квадратов;
- спиральные обобщенные координаты для решеток из шестиугольников;
- координаты для решеток из треугольников, базирующиеся на спиральных обобщенных координатах для решеток из шестиугольников;
- обобщенные координаты для решеток из квадратов, основанные на кривых Пеано (*Peano*).

Возможность введения этих и других разновидностей координат поддерживается разработанным в диссертации инструментальным средством. Обсуждаемые в настоящей главе примеры введения обобщенных координат были опробованы при приведении вычислительных экспериментов.

Глава 3 описывает важнейшие компоненты разработанного инструментального средства *CAME&L*: среду выполнения и библиотеку *CADLib* ("*Cellular Automata Development Library*" – "библиотека для разработки клеточных автоматов").

При описании среды рассматриваются основные принципы работы в ней, назначения всех пунктов меню, формат файлов и т. д.

При рассмотрении библиотеки приводятся описания большинства классов, функций, макроопределений и констант. Излагаются особенности

программирования с использованием библиотеки. Описывается применение компонентной модели декомпозиции клеточных автоматов.

В главе 4 приводятся примеры применения рассматриваемого инструментального средства автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов.

На примере игры "Жизнь" демонстрируется разработка простейшего вычислительного эксперимента, а также использование зональной оптимизации, поддержка многопроцессорной и кластерной вычислительных систем, обобщенных координат и создание универсальных правил, не зависящих от платформы.

В следующем разделе этой главы приводится пример численного решения уравнения теплопроводности, который показывает применение предложенного инструментального средства для решения физических задач.

Еще один раздел этой главы посвящен проведению научного исследования с помощью инструментального средства *CAME&L*, которое позволило провести классификацию структур, порождаемых одномерными двоичными клеточными автоматами из точечного зародыша. При этом подсчитано количество классов автоматов при использовании различных операций инвариантности, что вносит вклад в теорию клеточных автоматов.

В заключительном разделе главы описан набор средств, разработанный для того, чтобы организовывать вычисления на *FPGA*-процессорах (*Field Programmable Gate Array*) с помощью инструментального средства *CAME&L*, что расширяет множество вычислительных платформ, на которых могут быть организованы вычислительные эксперименты с помощь метода и инструментального средства, разработанных в диссертации.

Глава 1. Основные понятия теории клеточных автоматов. Обзор инструментальных средств для автоматизации проектирования программного обеспечения вычислительных экспериментов с их использованием

Настоящая глава содержит обзор истории развития теории клеточных автоматов. В ней вводятся основные понятия этой теории, а также математический аппарат и формализм. Доказывается корректность утверждаемых свойств и вводимых понятий. Предлагается математическая база для дальнейших рассуждений.

Кроме того, приводятся и анализируются требования к инструментальным средствам автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов. Перечисляются и описываются более двадцати существующих продуктов, разработанных для этих целей, аргументируется потребность в расширении этого списка, приводятся основные свойства, возможности и преимущества использования предлагаемого в настоящей работе инструментального средства *CAME&L*.

1.1. Введение в историю и идеологию клеточных автоматов

По всей видимости, в конце 60-х годов XX века клеточные автоматы были необходимы для дальнейшего прогресса физики, биологии, химии, математики, компьютерных наук и других областей знаний. Они многократно изобретались под разными названиями, во множестве стран, людьми,

работающими в различных областях науки. В результате, термины "итеративные массивы", "вычисляющие пространства", "однородные структуры" и "клеточные автоматы" практически являются синонимами.

Когда Станислав Улам (*Stanislaw Ulam*) начал свои исследования [6], он вряд ли мог предположить, что, основываясь на его идеях, выдающийся американский ученый Джон фон Нейман (*John von Neumann*) введет понятие "клеточный автомат".

Несмотря на то, что фон Нейман был математиком и физиком, идея пришла к нему при построении объяснении определенных объектов биологии. Он использовал клеточные автоматы для создания более правдоподобных моделей пространственно-протяженных систем. Действительно, механизмы, которые были предложены им для получения самовоспроизводящихся структур на клеточном автомате, сильно напоминают открытые в следующем десятилетии закономерности, наблюдаемые в биологических системах.

Результаты исследований фон Неймана приведены, в частности, в фундаментальной работе [3]. Первое ее издание появилось в 1966 году, в то время как автор умер в 1957 году. Книгу дописывал его ученик, Артур Беркс (*Arthur Burks*), ставший известным специалистом в области клеточных автоматов.

Однако, существенно раньше, в конце Второй мировой войны, в то время как фон Нейман создавал один из первых электронных компьютеров, немецкий инженер Конрад Цузе (*Konrad Zuse*) предложил ряд идей, которые сделали бы его одним из основоположников теории параллельных вычислений, если бы сразу стали широко известными.

Среди прочего Цузе придумал "вычисляющие пространства" [7] – клеточные автоматы. Особый его интерес вызывало применения этих систем

к задачам численного моделирования в механике. К сожалению, ситуация в мире помешала работам ученого получить распространение, в то время как за трудами фон Неймана следил весь научный мир.

Математики пришли к клеточным автоматам, рассматривая итерационные преобразования пространственно-распределенных структур с дискретным набором состояний [8]. Так появились "итерационные массивы".

Сразу стали возникать решения важных теоретических задач в этой области, например, вопросов обратимости, вычислимости, достижимости и т.п.

В группе компьютерной логики университета штата Мичиган Джон Холланд (*John Holland*) впервые применил клеточные автоматы для решения задач адаптации и оптимизации [9].

К сожалению, недостаточность общения и единой терминологии вели к значительному дублированию работ. Так важные характерные особенности клеточных автоматов, доказанные Ричардсоном (*Richardson*) [10] на двадцати страницах в континуальной подстановке в топологии канторовских множеств, могли быть записаны в двух строках в виде следствия предшествующей работы Хедлунда (*Hedlund*) [8].

Понятие "однородные структуры" [11] чрезвычайно близко к "клеточным автоматам" по своему происхождению и обозначает их аппаратную реализацию, в которой ключевым свойством этих систем является их гомогенность.

Спектр применения клеточных автоматов чрезвычайно широк [12–26, 83, 84].

Эффект разорвавшейся бомбы произвела статья ведущего рубрики математических игр и головоломок журнала "*Scientific American*" Мартина Гарднера (*Martin Gardner*) [27–29, 85, 86]. Он опубликовал описание

клеточного автомата Джона Хортона Конвея (*John Horton Conway*) "Жизнь". Игра "Жизнь" стала культовой и сделала понятие "клеточный автомат" неотъемлемой частью бытового жаргона людей с техническим образованием.

Клеточный автомат – это некий мир, живущий по определенным законам. Его эволюция, как жизнь любой замкнутой системы, бесцельна. С течением времени изменяется состояние решетки, но система сама по себе не может заметить этот процесс, не может делать выводы, понять, зачем это необходимо и как долго будет продолжаться. Развитие мира обретает смысл, когда появляется наблюдатель, который смотрит на него из другого мира и анализирует процесс эволюции. Таким образом, рассматриваемые модели служат для изучения эволюционных процессов.

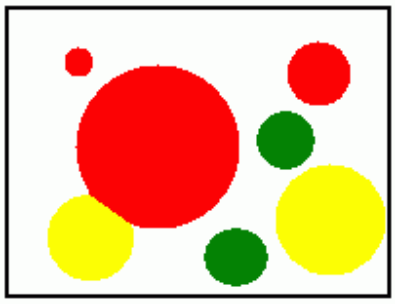
При решении задач моделирования решетка становится пространством для представления некой среды или вещества. Пусть на рис. 1.a схематически изображено какое-то вещество. Зоны, обладающие одинаковым значением некого существенного параметра, выделены одним и тем же цветом. Допустим, что при решении задачи никакие другие характеристики вещества не существенны.

Для численного моделирования необходимо описать вещество для вычислительной машины, определить пространственное распределение значений параметра. Для этого следует произвести дискретизацию, заменить реальный набор данных конечным числом информативных значений.

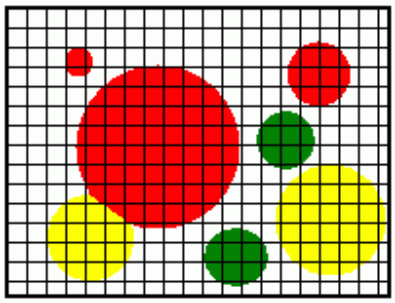
Приведем примеры дискретизации вещества, изображенного на рис. 1.a посредством квадратной и гексагональной решеток. Для этого наложим их на вещество (рис. 1.b и 1.d, соответственно). В каждую ячейку поместим значение, соответствующее той зоне материала, которую она охватывает. Если это – некий числовой параметр, то клетке можно присвоить его среднее значение по всем точкам, охватываемым клеткой, значение из ее центра или,

как в настоящем примере, значение, которым обладает большинство точек (рис. 1.c и 1.e, соответственно). Используются и другие способы определения значения, которое следует поместить в клетку, соответствующую данной области среды.

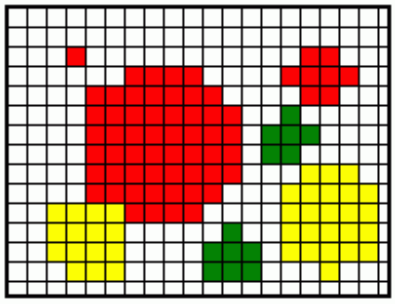
Клеточные автоматы образуют общую парадигму параллельных вычислений подобно тому, как машины Тьюринга образуют парадигму последовательных вычислений. В результате они могут быть использованы для реализации параллельных алгоритмов, как модели, обладающие естественным параллелизмом.



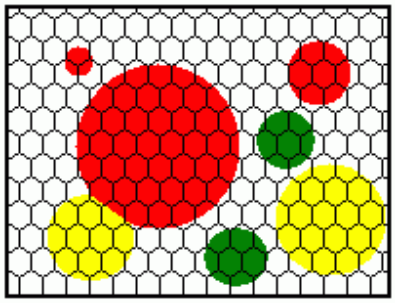
a. Некое вещество



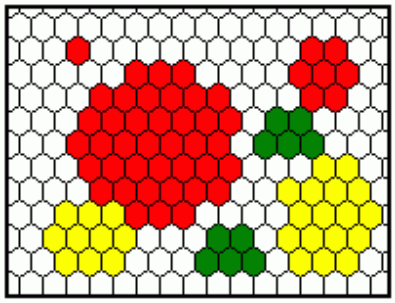
b. Квадратная решетка на веществе



c. Результат дискретизации вещества с помощью квадратной решетки



d. Гексагональная решетка на веществе



e. Результат дискретизации вещества с помощью гексагональной решетки

Рис. 1. Пример дискретизации элемента пространства

Одно из главных отличий клеточной системы от других вычислительных систем состоит в том, что во всех прочих присутствуют две принципиально различные части: архитектурная (управляющая), которая фиксирована и активна (то есть выполняет некоторые операции) и данные,

которые переменны и пассивны (то есть сами по себе они ничего сделать не могут).

У клеточных автоматов обе эти составляющие собираются из принципиально изоморфных, неотличимых друг от друга элементов. Таким образом, вычислительная система может оперировать своей материальной частью, модифицировать, расширять себя и строить себе подобных [3]. Хотя системы этого класса и были придуманы Джоном фон Нейманом, такая параллельная архитектура получила название "не-фон-неймановской", так как "фон-неймановскую" архитектуру он создал раньше.

Данное утверждение может показаться спорным. Казалось бы, к архитектурной части логичнее отнести, например, решетку и правила автомата. Однако это скорее аппаратная часть. Здесь имеется в виду то, что, например, при рассмотрении автомата, реализующего игру "Жизнь", при данных решетке и правилах, изменяя лишь состояния клеток, можно реализовать универсальную вычислительную систему, позволяющую производить любые вычисления. По своим выразительным способностям такая модель будет эквивалентна произвольным машинам Тьюринга и клеточным автоматам [28, 30]. При этом клетки, описывающие архитектуру моделируемой вычислительной системы, будут неотличимы от клеток, представляющих собой данные.

Клеточный автомат, называемый компьютером Бэнкса [1], также обладает свойством вычислительной универсальности. Использовать его крайне неэффективно, но с теоретической точки зрения факт его существования – важный результат.

1.2. Определение клеточного автомата

Формально клеточный автомат A представляет собой четверку объектов $\{G, Z, N, f\}$, где:

- G – дискретное метрическое пространство, решетка автомата, набор его клеток. Наличие метрики позволяет обеспечить то, чтобы все клетки находились на конечном расстоянии от данной, так как ни для какой метрики никакие две точки дискретного метрического пространства не могут быть бесконечно удалены друг от друга. Кроме того, понятие метрики и координат клетки оказывается чрезвычайно полезным при определении множества клеток окрестности N, описываемой далее. Как правило, пространство G является одно-, двух- или трехмерным, однако оно может иметь и большую размерность.
- Z – конечное множество возможных состояний клетки. Таким образом, состояние всей решетки является элементом множества $Z^{|G|}$.
- N – конечное множество, определяющее окрестность клетки – те $|N|$ клеток, которые влияют на новое состояние данной. Его можно задавать, например, как множество относительных смещений из клетки, необходимых для достижения окрестных ячеек. Элементы этого множества будем называть "соседями"[1] клетки.
- f – функция переходов или правила автомата. Вычислимая функция, действующая $Z \times Z^{|N|} \to Z$ для автоматов с клетками с памятью и $Z^{|N|} \to Z$ для автоматов с клетками без памяти. Отличие заключается в том, что в функции переходов клеточных автоматов с клетками с памятью текущее состояние клетки также передается в качестве параметра.

---

[1] Термин "сосед" не следует путать с терминами "главный сосед", "непосредственный сосед", "сосед i-го кольца" и прочими понятиями широко используемыми далее.

Шаг автомата состоит в применении функции переходов к каждой клетке решетки. Он считается завершенным только после того, как новое состояние будет определено для всех элементов.

Клеточные автоматы в общем случае характеризуются следующими фундаментальными свойствами:

1. Решетка однородна – никакие две области не могут быть отличены друг от друга по ландшафту. Это обеспечивается семантикой пространства G, а также единственностью определения окрестности N, множества значений Z и функции переходов f.
2. Взаимодействия локальны. Состояния только клеток окрестности влияют на состояние данной клетки, что обеспечивается семантикой функции f.
3. Обновление значений состояний всех клеток решетки происходит одновременно после вычисления нового состояния каждой из них.

Необходимо отметить, что при решении некоторых задач возникает необходимость отказаться от выполнения этих свойств "классических" клеточных автоматов.

Автоматы, для которых не выполняется третье из них, называются асинхронными клеточными автоматами.

1.3. Теорема о трех двумерных решетках из правильных многоугольников

На практике часто используются клеточные автоматы с двумерными решетками из правильных многоугольников. Автоматы именно с такими решетками будут преимущественно обсуждаться в настоящей работе. На

рис. 2–4 показаны фрагменты треугольной, квадратной и шестиугольной (гексагональной) решеток[2], соответственно.

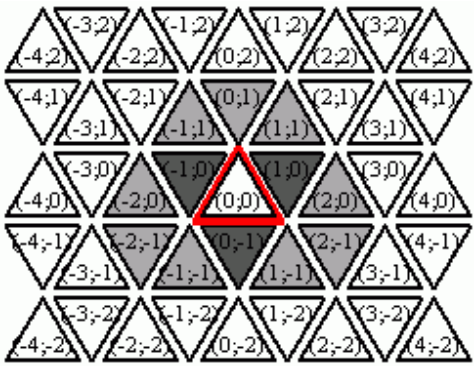


Рис. 2. Фрагмент треугольной решетки

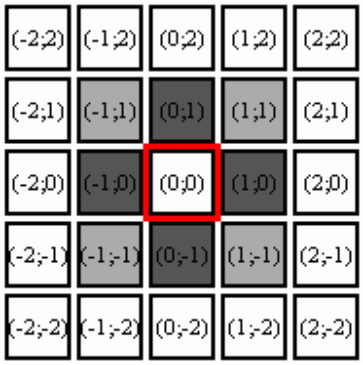


Рис. 3. Фрагмент квадратной решетки

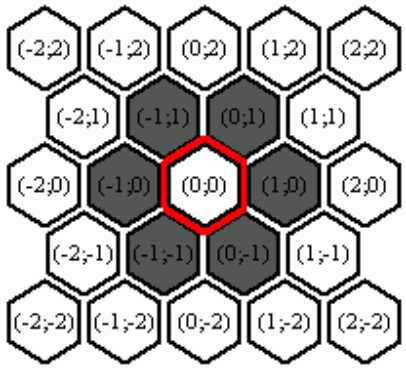


Рис. 4. Фрагмент шестиугольной решетки

---

[2] Термины "квадратная решетка", "решетка квадратов" и "решетка из квадратов" (аналогично для других многоугольников) будем считать синонимами.

В большинстве случаев, в качестве окрестности клетки используют подмножество набора ее непосредственных соседей.

Окрестность, состоящая из клеток, имеющих общую сторону с данной, называется окрестностью фон Неймана или множеством главных соседей. На рис. 2–4 элементы такой окрестности для центральной клетки выделены темно-серым цветом.

Окрестность, состоящая из клеток, имеющих хотя бы общую вершину с данной, называется окрестностью Мура или множеством непосредственных соседей. На рис. 2–4 элементы такой окрестности для центральной клетки выделены светло-серым или темно-серым цветами.

Ясно, что множество непосредственных соседей любой клетки содержит или равно множеству ее главных соседей.

Пусть a – элемент множества G, клетка решетки. Множество непосредственных соседей клетки a будем обозначать, как $S(a)$, множество главных соседей – как $S_0(a)$, а множество соседей клетки a, найденное в соответствии с определением окрестности N – как $N(a)$ (в общем случае оно может не совпадать ни с $S(a)$, ни с $S_0(a)$).

Докажем следующую теорему.

Теорема 1. О трех двумерных решетках из правильных многоугольников.
Не существует другой решетки из правильных многоугольников, кроме треугольной, квадратной и шестиугольной[3].

Доказательство:

[3] В данном случае имеется в виду именно регулярное составление одинаковых клеток так, чтобы они имели общие стороны или вершины.

Сумма углов правильного *n*-угольника $A(n) = \pi \cdot (n-2)$. Тогда $\frac{A(n)}{n}$ – величина каждого угла *n*-угольника.

Пусть из правильных *n*-угольников удалось составить решетку. Тогда полный угол в 2π радиан образуют углы целого числа фигур. Обозначим это число, как *k*.

Таким образом, *k* многоугольников можно составить так, чтобы они имели общую вершину. Определим *k*, как функцию от *n*:

$$2\pi = k \cdot \frac{A(n)}{n} \Leftrightarrow 2\pi = k \cdot \frac{\pi \cdot (n-2)}{n} \Rightarrow k(n) = \frac{2n}{n-2}$$

Будем рассматривать эту функцию при $n \geq 3$, так как треугольник – многоугольник с наименьшим количеством вершин. Возьмем производную от $k(n)$ по *n*:

$$k'(n) = -\frac{n}{(n-2)^2}$$

При $n \geq 3$ функция $k(n)$ убывает, так как $k'(n) < 0$. Следовательно, все возможные значения *k* меньше *k*(3)=6. Отметим также, что $k(n) > 2$, так как истинно следующее неравенство:

$$k(n) > 2 \Leftrightarrow \frac{2n}{n-2} > 2 \Leftrightarrow 2n > 2n - 4 \Leftrightarrow 0 > -4$$

Решетку можно построить, если целому *n* будет соответствовать целое *k*. Из изложенного выше следует, что возможны лишь четыре значения *k* – от трех до шести. Построим функцию *n*(*k*), обратную к *k*(*n*), и проверим, каким из возможных значений *k* соответствует целое *n*:

$$k = \frac{2n}{n-2} \Leftrightarrow k \cdot (n-2) = 2n \Leftrightarrow n \cdot (k-2) = 2k \Rightarrow n(k) = \frac{2k}{k-2}$$

Имеем:

- $k = 3 \Rightarrow n(k) = 6$ – гексагональная решетка.

- $k = 4 \Rightarrow n(k) = 4$ – квадратная решетка.
- $k = 5 \Rightarrow n(k) = \frac{10}{3}$ – не целое *n*, решетку не построить.
- $k = 6 \Rightarrow n(k) = 3$ – треугольная решетка.

Таким образом, действительно существуют только три перечисленные выше двумерные решетки из правильных многоугольников.

1.4. Кольца. Свойства колец

Введем понятие кольца для данной клетки решетки.

Клетка является элементом (соседом) нулевого кольца для самой себя. Ее соседи – элементами первого кольца. Далее – рекурсивно: для данной клетки элементами *i*-го кольца (кольца *i*-го порядка) называются все клетки, являющиеся соседями какой-либо клетки ее (*i*–1)-го кольца, исключая те из них, которые сами принадлежат кольцам меньших порядков.

Формально, если обозначить через $R(a, i)$ множество клеток *i*-го кольца ($i \geq 0$) для клетки *a*, то приведенное выше определение можно записать так:

$$\begin{gathered} R(a,0) = \{a\} \\ R(a,i) = \left\{ b \middle| \exists c \in R(a,i-1) : b \in N(c), \neg\exists j < i : b \in R(a,j) \right\} \end{gathered} \tag{1}$$

Также имеет место следующее соотношение:

$$\left|R(a,i)\right| \leq \left|N\right|^i, i \geq 0 \tag{2}$$

Допустим, что в качестве окрестности клетки будет использоваться множество ее непосредственных соседей – для любой ячейки *a* верно равенство $N(a)=S(a)$. Понятие кольца для всех трех рассматриваемых решеток в этом случае иллюстрируют рис. 5–7. Смежные кольца для центральной клетки выделены разными цветами.

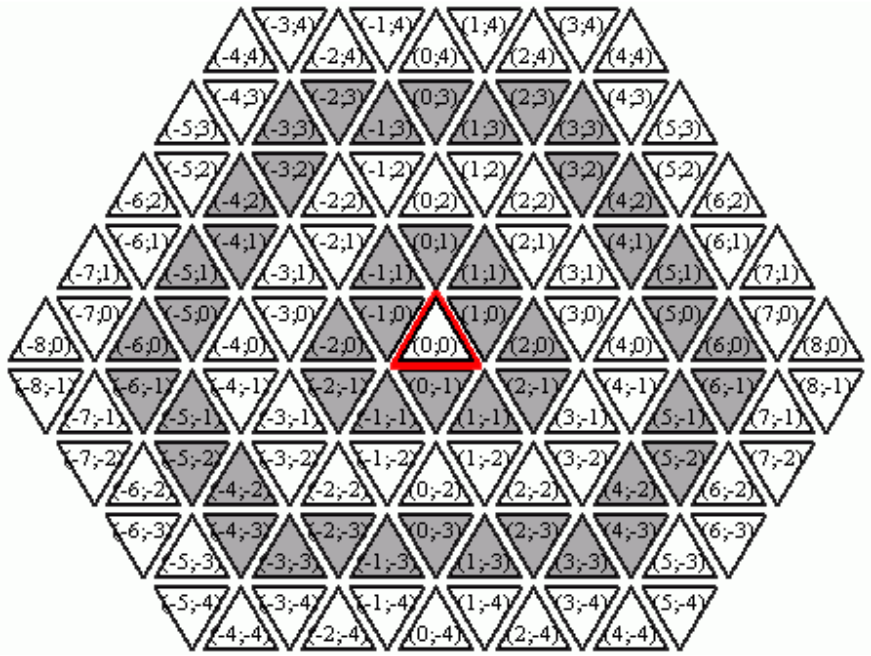


Рис. 5. Кольца для треугольной решетки

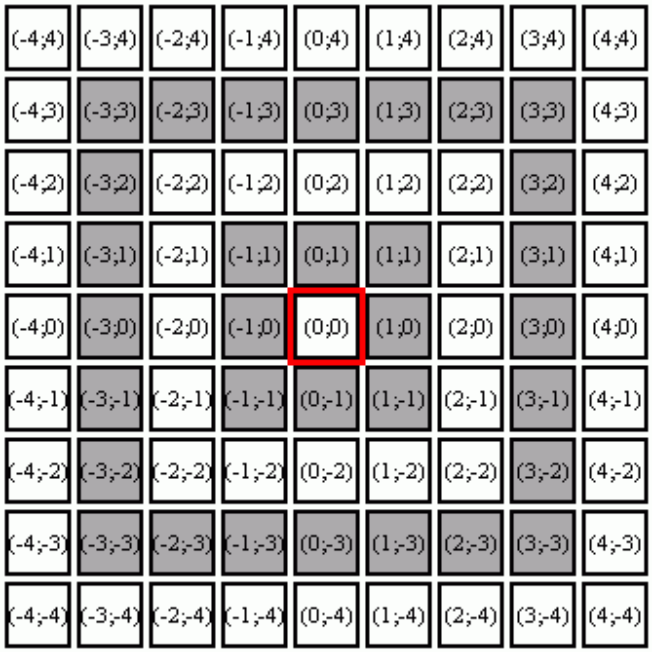


Рис. 6. Кольца для квадратной решетки

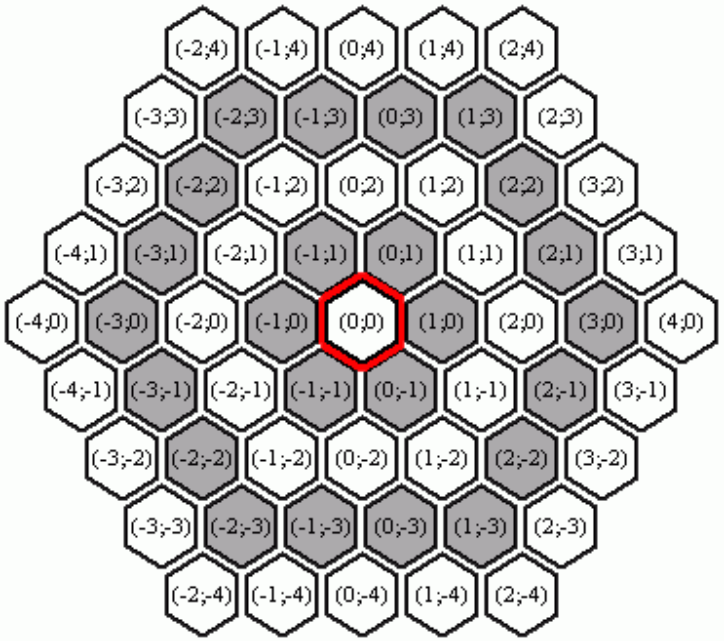


Рис. 7. Кольца для шестиугольной решетки

Далее, если для любых двух клеток a и b выполнение соотношения $a \in N(b)$ влечет выполнение соотношения $b \in N(a)$, то будем называть такую окрестность симметричной. Необходимо отметить, что окрестность, состоящая из непосредственных или главных соседей клетки, является симметричной.

При решении большинства задач рассматриваются симметричные окрестности. Далее в рассуждениях будем полагать окрестности симметричными.

Докажем ряд теорем, утверждающих основное свойство понятия кольца.

Теорема 2. О существовании и единственности индекса кольца.

Для любых двух клеток a и b, существует и единственно неотрицательное число i такое, что $b \in R(a, i)$.

Доказательство:

Существование. Пусть не существует такого i, чтобы $b \in R(a, i)$. Тогда и для любого из соседей клетки b, например, клетки b_1, не существует такого индекса i_1, чтобы $b_1 \in R(a, i_1)$. Для любого из соседей клетки b_1, например, клетки b_2, также не найдется индекс i_2, чтобы $b_2 \in R(a, i_2)$. Продолжая эти рассуждения и учитывая симметричность отношения соседства, приходим к выводу, что в этом случае решетка будет разделена, как минимум, на два подмножества клеток так, что ни одна из клеток одного подмножества не будет соседкой никакой из клеток другого подмножества, что противоречит условию однородности решетки.

Единственность. Пусть значение индекса не единственно и существуют i_1 и i_2, такие, что $b \in R(a, i_1)$ и $b \in R(a, i_2)$. Это означает, что:

$$\exists c_1 \in R(a, i_1 - 1) : b \in N(c_1), \neg \exists j < i_1 : b \in R(a, j)$$

$$\exists c_2 \in R(a, i_2 - 1) : b \in N(c_2), \neg \exists j < i_2 : b \in R(a, j)$$

Из первого утверждения следует, что i_2 не может быть больше i_1, а из второго – что i_1 не может быть больше i_2. Таким образом, i_1 и i_2 равны.

Теорема 3. О симметрии отношения принадлежности и кольцу.

Для любых двух клеток a и b, утверждения $b \in R(a, i)$ и $a \in R(b, i)$ *эквивалентны.*

Доказательство:

Докажем утверждение методом математической индукции по i:

База. Пусть $i=0$. Тогда, если $b \in R(a, 0)$, то a и b совпадают и, следовательно, $a \in R(b, 0)$. Аналогично проводятся рассуждения в другую сторону. Истинность утверждения для $i=1$ вытекает из предположения симметричности отношения соседства.

Переход. Пусть для $i=j-1$ и $i=j$ утверждение верно. И, в частности, $a \in R(b_{j-1}, j-1) \Leftrightarrow b_{j-1} \in R(a, j-1)$ и $a \in R(b_j, j) \Leftrightarrow b_j \in R(a, j)$. Докажем для $i=j+1$, что $a \in R(b_{j+1}, j+1) \Leftrightarrow b_{j+1} \in R(a, j+1)$.

$\Rightarrow$ Докажем, что из утверждения $a \in R(b_{j+1}, j+1)$ следует, что $b_{j+1} \in R(a, j+1)$. Действительно, если $a \in R(b_{j+1}, j+1)$, то, по формуле (1) существует клетка c, такая, что $a \in N(c)$, $c \in R(b_{j+1}, j)$. Кроме того, $a \notin R(b_{j+1}, k)$, где $k<j+1$ в силу теоремы *2*.

По индукционному предположению $c \in R(b_{j+1}, j) \Leftrightarrow b_{j+1} \in R(c, j)$. По предположению симметричности окрестности $a \in N(c) \Leftrightarrow c \in N(a)$. В результате, можно утверждать, что $b_{j+1} \in R(a, j+1)$.

$\Leftarrow$ Докажем, что из утверждения $b_{j+1} \in R(a, j+1)$ следует, что $a \in R(b_{j+1}, j+1)$. Действительно, если $b_{j+1} \in R(a, j+1)$, то, по формуле (1) существует клетка c, такая, что $b_{j+1} \in N(c)$, $c \in R(a, j)$. Кроме того, $b_{j+1} \notin R(a, k)$, где $k<j+1$.

По индукционному предположению $c \in R(a, j) \Leftrightarrow a \in R(c, j)$. По предположению симметричности окрестности $b_{j+1} \in N(c) \Leftrightarrow c \in N(b_{j+1})$. В результате, можно утверждать, что $a \in R(b_{j+1}, j+1)$.

Таким образом, действительно $b_{j+1} \in R(a, j+1) \Leftrightarrow a \in R(b_{j+1}, j+1)$.

1.5. Метрика

Введем метрику [31] для пространства G. Это можно сделать множеством способов. В качестве примера выберем следующий: расстоянием $D(a, b)$ от клетки a до клетки b будем называть номер кольца клетки a (или b), которому принадлежит клетка b (или a). Формально, выражение будет иметь вид:

$$D(a,b) = i : a \in R(b,i)$$
$$D(a,b) = i : b \in R(a,i) \qquad (3)$$

Докажем, что введенная таким образом функция является метрикой.

Теорема 4. О метрике.

Функция D(a, b), определенная формулой (3) является метрикой.

Доказательство:

Проверим три основные свойства метрики [31].

1. $D(a, b)$ неотрицательно определена, так как по определению индекс кольца неотрицателен.
2. $D(a, b)=D(b, a)$ – верно, так как $b \in R(a, i) \Leftrightarrow a \in R(b, i)$ по теореме *3*.
3. Докажем правило треугольника, то есть тот факт, что $D(a, b) \le D(a, c)+D(c, b)$, методом математической индукции по $D(c, b)$:

База. Если $D(c, b)=0$, то клетки b и c совпадают и $D(a, b)=D(a, c)+0$.

Переход. Пусть для всех клеток b и c_i таких, что $D(c_i, b)=i$, и любых a, верно утверждение $D(a, b) \le D(a, c_i)+ D(c_i, b)$. Докажем, что для всех клеток b и c_{i+1} таких, что $D(c_{i+1}, b)=i+1$, и любых a, верно утверждение $D(a, b) \le D(a, c_{i+1})+D(c_{i+1}, b)$.

Выберем клетку c_i, которая является соседней с клеткой c_{i+1}. В результате значение $D(a, c_{i+1})$ будет находиться между $D(a, c_i)-1$ и $D(a, c_i)+1$. Проверим неравенство $D(a, b) \le D(a, c_{i+1})+D(c_{i+1}, b)$ для всех трех вариантов, принимая во внимание, что $D(a, b) \le D(a, c_i)+i$ по индукционному предположению:

- $D(a, c_{i+1})=D(a, c_i)-1 \Rightarrow D(a, b) \le D(a, c_i)-1+(i+1)=D(a, c_i)+i$ – верно.
- $D(a, c_{i+1})=D(a, c_i) \Rightarrow D(a, b) \le D(a, c_i)+(i+1)$ – верно.
- $D(a, c_{i+1})=D(a, c_i)+1 \Rightarrow D(a, b) \le D(a, c_i)+1+(i+1)=D(a, c_i)+(i+2)$ – верно.

Таким образом, индукционное предположение доказано и функция $D(a, b)$, определенная формулой (3) является метрикой.

Понятие кольца и расстояния могут быть обобщены и распространены на случай решеток произвольной размерности.

Необходимо отметить, что вопрос о метрике для решетки клеточного автомата не поднимался другими авторами [5]. Тем не менее, она неявно присутствует при решении подавляющего большинства задач хотя бы потому, что окрестность клеток, обычно описывается в терминах относительных смещений из данной.

Для одной и той же метрики, в зависимости от задачи, координаты можно ввести различными способами. Простейший из них – декартовы координаты, при использовании которых каждая клетка характеризуется двумя (в случае двумерной решетки) целыми числами x и y.

Пусть клетка a имеет координаты $(x_a. y_a)$, а клетка b – $(x_b. y_b)$. Тогда описанную выше метрику $D(a, b)$ можно ввести по следующей формуле:

$$D(a,b) = \max\left(|x_b - x_a|, |y_b - y_a|\right) \quad (4)$$

1.6. Теорема об эквивалентности клеточного автомата набору конечных автоматов

Тот факт, что клеточные автоматы являются моделью параллельных вычислений, позволяет сделать предположение, что они представляют собой множество моделей последовательных вычислений, например, конечных автоматов.

Введем определения конечных автоматов Мили и Мура [32].

Конечный автомат Мили есть пятерка объектов $A = \{Z, X, Y, f, g\}$, где

- Z – конечное множество состояний автомата.

- X – конечное множество входных воздействий на автомат.
- Y – конечное множество выходных воздействий автомата.
- f – функция переходов автомата, $Z \times X \to Z$.
- g – функция выходов автомата, $Z \times X \to Y$.

Конечный автомат Мура есть пятерка объектов $A=\{Z, X, Y, f, g\}$, отличающаяся от автомата Мили лишь тем, что функция выходов g – действует следующим образом: $X \to Y$.

Понятие "функция" в двух последних определениях может быть заменено понятием "вычислимая программа".

Докажем теорему, утверждающую, что клеточный автомат эквивалентен не более чем счетному набору одинаковых конечных автоматов Мили.

Теорема 5. Теорема об эквивалентности набору конечных автоматов.
Клеточный автомат $A=\{G, N, Z, f\}$ может быть реализован посредством множества конечных автоматов Мили $\{A_i\}$, где $A_i=\{Z_A, X_A, Y_A, f_A, g_A\}$, где $1 \le i \le |G|$.

> Доказательство:
> Приведем формальный конструктивный метод построения множества конечных автоматов по клеточному автомату.
> Пусть дан клеточный автомат $A=\{G, N, Z, f\}$. Рассмотрим построение набора конечных автоматов Мили $\{A_i\}$, где $A_i=\{Z_A, X_A, Y_A, f_A, g_A\}$, реализующего функциональность эквивалентную функциональности автомата A.
> Каждый конечный автомат будет выполнять функции одной клетки. При этом в нем должны быть две внутренние переменные. Назовем их

"новое состояние" (она будет хранить новое значение переменной состояния, вычисленное для следующего шага) и "значение определено" (булев флаг, значение которого равно единице, если "новое состояние" уже было вычислено и нулю в противном случае), изначально установленный в ноль.

Опишем пять составляющих компонентов автомата Мили. Здесь и далее в данном доказательстве символом "⊥" будем обозначать, так называемый, "сигнал обновления", которым обмениваются автоматы для синхронизации.

- $Z_A=Z$. Переменная, хранящая состояние конечного автомата имеет то же множество значений, что и для клетки клеточного автомата. Значения переменной "новое состояние", а также булевского флага "значение определено" не следует включать в состояние автомата в целом, так как обмен ими с соседними автоматами не должен производиться.
- $X_A=Z^{|N|}\cup\{\perp\}$. Через входные воздействия автомат получает информацию о состоянии соседей. Кроме того, к множеству входных воздействий необходимо добавить сигнал обновления для обеспечения единовременности присвоения клеткам новых значений.
- $Y_A=Z\cup\{\perp\}$. Через выходные воздействия автомат сообщает соседям свое состояние или пересылает сигнал обновления.
- f_A должна реализовывать следующую функциональность. Если входное воздействие на автомат не равно сигналу обновления, то f_i должна вызвать для соответствующей клетки функцию переходов автомата $f(t)$ (где $t\in X_A$ для клеточного автомата с клетками без

памяти и $t \in X_A \times Z$ для клеточного автомата с клетками с памятью). Полученное значение следует сохранить во внутренней переменной "новое состояние". Переменной "значение определено" при этом должно быть присвоено значение единица. Результат, возвращаемый при этом функцией f_i должен быть равен номеру текущего состояния автомата. Если входное воздействие на автомат равно сигналу обновления, то функция f_i должна вернуть значение внутренней переменной "новое состояние", сменив, тем самым, свое состояние.

- g_A должна реализовывать следующую функциональность. Если входное воздействие на автомат не равно сигналу обновления то результат, возвращаемый функцией должен быть равен номеру текущего состояния автомата. Если входное воздействие на автомат равно сигналу обновления, то поведение определяется состоянием флага "значение определено". Если флаг установлен (равен единице), то функция должна вернуть значение, соответствующее сигналу обновления. Если он не установлен, то возвращаемый функцией результат должен быть равен номеру текущего состояния автомата.

По завершении каждой итерации автоматы должны получить сигналы обновления, для чего достаточно подать этот сигнал на вход одному из них.

Необходимо отметить, что для асинхронных автоматов потребность в сигналах обновления отпадает.

В силу того, что вопросы преобразования конечных автоматов Мили в конечные автоматы Мура рассматривались неоднократно [33], отдельно рассматривать эквивалентность клеточного автомата набору автоматов Мура не имеет смысла.

1.7. Аппаратные и программные реализации клеточных автоматов

Клеточный автомат – модель, обладающая естественным параллелизмом и для того, чтобы получить некие преимущества от ее использования, необходимо применять параллельное аппаратное или программное обеспечение.

Использование специализированного оборудования выглядит весьма заманчивым, так как при решении задач с помощью клеточных автоматов требуется большой объем памяти для хранения состояний решетки. Однако взаимодействие переменных, соответствующих клеткам, локальны, что может быть выгодно использовано при аппаратной реализации.

Кроме того, при проведении экспериментов с помощью клеточных автоматов, необходимо выполнять огромное количество итераций. Допустим, для получения удовлетворительных результатов при решении прикладной задачи требуется сделать порядка 10^{15} шагов. По чрезвычайно оптимистичной оценке, при моделировании клеточного автомата на персональном компьютере с архитектурой *iX86*, итерация может потребовать несколько микросекунд. Тогда эксперимент займет тысячелетия. Специализированное аппаратное обеспечение позволит сделать эти показатели более реальными.

Самый известный образец такого оборудования был разработан группой информационной механики Массачусетского технологического института и называется *CAM* (*Cellular Automata Machine* – машина клеточных автоматов) [1, 34].

Довольно популярной была шестая версия этого продукта, *CAM*-6, представляющая собой *PCI*-устройство, подключаемое к компьютеру с архитектурой *iX86* под управлением операционной системы *PC-DOS*. Оно

использует видеосистему персонального компьютера для визуализации состояния решетки автомата, а также его электропитание, дисковую память и т.д. В обмен машина предоставляет свои вычислительные возможности. Симбиоз получался чрезвычайно выгодным. Применению *CAM*-6 посвящена работа [1]

Последняя, на данный момент, версия этого устройства, *CAM*-8 [34], работает в тандеме с компьютерами семейства *Sun workstation* посредством интерфейса *SBus*. На компьютере должно быть запущено прилагаемое к устройству программное обеспечение *STEP*.

Для программирования этих устройств используется интерпретируемый язык *Forth* [35], расширенный и модифицированный для поддержки соответствующих примитивов.

Для программирования *CAM*-8 на низком уровне разработана библиотека *StepLib* для языка *C*.

Однако аппаратные решения требуют капиталовложений, четкого осознания потребностей решаемой задачи и используются, преимущественно, в специализированных лабораториях. Инструментальные средства автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов позволяют легко превратить офис или компьютерный класс университета в вычислительный центр. Соотношение цена/качество для них выглядит в настоящее время наиболее перспективным, так как возможности персональных компьютеров неумолимо растут.

Это иллюстрируется тем фактом, что группа ученых под руководством Томазо Тоффоли (*Tommaso Toffoli*), создавшая *CAM*, сначала работала над инструментальным средством для моделирования автоматов [36]. Потом из него "выросло" упомянутое выше аппаратное решение. Однако сейчас эта

группа снова работает над программным средством моделирования (проект носит имя *SIMP*/*STEP* и описывается в разд. 1.9), а разработка *CAM* приостановлена.

1.8. Требования к средствам автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов

Перечислим требования к средствам автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов. Приводимые ниже характеристики не являются обязательными, однако, такая функциональность позволит проекту претендовать на то, чтобы быть универсальной средой для клеточных вычислений. Инструментальное средство, по возможности, должно:

1. Предоставлять пользователям дружественный, удобный, современный интерфейс с богатым набором возможностей для работы.
2. Наглядно визуализировать состояния решетки, обеспечивать удобную навигацию по ней.
3. Быть максимально универсальным, не накладывать ограничений на множество автоматов, которые можно использовать.
4. Выполнять итерации, используя современные вычислительные и коммуникационные технологии, быстро и эффективно распараллеливая вычисления автомата с помощью доступных ресурсов (машин вычислительного кластера, процессоров многопроцессорной системы и т.д.).

5. Оптимизировать вычисления, например, исключать не обновляемые зоны решетки из рассмотрения, использовать шаблоны для сравнения окрестностей во избежание лишних трудоемких вычислений и т.п.
6. Позволять удобно и надежно проводить продолжительные (возможно, многомесячные) вычислительные эксперименты. Несмотря на то, что почти все вычислительные ресурсы системы будут заняты осуществлением итераций, программа должна продолжать быть интерактивной и отвечать на запросы пользователя.
7. Позволять производить не только сами вычисления, но и анализ результатов вычислений, а именно, поддерживать построение разнообразных графиков, демонстрировать изменения различных параметров моделируемой системы и т.п.

1.9. Обзор существующих средств автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов

В настоящем разделе приводится список из 21 наиболее популярного и, следовательно, удачного средства автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов [87]. Для каждого продукта указывается следующая информация:

- *Название.* Имена авторов.
  - Условия распространения.
  - Требования к аппаратному обеспечению, а также операционной системе.
  - Краткий комментарий.

Далее приводится список продуктов в алфавитном порядке.

1. *CAGE* [37]. Авторы: Д. Трунфио (*D. Trunfio*) и другие.
   - Распространяется свободно, однако официальный релиз еще не выпущен, продукт не готов.
   - Рабочая станция под управлением операционной системы семейства *Windows* или *Windows NT*.
   - "*Cellular Automata General Environment*" – один из самых новых продуктов. Программа изначально ориентирована на использование в образовательных целях. Не поддерживает распределенных или параллельных вычислений. Обладает чрезвычайно богатым интерфейсом. Однако видимо, спектр его возможностей настолько широк, что его приходится признать сложным для понимания. Это обстоятельство в купе с тем, что документация к проекту доступна только на итальянском языке, создает существенную преграду для его применения. Функции переходов задаются на С-подобном языке и компилируются. Одним из преимуществ данного средства является то, что оно позволяет использовать нерегулярные решетки.
2. *CAM Simulator* [38]. Авторы: З. Белзо (*Z. Belso*) и М. Варджиас (*M. Vargyas*).
   - Распространяется свободно.
   - Рабочая станция под управлением операционной системы *MS-DOS*.
   - Программа представляет собой симулятор устройства *CAM*-6 (*Cellular Automata Machine*) [1, 34], от которого программа унаследовала ряд существенных ограничений (использование только квадратной решетки и окрестности, состоящей лишь из непосредственных соседей и т.п.).
3. *CAMEL* [39]. Авторы: Д. Талия (*D. Talia*) и Дж. Спеззано (*G. Spezzano*).

- Свободно не распространяется, можно получить только версию для однопроцессорной машины, что лишает смысла применение этого средства для реальных задач.
- Рабочая станция под управлением операционной системы *UNIX*/*Linux* или вычислительный кластер, использующий библиотеку *MPICH*, реализующую стандарт *MPI* [40].
- По иронии судьбы, этот проект имеет практически то же название, что и проект, описываемый в настоящей работе (амперсант в названии последнего появился именно в связи с существованием этого проекта). Однако, в данном случае слово "*CAMEL*" означает "*Cellular Automata environMent for systEms modeLing*". Для описания автоматов, определения решетки, задания функции переходов, окрестности и прочего используется специально разработанный авторами язык *CARPET* (*CellulAR Programming EnvironmenT*) [41]. Программа с этого языка транслируется в программу на языке *C* с использованием библиотеки *MPICH* и выполняется средой *CAMEL*. Поддерживаются только квадратные и не более чем трехмерные решетки. Проект довольно старый. Пользовательский интерфейс реализован с помощью библиотеки *Motif*.

4. *CANL*. Авторы: Р. Наполитано (*R. Napolitano*), К. ди Наполи (*C. Di Napoli*).
   - Свободно не распространяется.
   - Платформа *Cray YMP* или *Sun workstation*.
   - Программа поддерживает автоматы с двумерными квадратными решетками и декартовыми метриками. В окрестности могут присутствовать только непосредственные соседи.
5. *CAPow* [42]. Автор: Р. Ракер (*R. Rucker*).

- Распространяется свободно.
- Рабочая станция под управлением с операционной системы семейства *Windows* или *Windows NT*.
- Проект с очень богатым инструментарием для визуализации результатов. Программа поддерживает автоматы с одно- и двумерными решетками и декартовыми метриками. Правила для автомата описываются на языке *C++* с использованием предназначенной для этого библиотеки. Собранные в динамическую библиотеку правила выполняются средой. Однако средство не поддерживает распределенных или параллельных вычислений.

6. *CASim* [43]. Автор: С. Колер (*S. Kohler*).
   - Распространяется свободно.
   - Рабочая станция под управлением операционной системой семейства *UNIX*/*Linux* с установленной библиотекой *X11*/*Motif*.
   - Программа позволяет моделировать автоматы с двумерными решетками и декартовыми метриками.
7. *CAT/CARP* [44]. Авторы: И. Ясеньяк (*I. Jaceniak*), С. Фоке (*S. Focke*), У. Линк (*U. Link*) и Г. Юнгер (*G. Junger*).
   - Распространяется свободно.
   - Рабочая станция под управлением операционной системы семейства *Windows*, *Windows NT* или *OS/2*.
   - Программа поддерживает автоматы с двумерными квадратными решетками и декартовыми метриками.
8. *CDL* [45]. Автор: К. Хошбергер (*C. Hochberger*).
   - Распространяется свободно.

- Рабочая станция под управлением операционной системы семейства *UNIX*/*Linux*, а также вычислительный кластер или платформа *FPGA* семейства *CEPRA*.
- Программа поддерживает автоматы с двумерными квадратными решетками и декартовыми метриками. Правила задаются на языке *C* и компилируются.

9. *CDM/SLANG* [46]. Автор: Х. Зибург (*H. Sieburg*).
   - Распространяется свободно.
   - Рабочая станция *Apple Macintosh*.
   - Программа поддерживает автоматы с двумерными решетками и декартовыми метриками.
10. *CelLab* [47]. Авторы: Р. Ракер (*R. Rucker*) и Дж. Уолкер (*J. Walker*).
    - Распространяется свободно.
    - Рабочая станция под управлением операционной системы семейства *Windows*.
    - Правила пишутся на языке *C* и компилируются.
11. *CELLAS/FUNDEF*. Автор: Т. Легенди (*T. Legendi*).
    - Свободно не распространяется.
    - Рабочая станция под управлением операционной системы *MS-DOS*.
    - Программа поддерживает автоматы с двумерными решетками и декартовыми метриками.
12. *Cellsim* [48]. Авторы: К. Лэнгтон (*C. Langton*) и Д. Хибелер (*D. Hiebeler*).
    - Распространяется свободно.
    - Рабочая станция под управлением операционной системы семейства *UNIX*/*Linux* или станция *Connection Machine CM*-2.

- Программа поддерживает автоматы с одно- и двумерными квадратными решетками и декартовыми метриками, а также окрестности радиусом не больше трех. Программа разрабатывалась в Университете Сантафе (США), одном из лидеров в области исследования клеточных автоматов.

13. *Cellular/Cellang* [49]. Автор: Дж. Дана Эккарт (*J. Dana Eckart*).
    - Распространяется свободно.
    - Рабочая станция *Sun workstation* или *SGI* с операционной системой семейства *UNIX/Linux*. Также поддерживается работа под управление операционной системы семейства *MS-DOS*.
    - Программа поддерживает автоматы с произвольными квадратными, кубическими, гиперкубическими решетками и декартовыми метриками.
14. *CEPROL.* Автор: F. Зойтер (*F. Seutter*).
    - Условия распространения неизвестны.
    - Рабочая станция под управлением операционной системы семейства *UNIX*/*Linux*.
    - Программа поддерживает автоматы с двумерными решетками и декартовыми метриками. Правила пишутся на языке *Pascal* и компилируются.
15. *DDLab* [50]. Автор: К. Ленгтон (*A. Wuensche*).
    - Распространяется свободно.
    - Рабочая станция *Sun workstation*, *Apple Macintosh* или станция под управлением операционной системой семейства *MS-DOS*.
    - Весьма богатый по своим возможностям проект, который разрабатывался в Университете Сантафе. Не поддерживает распределенных или параллельных вычислений.

16. *HICAL* [51]. Авторы: А. Бекерс (*A. Beckers*), Т. Уорш (*T. Worsch*) и другие.
    - Распространяется свободно.
    - Рабочая станция под управлением операционной системы семейства *UNIX*/*Linux* с системой *X11*.
    - Инструментальное средство с довольно широкими возможностями. Поддерживает автоматы с одно- и двумерными квадратными решетками и декартовыми метриками. Позволяет использовать различные локальные правила, а также реализовывать межавтоматные взаимодействия и использовать структурированные значения состояний клеток. Удобный инструментарий для наблюдения за экспериментами.
17. *LCAU* [52]. Автор: Х. В. Макинтош (*H. V. Macintosh*).
    - Распространяется свободно.
    - Рабочая станция под управлением операционной системы *MS-DOS* или станция *NeXTSTEP*.
    - Программа предоставляет качественно новые и важные средства исследования клеточных автоматов. В частности, с ее помощью можно находить циклы в изменении состояния решетки, строить диаграммы де Брюна, и определять, с некоторой вероятностью, предыдущее состояние решетки. Однако продукт не поддерживает распределенных или параллельных вычислений.
18. *Mirek's Cellebration* (*MCell*) [53–55]. Автор: М. Выйтовиц (*M. Wyjtowicz*).
    - Распространяется свободно.
    - Рабочая станция под управлением операционной системы семейства *Windows*.

- Несмотря на значительный срок с момента релиза (последняя версия датирована 2001 годом) это – одна из самых популярный программ рассматриваемого класса. Хотя программа не поддерживает средств для параллельных или распределенных вычислений и позволяет использовать только квадратные решетки, богатство набора примеров делает ее идеальной для визуализации эволюции состояний. Однако средства определения функции переходов неудобны.

19. *SCARLET* [56]. Авторы: М. Кутриб (*M. Kutrib*) и другие.
    - Условия распространения неизвестны.
    - Рабочая станция под управлением операционной системы *Sun Solaris* 2.*x*.
    - Программа поддерживает автоматы с произвольными решетками и декартовыми метриками. В качестве состояний клеток могут быть использованы целые числа и строки.
20. *SIMP/STEP* [57]. Авторы: Т. Тоффоли (*T. Toffoli*), Т. Бах (*T. Bach*).
    - Распространяется свободно с исходным кодом.
    - Рабочая станция под управлением операционной системы семейства *UNIX*/*Linux*.
    - Данный продукт разрабатывается под руководством одного из авторов аппаратной платформы для моделирования клеточных автоматов, устройства *CAM* (*Cellular Automata Machine*) [1, 38], который возглавляет сейчас "лабораторию программируемой материи" в Бостонском университете. Программа позволяет описывать, инициализировать, генерировать, визуализировать и анализировать эксперименты на клеточных автоматах. Поддерживает только предопределенные картезианские метрики.

Функции переходов описываются с помощью интерпретируемого языка *Python*.

21. *WinCA* [58]. Авторы: Б. Фиш (*B. Fisch*) и Д. Гриффит (*D. Griffeath*).
    - Распространяется свободно.
    - Рабочая станция под управлением операционной системы семейства *Windows* или *Windows NT*.
    - Программа поддерживает автоматы с двумерными решетками и декартовыми метриками. Не поддерживает распределенных или параллельных вычислений.

В таблице 1 знаками "+" и "–" показано поддерживает ли тот или иной продукт каждое из семи требований к инструментальным средствам обсуждаемого класса, перечисленных в разд. 1.9.

Таким образом, ни одно из перечисленных выше инструментальных средств автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов ни только не удовлетворяет всем требованиям, но даже не поддерживает больше трех из них. Особенно плохо обстоит дело с такими свойствами, как универсальность (третье свойство), поддержка технологий параллельных и распределенных вычислений (четвертое свойство), оптимизация (пятое свойство) и анализ (седьмое свойство), без выполнения которых трудно представить себе инструментальное средство для научных вычислений.

Таблица 1. Наличие тех или иных свойств у обсуждаемых инструментальных средств

Название	1	2	3	4	5	6	7
CAGE	+	+	–	–	–	+	–
CAM Simulator	–	+	–	–	–	+	–
CAMEL	–	–	+	+	–	+	–
CANL	+	–	–	+	–	+	–
CAPow	+	+	–	–	–	+	–
CASim	–	–	–	–	–	+	–
CAT/CARP	–	–	–	–	–	+	–
CDL	–	–	–	+	–	+	–
CDM/SLANG	–	–	–	–	–	+	–
CelLab	–	–	–	+	–	+	–
CELLAS/FUNDEF	–	–	–	–	–	+	–
Cellsim	+	+	–	–	–	+	–
Cellular/Cellang	+	–	+	–	–	+	–
CEPROL	–	–	–	–	–	+	–
DDLab	+	+	–	–	–	+	–
HICAL	+	–	–	–	–	+	+
LCAU	–	+	–	–	–	+	+
Mirek's Cellebration	+	+	–	–	–	+	–
SCARLET	+	+	–	–	–	+	–
SIMP/STEP	–	–	–	+	+	+	–
WinCA	+	+	–	–	–	–	–

1.10. Причины возникновения нового инструментального средства автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов

Анализ перечисленных выше существующих средств подтвердил адекватность и необходимость разработки нового инструментального средства автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов в силу приведенных ниже обстоятельств:

- Большинство из существующих средств решения задач с помощью клеточных автоматов не удовлетворяют требованиям к пользовательскому интерфейсу и не предоставляют возможностей, которые стали обычными и неотъемлемыми, на поддержку которых пользователь имеет основания рассчитывать. В настоящее время отсутствуют инструментальные средства со столь удобным интерфейсом, каким обладает среда, входящая в рассматриваемый пакет. В ней, в частности, реализованы:
  - поддержка буфера обмена;
  - поддержка отмены/повторения последней операции;
  - поддержка печати;
  - возможность сохранения состояния решетки в виде рисунков для иллюстрирования публикаций.
- Все известные продукты накладывают жесткие ограничения на используемый тип клеточных автоматов. Разрабатываемая среда универсальна и позволяет использовать модели, принадлежащие даже более широкому классу, чем класс "клеточные автоматы".

- Почти все средства используют различные языки для описания автомата, причем иногда эти языки весьма сложны. Таким образом, появляется еще один навык, которым должен овладеть исследователь прежде, чем приступить к решению задачи.
- Большинство известных средств моделирования клеточных автоматов разрабатывались для операционных систем семейства *UNIX*/*Linux*. По крайней мере, хорошие экземпляры существуют только для них. Предлагаемое инструментальное средство разработано для операционных систем семейства *Windows NT*[4]. Однако части, не связанные с пользовательским интерфейсом, легко переносимы на другую операционную систему.
- Подавляющее большинство существующих продуктов не "моделируют" клеточные автоматы, а только "имитируют" их, так как не используют естественного параллелизма этих моделей, не поддерживают средств для осуществления параллельных или распределенных вычислений. При решении задач на кластерной платформе инструментальное средство *CAME&L* использует специализированный сетевой протокол *CTP* (*Commands Transfer Protocol*) [88–90], разработанный автором. Поддержка многопроцессорной платформы также предусмотрена.

Рассмотрим вновь семь требований к инструментальным средствам автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов, перечисленных в

---

[4] Имеются в виду такие операционные системы, как *Windows NT* 4.*x*, *Windows* 2000, *Windows XP* и т.д.

разд. 1.9 и опишем, в какой степени они будут присущи предлагаемому инструментальному средству.

1. Интерфейс предлагаемого инструментального средства должен быть весьма разнообразным и поддерживать множество типичных функций, присутствия которых пользователь в праве ожидать от современного программного обеспечения.
2. За визуализацию состояния решетки и удобную навигацию по ней отвечает отдельный класс компонентов продукта. Таким образом, в стандартную поставку включен набор средств для удовлетворения этого требования, кроме того, любой пользователь может расширить его, разработав свое средство, обладающее конкретными специфическими свойствами, встраиваемое в среду
3. Инструментальное средство должно позволять моделировать класс систем даже более широкий, чем класс "клеточные автоматы".
4. Проект будет позволять организовывать вычисления, как на отдельном персональном компьютере, так и на многопроцессорной системе и в вычислительном кластере.
5. Средства для оптимизации вычислений будут включены в стандартный набор поставки программного обеспечения и могут быть использованы при решении пользовательских задач. Этот набор может также быть легко расширен пользователями.
6. Проект должен позволять удобно и надежно проводить продолжительные вычислительные эксперименты, сохраняя при этом интерактивность и предоставляя возможности управления его течением.
7. Инструментальное средство будет использовать специальные компоненты для наблюдения и изучения динамики ключевых

параметров эксперимента, построения графиков, файлов-отчетов и прочих задач такого рода.

1.11. Компонентная модель декомпозиции клеточных автоматов

Каждый клеточный автомат предлагается представлять набором компонентов следующих четырех типов[5]:

- Решетка (*grid*) – осуществляет визуализацию автомата и навигацию по клеткам.
- Метрика (*metrics*) – сопоставляет клеткам координаты, вводит отношение соседства и функции вычисления расстояний между клетками по разным направлениям.
- Хранилище данных (*datum*) – обеспечивает хранение, обмен и преобразования состояний клеток, а также некоторые аспекты визуализации (соответствие между состоянием клетки и цветами, используемыми для ее отображения).
- Правила (*rules*) – описывает итерацию. Не только функцию переходов, но и метод разделения задачи на подзадачи, оптимизацию вычислений, а также многое другое.

Идеология такой компонентной декомпозиции клеточных автоматов позволяет "собирать" автоматы с различной функциональностью, посредством незначительных изменений переходить от одной задачи к другой, пробовать решать одну и ту же задачу на разных решетках, с

---

[5] Необходимо отметить, что речь идет не о тех же четырех составляющих, что в разд. 1.2. В данном случае декомпозицию логичнее вести по другим принципам и руководствоваться вопросам реализации автомата для решения задач.

различными метриками и т.п. Кроме того, она предоставляет возможность распределять разработку программного обеспечения эксперимента между различными исполнителями.

Каждый компонент должен декларировать список собственных параметров, изменение значений которых обеспечивает настройку компонента.

Компонент, реализующий правила автомата, декларирует также список анализируемых параметров, характеризующих работу системы в целом. Среда должна позволять наблюдать изменение их значений в динамике, строить графики, файлы отчетов и т. п. Таким образом, на анализ эксперимента не будет тратиться дополнительное вычислительное время.

Для анализа таких параметров должен быть предусмотрен пятый тип компонентов – анализатор (*analyzer*) .

Реализация метрики в виде отдельного компонента позволяет использовать нестандартные системы координат, как, например, обобщенные координаты [**Error! Bookmark not defined.**, 91].

Выводы по главе 1

1. Введен термин "клеточный автомат", а также сопутствующие понятия и математический аппарат.
2. Сформулированы требования к инструментальным средствам автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов.
3. Показано, что ни одно из рассмотренных средств указанного класса не удовлетворяет даже более чем трем из семи сформулированных требований.

4. Предложена компонентная модель декомпозиции клеточных автоматов для удобства их реализации.
5. Введены основные понятия и описаны базовые свойства разработанного инструментального средства, удовлетворяющего всем требованиям.

Глава 2. Метод введения обобщенных координат для решеток клеточных автоматов

В настоящей главе разрабатывается метод введения обобщенных координат для решеток клеточных автоматов, позволяющий обеспечить универсальный подход к хранению информации о состояниях клеток для различных решеток. Обобщенная координата – неотрицательное целое число, однозначно указывающее на клетку решетки клеточного автомата произвольной размерности.

Предлагаемые координаты имеют ряд важных свойств. Введение обобщенных координат фактически представляет собой нумерацию неотрицательными целыми числами элементов многомерного дискретного пространства. Метод иллюстрируется набором примеров для двумерных клеточных автоматов, для которых вводятся:

- спиральные обобщенные координаты для решеток из треугольников;
- спиральные обобщенные координаты для решеток из квадратов;
- спиральные обобщенные координаты для решеток из шестиугольников;
- координаты для решеток из треугольников, базирующиеся на спиральных обобщенных координатах для решеток из шестиугольников;
- обобщенные координаты для решеток из квадратов, основанные на кривых Пеано (*Peano*).

Возможность введения этих и других разновидностей координат поддерживается разработанным в диссертации инструментальным средством. Обсуждаемые в настоящей главе примеры введения обобщенных координат были опробованы при приведении вычислительных экспериментов.

2.1. Основы метода введения обобщенных координат для решеток клеточных автоматов

На рис. 2–7 в каждой клетке указана пара декартовых координат. Идея обобщенных координат [91] состоит в том, чтобы пронумеровать все клетки решетки неотрицательными целыми числами так, чтобы для каждого номера существовала клетка, и для каждой клетки существовал номер, с точностью до ограниченности размеров решетки. Эти номера будут называться обобщенными координатами клеток.

Одной из клеток присваивается номер ноль, она принимается за начало координат. Номера остальных клеток характеризуют их положение относительно нее.

В общем случае введение обобщенных координат заключается в построении взаимнооднозначного отображения из множества неотрицательных целых чисел во множество целых чисел в степени k. В настоящей главе будут рассмотрены различные отображения для k=2, однако предлагаемые методы могут быть применены и для решеток большей размерности.

Для того чтобы при совершении итераций не пришлось преобразовывать обобщенные координаты в декартовы, и обратно, необходим математический аппарат, позволяющий для каждой клетки находить всех ее соседей, не переходя в другую систему координат.

В силу локальности взаимодействий для этого достаточно обеспечить нахождение лишь непосредственных соседей. В случае более сложной окрестности, если она описана в терминах относительных смещений, ее элементы можно будет найти при помощи рекурсии.

Дальнейшее обсуждение посвящено тому, как обеспечить нахождение непосредственных соседей при использовании определенных способов введения обобщенных координат для всех трех возможных двумерных решеток из правильных многоугольников. Первой будет рассмотрена квадратная решетка, как обладающая простейшим кольцом и наиболее часто используемая на практике. Далее будут рассмотрены шестиугольная и треугольная решетки. Затем будет предложен еще один метод введения обобщенных координат для треугольной решетки, обеспечивающий более эффективное нахождение непосредственных соседей клетки. В заключении будет предложено ввести координаты на решетке из квадратов сообразно кривой Пеано.

Тот факт, что согласно предложенному в предыдущей главе методу компонентной декомпозиции автоматов, в рассматриваемом инструментальном средстве метрика выделяется в отдельный, независимый компонент, позволяет использовать нестандартные системы координат в среде *CAME&L*. Обсуждаемые далее примеры введения обобщенных координат были реализованы в ней.

2.2. Спиральные обобщенные координаты для квадратной решетки

Введем так называемые "спиральные" обобщенные координаты [91] для квадратной решетки. Присвоим некоторой, клетке координату ноль. Далее пронумеруем элементы ее первого кольца по часовой стрелке, начиная с левого-верхнего угла. Затем пронумеруем также клетки второго кольца, третьего и т.д. Результат показан на рис. 8.

Смежные кольца выделены белым или светло-серым цветами, однако поверх этого выделения темно-серым цветом показаны углы колец.

Сопоставим каждому из четырех углов индекс от нуля до трех по часовой стрелке, начиная с левого-верхнего угла. Индексы также приведены на рисунке.

Линия, соединяющая клетки решетки в порядке, определенном данной нумерацией, представляет собой спираль. Поэтому данные обобщенные координаты и названы "спиральными".

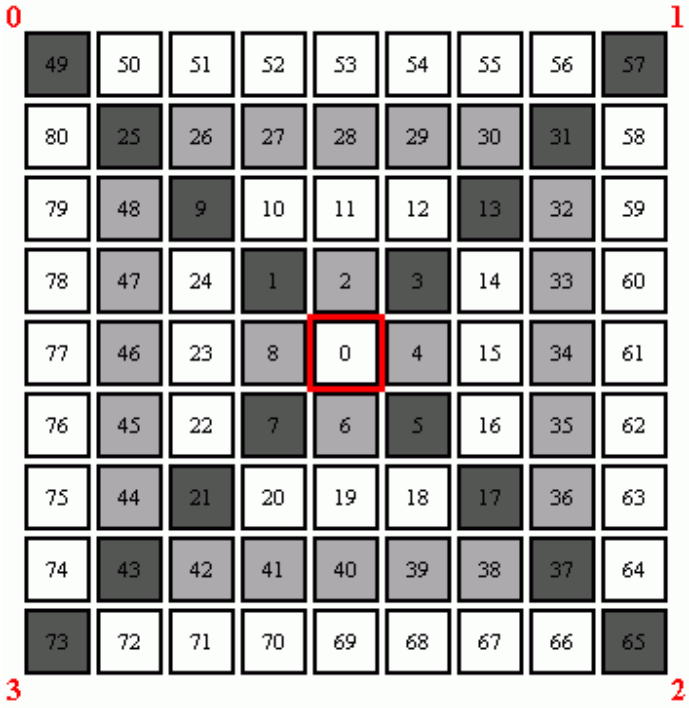


Рис. 8. Спиральные обобщенные координаты для квадратной решетки

Прежде чем предложить метод нахождения непосредственных соседей любой клетки при данном способе введения координат необходимо определить несколько вспомогательных функций.

Первая из них – длина стороны n-го кольца a_n, количество клеток от одного угла до следующего, включая сами углы. Так как a_0=1, а длина стороны каждого следующего кольца больше, чем предыдущего на две клетки, то получим

$$a_n = 2n + 1 \quad (5)$$

Вторая функция, $C_{n,m}$ – координата угла с индексом m для n-го кольца, где m принимает значения от нуля до трех. Ясно, что

$$C_{n,0} = a_{n-1}^2 = (2n-1)^2 \quad (6)$$

Координаты остальных трех углов могут быть найдены прибавлениями величины a_n–1 к координате предыдущего. В результате имеем:

$$C_{n,m} = a_{n-1}^2 + m \cdot (a_n - 1) = (2n-1)^2 + 2m \cdot n \quad (7)$$

Необходимо отметить, что формула (7), как многие другие выводимые далее выражения, не работает при n=0. Однако, это не существенно, так как единственная клетка нулевого кольца (начало координат) будет рассматриваться отдельно.

Третьей необходимой величиной является функция $n(a)$ – номер кольца клетки ноль, которому принадлежит клетка a. Выражение для нее получается, как решение уравнения $a=C_{n(a),0}$ относительно $n(a)$, в целых числах. Из формулы (6) имеем[6]:

$$n(a) = \left\lceil \sqrt{a+1} \right\rceil \text{div } 2 \quad (8)$$

Переходя к методу определения непосредственных соседей, необходимо ввести их индексацию: с левого-верхнего угла по часовой стрелке, сначала – по главным соседям, потом – по остальным. Результат приведен на рис. 9.

[6] Операция a div b обозначает целую часть от деления a на b.

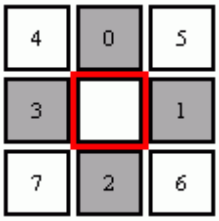


Рис. 9. Индексы непосредственных соседей для квадратной решетки

Через $N_m(a)$ будем обозначать соседа клетки a с индексом m. Достаточно обеспечить нахождение главных соседей, так как остальные могут быть определены из соотношений (9):

$$\begin{aligned} N_4(a) &= N_0(N_1(a)) = N_1(N_0(a)) \\ N_5(a) &= N_1(N_2(a)) = N_2(N_1(a)) \\ N_6(a) &= N_2(N_3(a)) = N_3(N_2(a)) \\ N_7(a) &= N_3(N_0(a)) = N_0(N_3(a)) \end{aligned} . \qquad (9)$$

Вычисление координат соседей клетки начинается с определения номера кольца, которому она принадлежит, по формуле (8). Для данной клетки a этот номер $n=n(a)$. Далее требуется рассмотреть девять вариантов расположения клетки внутри кольца. Необходимо отметить, что диапазон значений координат на n-ом кольце – от $C_{n,0}$ до $C_{n+1,0}-1$.

В таблице 2 приводятся формулы для нахождения четырех главных соседей клетки (соседей с индексами от нуля до трех) в зависимости от ее расположения. В столбце "расположение" указывается либо конкретная координата клетки, либо интервал, в который она может попасть в данном кольце.

Таблица 2. Функции определения главных соседей для спиральных обобщенных координат на квадратной решетке

Расположение	**Индекс главного соседа**			
	0	**1**	**2**	**3**
$C_{n,0}$	$C_{n+1,0}$+1	a+1	$C_{n+1,0}$–1	$C_{n+2,0}$–1
($C_{n,0}$; $C_{n,1}$)	$C_{n+1,0}$+1+a–$C_{n,0}$	a+1	$C_{n-1,0}$–1+a–$C_{n,0}$	a–1
$C_{n,1}$	$C_{n+1,1}$–1	$C_{n+1,1}$+1	a+1	a–1
($C_{n,1}$; $C_{n,2}$)	a–1	$C_{n+1,1}$+1+a–$C_{n,1}$	a+1	$C_{n-1,1}$–1+a–$C_{n,1}$
$C_{n,2}$	a–1	$C_{n+1,2}$–1	$C_{n+1,2}$+1	a+1
($C_{n,2}$; $C_{n,3}$)	$C_{n-1,2}$–1+a–$C_{n,2}$	a–1	$C_{n+1,2}$+1+a–$C_{n,2}$	a+1
$C_{n,3}$	a+1	a–1	$C_{n+1,3}$–1	$C_{n+1,3}$+1
($C_{n,3}$; $C_{n+1,0}$–1)	a+1	$C_{n-1,3}$–1+a–$C_{n,3}$	a–1	$C_{n+1,3}$+1+a–$C_{n,3}$
$C_{n+1,0}$–1	$C_{n,0}$	$C_{n-1,0}$	a–1	$C_{n+2,0}$–2

Отдельного рассмотрения требует клетка ноль, так как она является всеми четырьмя углами нулевого кольца и ее "расположение" определить невозможно. Поэтому должны быть особо определены восемь случаев – четыре соседа нуля и ноль, как сосед четырех клеток: $N_0(0)=2$, $N_1(0)=4$, $N_2(0)=6$, $N_3(0)=8$, $N_2(2)=0$, $N_3(4)=0$, $N_0(6)=0$, $N_1(8)=0$.

Использование формул (9) приведет к рекурсивным вызовам функций при реализации предлагаемого метода, однако, если не воспользоваться ими, то число возможных "расположений" клеток существенно увеличится, так как для каждого кольца необходимо будет отдельно рассматривать не только угловые клетки, но и смежные с ними.

2.3. Спиральные обобщенные координаты для шестиугольной решетки

К шестиугольной решетке применим тот же подход, что и к квадратной. Некоторой клетке присваивается координата ноль. Далее, начиная с левого-верхнего угла, по часовой стрелке последовательно нумеруются клетки ее колец. Результат показан на рис. 10.

Смежные кольца выделены белым или светло-серым цветами. Темно-серым цветом показаны углы колец. Каждому из шести углов сопоставлен индекс от нуля до пяти по часовой стрелке, начиная с левого-верхнего угла.

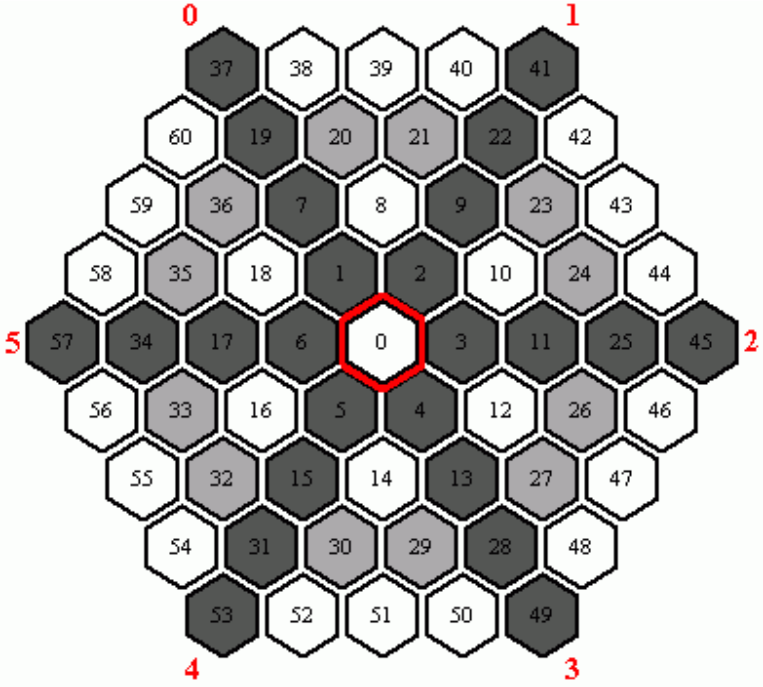


Рис. 10. Спиральные обобщенные координаты для шестиугольной решетки

Необходимо определить для шестиугольной решетки те же вспомогательные функции, которые были введены выше для квадратной решетки.

Первая функция – длина стороны n-го кольца a_n. Так как a_0=1, а длина стороны каждого следующего кольца на одну клетку больше, чем предыдущего, то

$$a_n = n + 1. \tag{10}$$

Вторая функция, $C_{n,m}$ – координата угла с индексом m для n-го кольца, где m принимает значения от нуля до пяти. Из формулы (10) следует, что общая длина каждого следующего кольца на шесть клеток больше, чем предыдущего. Принимая во внимание тот факт, что длина первого кольца – шесть клеток, получим:

$$C_{n,0} = 6E(n-1) + 1, \tag{11}$$

где

$$E(x) = 1 + 2 + \ldots + (x-1) + x = \sum_{i=1}^{x} i = \frac{x^2 + x}{2}. \tag{12}$$

Иными словами, $E(x)$ – сумма x первых натуральных чисел.

Остальные углы могут быть найдены прибавлениями величины a_n–1 к координате предыдущего. В результате выражение для угла будет иметь вид:

$$C_{n,m} = 6E(n-1) + 1 + m \cdot (a_n - 1) = 3(n^2 - n) + 1 + m \cdot n. \tag{13}$$

Третьей функцией является функция $n(a)$. Выражение для нее получается, как решение уравнения $\lceil a/6 \rceil = E(n(a))$ в целых числах относительно $n(a)$. В результате имеем:

$$n(a) = \left\lceil \frac{\sqrt{1 + 8\lceil a/6 \rceil} - 1}{2} \right\rceil. \tag{14}$$

Переходя к методу определения непосредственных соседей, введем их индексацию с левого-верхнего угла по часовой стрелке. Результат приведен на рис. 11.

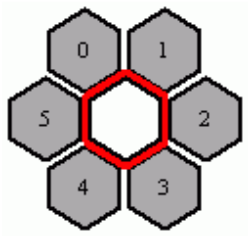


Рис. 11. Индексы непосредственных соседей для шестиугольной решетки

Как и в предыдущем случае, через $N_m(a)$ будем обозначать соседа клетки a с индексом m. В шестиугольной решетке все непосредственные соседи – главные и во избежание неоправданного использования рекурсивных функций приведем формулы для нахождения каждого из них.

Вычисление координат соседей клетки начинается с определения номера кольца, которому она принадлежит, по формуле (14). Для данной клетки a этот номер $n=n(a)$. Далее необходимо рассмотреть тринадцать вариантов расположения клетки внутри кольца.

В таблице 3 приводятся формулы для нахождения главных соседей клетки в зависимости от ее расположения.

Таблица 3. Функции определения главных соседей для спиральных обобщенных координат на шестиугольной решетке

Расположение	**Индекс главного соседа**					
	0	**1**	**2**	**3**	**4**	**5**
$C_{n,0}$	$C_{n+1,0}$	$C_{n+1,0}+1$	a+1	$C_{n-1,0}$	$C_{n+1,0}-1$	$C_{n+2,0}-1$
$(C_{n,0}; C_{n,1})$	$C_{n+1,0}+a-C_{n,0}$	$C_{n+1,0}+a-C_{n,0}+1$	a+1	$C_{n-1,0}+a-C_{n,0}$	$C_{n-1,0}+a-C_{n,0}-1$	a–1
$C_{n,1}$	$C_{n+1,1}-1$	$C_{n+1,1}$	$C_{n+1,1}+1$	a+1	$C_{n-1,1}$	a–1
$(C_{n,1}; C_{n,2})$	a–1	$C_{n+1,1}+a-C_{n,1}$	$C_{n+1,1}+a-C_{n,1}+1$	a+1	$C_{n-1,1}+a-C_{n,1}$	$C_{n-1,1}+a-C_{n,1}-1$
$C_{n,2}$	a–1	$C_{n+1,2}-1$	$C_{n+1,2}$	$C_{n+1,2}+1$	a+1	$C_{n-1,2}$

$(C_{n,2}; C_{n,3})$	$C_{n-1,2}$+a– $C_{n,2}$–1	a–1	$C_{n+1,2}$+a– $C_{n,2}$	$C_{n+1,2}$+a– $C_{n,2}$+1	a+1	$C_{n-1,2}$+a– $C_{n,2}$
$C_{n,3}$	$C_{n-1,3}$	a–1	$C_{n+1,3}$–1	$C_{n+1,3}$	$C_{n+1,3}$+1	a+1
$(C_{n,3}; C_{n,4})$	$C_{n-1,3}$+a– $C_{n,3}$	$C_{n-1,3}$+a– $C_{n,3}$–1	a–1	$C_{n+1,3}$+a– $C_{n,3}$	$C_{n+1,3}$+a– $C_{n,3}$+1	a+1
$C_{n,4}$	a+1	$C_{n-1,4}$	a–1	$C_{n+1,4}$–1	$C_{n+1,4}$	$C_{n+1,4}$+1
$(C_{n,4}; C_{n,5})$	a+1	$C_{n-1,4}$+a– $C_{n,4}$	$C_{n-1,4}$+a– $C_{n,4}$–1	a–1	$C_{n+1,4}$+a– $C_{n,4}$	$C_{n+1,4}$+a– $C_{n,4}$+1
$C_{n,5}$	$C_{n+1,5}$+1	a+1	$C_{n-1,5}$	a–1	$C_{n+1,5}$–1	$C_{n+1,5}$
$(C_{n,5}; C_{n+1,0}–1)$	$C_{n+1,5}$+a– $C_{n,5}$+1	a+1	$C_{n-1,5}$+a– $C_{n,5}$	$C_{n-1,5}$+a– $C_{n,5}$–1	a–1	$C_{n+1,5}$+a– $C_{n,5}$
$C_{n+1,0}$–1	$C_{n+2,0}$–1	$C_{n,0}$	$C_{n-1,0}$	$C_{n,0}$–1	a–1	$C_{n+2,0}$–2

Как и для квадратной решетки, отдельного рассмотрения требует клетка ноль, так как она является всеми шестью углами нулевого кольца и ее "расположение" определить невозможно. В силу этого должны быть особо определены двенадцать случаев – шесть соседей нуля и ноль, как сосед шести клеток: $N_0(0)=1$, $N_1(0)=2$, $N_2(0)=3$, $N_3(0)=4$, $N_4(0)=5$, $N_5(0)=6$, $N_3(1)=0$, $N_4(2)=0$, $N_5(3)=0$, $N_0(4)=0$, $N_1(5)=0$, $N_2(6)=0$.

2.4. Спиральные обобщенные координаты для треугольной решетки

Для треугольной решетки также можно ввести спиральные обобщенные координаты. Некоторой клетке присваивается координата ноль. Далее, начиная с левого-верхнего угла, последовательно по часовой стрелке нумеруются клетки ее колец. Результат показан на рис. 12.

Смежные кольца выделены белым или светло-серым цветами. Темно-серым цветом показаны углы колец. Сопоставим каждому из шести углов индекс от нуля до пяти, по часовой стрелке, начиная с левого-верхнего угла.

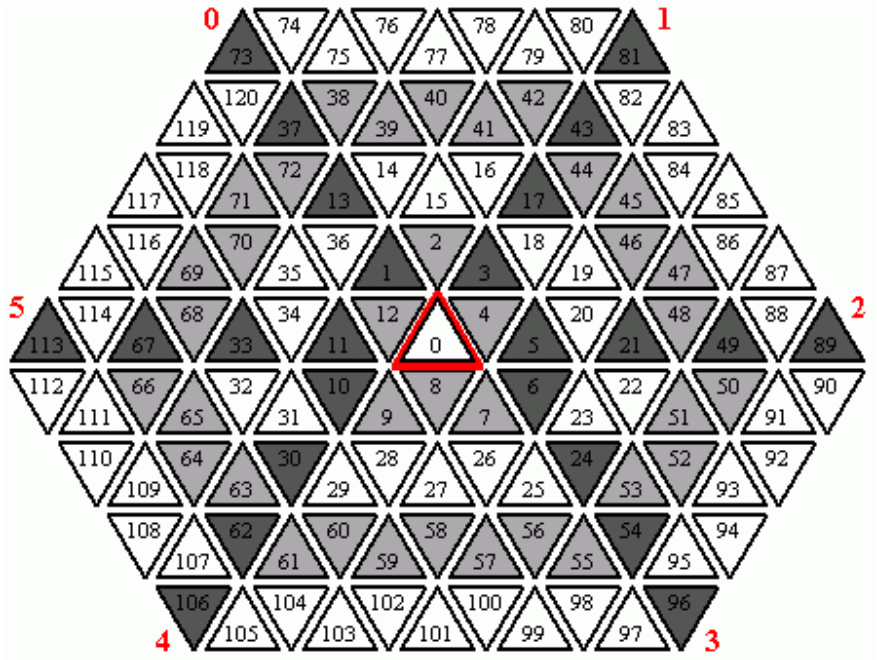


Рис. 12. Спиральные обобщенные координаты для треугольной решетки

Необходимо определить для треугольной решетки вспомогательные функции, описанные выше.

Длина стороны n-го кольца a_n для решетки треугольников не может быть введена так просто, как для рассмотренных выше решеток. У каждого кольца шесть сторон и длины некоторых из них различаются. В силу этого необходимо задать длину ребра n-го кольца, как $a_{n,m}$, где m – индекс угла из которого "выходит" сторона при обходе по часовой стрелке. Принимая во внимание длины ребер первого кольца, и учитывая, что они увеличиваются на две клетки при переходе к каждому следующему кольцу, получим:

$$\begin{cases} a_{1,0} = 3 \\ a_{1,1} = 3 \\ a_{1,2} = 2 \\ a_{1,3} = 5 \\ a_{1,4} = 2 \\ a_{1,5} = 3 \end{cases} \Rightarrow \begin{cases} a_{n,0} = 2n+1 \\ a_{n,1} = 2n+1 \\ a_{n,2} = 2n \\ a_{n,3} = 2n+3 \cdot \\ a_{n,4} = 2n \\ a_{n,5} = 2n+1 \end{cases} \tag{16}$$

Последние формулы не верны для нулевого кольца, однако это не столь важно, ведь, как неоднократно отмечалось выше, клетка ноль всегда рассматривается отдельно, как особый случай.

Из формул (16) следует, что общая длина каждого следующего кольца больше по сравнению с предыдущим на двенадцать клеток. Принимая во внимание тот факт, что длина первого кольца – двенадцать клеток, получим следующее выражение для вычисления $C_{n,0}$, координаты нулевого угла n-го кольца.

$$C_{n,0} = 12E(n-1)+1, \tag{17}$$

где $E(x)$ определяется по формуле (12).

Координаты остальных углов $C_{n,m}$ могут быть найдены последовательными прибавлениями величины $a_{n,m-1}-1$ к координате $C_{n,m-1}$-го угла. В результате имеем следующее выражение:

$$C_{n,m} = 12E(n-1)+1+\sum_{i=0}^{m-1}(a_{n,i}-1) = 6(n^2-n)+1+2m\cdot n+\begin{cases} -1, \text{при } m=3 \\ 1, \text{при } m=4 \\ 0, \text{иначе} \end{cases}. \tag{18}$$

Третьей необходимой функцией является функция $n(a)$. Выражение для нее получается, как решение уравнения $\lceil 12/a \rceil = E(n(a))$ в целых числах относительно $n(a)$. В результате, имеем:

$$n(a) = \left\lceil \frac{\sqrt{1+8\lceil a/12 \rceil}-1}{2} \right\rceil. \tag{19}$$

В треугольной решетке встречаются клетки двух возможных типов: ориентированные "вверх" (▲) и "вниз" (▼). При использовании излагаемого метода введения обобщенных координат, все клетки с нечетными номерами и нулевая клетка будут ориентированны "вверх", а остальные – "вниз".

Переходя к методу определения непосредственных соседей, введем их индексацию так, чтобы она была удобна для обоих вариантов ориентации. Для клеток, ориентированных "вверх", пронумеруем непосредственных соседей с левого-верхнего угла по часовой стрелке сначала – по главным соседям, потом – по остальным. Индексация для клеток, ориентированных "вниз", получается из этой разворотом. Результат приведен на рис. 13.

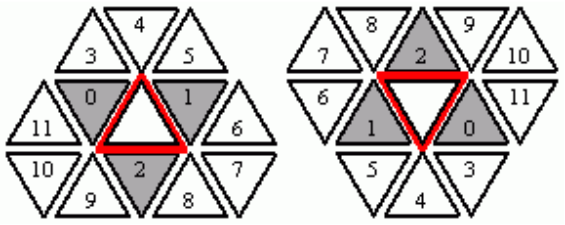


Рис. 13. Индексы непосредственных соседей для треугольной решетки

Снова через $N_m(a)$ будем обозначать соседа клетки a с индексом m. Достаточно обеспечить нахождение главных соседей, так как остальные могут быть найдены из соотношений (20):

$$
\begin{aligned}
&N_3(x) = N_2(N_0(x))\\
&N_4(x) = N_1(N_3(x)) = N_0(N_5(x)) = N_1(N_2(N_0(x))) = N_0(N_2(N_1(x)))\\
&N_5(x) = N_2(N_1(x))\\
&N_6(x) = N_0(N_1(x))\\
&N_7(x) = N_2(N_6(x)) = N_1(N_8(x)) = N_2(N_0(N_1(x))) = N_1(N_0(N_2(x)))\\
&N_8(x) = N_0(N_2(x))\\
&N_9(x) = N_1(N_2(x))\\
&N_{10}(x) = N_0(N_9(x)) = N_2(N_{11}(x)) = N_0(N_1(N_2(x))) = N_2(N_1(N_0(x)))\\
&N_{11}(x) = N_1(N_0(x))
\end{aligned}
\qquad (20)
$$

.

Вычисление координат соседей клетки начинается с определения номера кольца, которому она принадлежит, по формуле (19), а также ее ориентации.

Для данного a номер $n=n(a)$. Далее необходимо рассмотреть тринадцать вариантов расположения клетки внутри кольца (с учетом отдельного рассмотрения различных ориентаций при одном и том же расположении – девятнадцать вариантов).

Необходимо отметить, что ориентации всех углов, а также последних клеток в кольце известны заранее.

В таблице 4 приводятся формулы для нахождения непосредственных соседей клетки в зависимости от ее расположения и ориентации (▲/▼). Для углов ориентация известна заранее.

Таблица 4. Функции определения главных соседей для спиральных обобщенных координат на треугольной решетке

Расположение	**▲/▼**	**Индекс главного соседа**		
		0	**1**	**2**
$C_{n,0}$	▲	$C_{n+2,0}-1$	a+1	$C_{n+1,0}-1$
$(C_{n,0}.\ C_{n,1})$	▲	a–1	a+1	$C_{n-1,0}+a-C_{n,0}-1$
	▼	a+1	a–1	$C_{n+1,0}+a-C_{n,0}+1$
$C_{n,1}$	▲	a–1	$C_{n+1,1}+1$	a+1
$(C_{n,1}.\ C_{n,2})$	▲	a–1	$C_{n+1,1}+a-C_{n,1}+1$	a+1
	▼	a+1	$C_{n-1,1}+a-C_{n,1}-1$	a–1
$C_{n,2}$	▲	a–1	$C_{n+1,2}-1$	a+1
$(C_{n,2}.\ C_{n,3})$	▲	$C_{n-1,2}+a-C_{n,2}-1$	a–1	a+1
	▼	$C_{n+1,2}+a-C_{n,2}+1$	a+1	a–1
$C_{n,3}$	▼	$C_{n+1,3}-1$	a+1	a–1

($C_{n,3}$. $C_{n,4}$)	▲	a+1	a–1	$C_{n+1,3}$+a–$C_{n,3}$+1
	▼	a–1	a+1	$C_{n-1,3}$+a–$C_{n,3}$–1
$C_{n,4}$	▼	a–1	$C_{n+1,4}$+1	a+1
($C_{n,4}$. $C_{n,5}$)	▲	a+1	$C_{n-1,4}$+a–$C_{n,4}$–1	a–1
	▼	a–1	$C_{n+1,4}$+a–$C_{n,4}$+1	a+1
$C_{n,5}$	▲	$C_{n+1,5}$+1	a+1	a–1
($C_{n,5}$. $C_{n+1,0}$–1)	▲	$C_{n+1,5}$+a–$C_{n,5}$+1	a+1	a–1
	▼	$C_{n-1,5}$+a–$C_{n,5}$–1	a–1	a+1
$C_{n+1,0}$–1	▼	$C_{n-1,0}$	a–1	$C_{n,0}$

Как и в описанных выше методах, отдельного рассмотрения требует клетка ноль, так как она является всеми шестью углами нулевого кольца и ее расположение определить невозможно. В силу этого должны быть особо определены шесть случаев – три соседа нуля и ноль, как сосед трех клеток: $N_0(0)=12$, $N_1(0)=4$, $N_2(0)=8$, $N_0(12)=0$, $N_1(4)=0$, $N_2(8)=0$.

Использование формул (20) приводит к рекурсивным вызовам функций при реализации предлагаемого метода. В отличие от квадратной решетки, где были использованы аналогичные свойства, описываемые формулами (15), для треугольной решетки глубина рекурсии будет достигать трех при нахождении непосредственных соседей с индексами 4, 7 и 10. Однако, если не пользоваться рекурсивными формулами (20), то число "расположений" клеток, которые необходимо будет рассматривать, увеличится значительно существеннее, чем для квадратной решетки.

В связи с тем, что только четверть непосредственных соседей клетки может быть найдена без использования рекурсии, данный метод введения координат является весьма неэффективным, если требуется находить все двенадцать соседей для каждой клетки большой решетки на каждой

итерации. В следующем разделе предлагается метод введения обобщенных координат для клеточных автоматов с треугольной решеткой, который базируется на спиральных обобщенных координатах для шестиугольной решетки и предоставляет более производительных механизм нахождения непосредственных соседей.

2.5. Обобщенные координаты для треугольной решетки, базирующиеся на спиральных обобщенных координатах для шестиугольной решетки

Существенно более эффективный метод введения обобщенных координат для треугольной решетки опирается на тот факт, что шесть треугольников, расположенных определенным образом, формируют шестиугольник.

Методика введения спиральных обобщенных координат для шестиугольной решетки была рассмотрена выше. Предлагается с ее помощью пронумеровать шестерки треугольников, а затем последовательно присвоить координаты треугольникам внутри шестиугольников в порядке возрастания номеров последних. Порядок обхода треугольников внутри всех шестиугольников должен быть одинаковым, например, по часовой стрелке, начиная с левой верхней клетки. Результат такой нумерации показан на рис. 14.

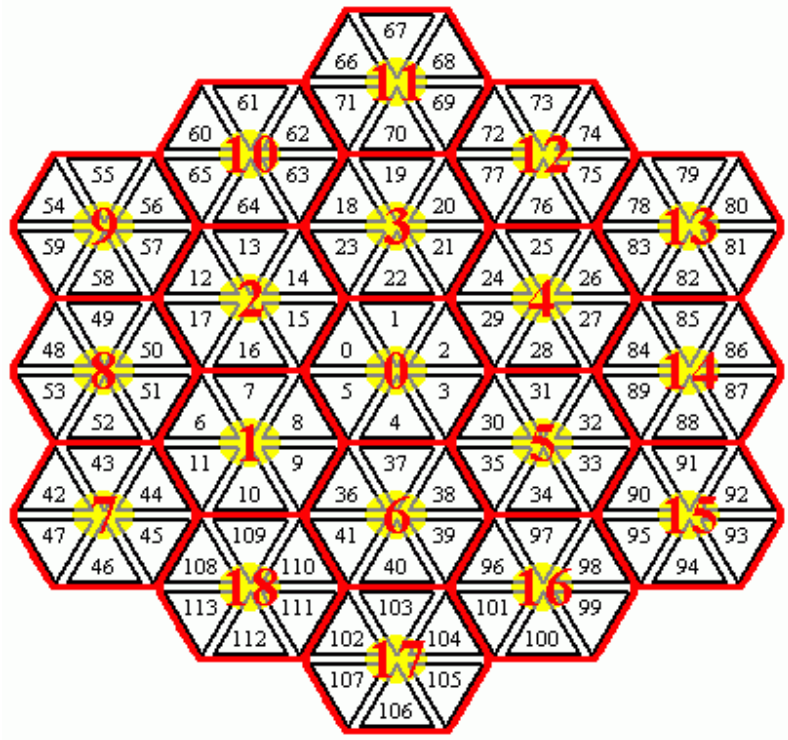


Рис. 14. Обобщенные координаты для треугольной решетки, базирующиеся на спиральных обобщенных координатах для шестиугольной решетки

На этом рисунке выделена решетка шестиугольников и в центре каждой ее ячейки крупным числом показана координата. По сравнению с решеткой, которая изображена на рис. 10, эта решетка повернута на четверть оборота против часовой стрелки. Маленькими числами на рисунке показаны координаты треугольников.

Каждому из шести возможных вариантов расположения треугольника внутри шестиугольника необходимо сопоставить индекс от нуля до пяти. Этот индекс совпадает с координатой для треугольников внутри нулевого шестиугольника на рис. 14.

Необходимо отметить, что при введении координат таким способом заранее известно, что клетки с четными номерами ориентированны "вверх", а с нечетными – "вниз". Аналогично, клетки с четными индексами внутри шестиугольника ориентированны "вверх", а с нечетными – "вниз".

Пусть для треугольника с координатой a значение $H(a)$ – координата шестиугольника, которому он принадлежит, а $h(a)$ – его индекс внутри этого шестиугольника (он принимает значения от нуля до пяти). Первая величина связана с последними двумя следующими соотношениями[7]:

$$\begin{cases} H(a) = a \operatorname{div} 6 \\ h(a) = a \bmod 6 \end{cases}, \tag{21}$$

$$a = 6H(a) + h(a). \tag{22}$$

В качестве индексов непосредственных соседей для треугольной и шестиугольной решеток будем использовать те, что были приведены на рис. 13 и 11, соответственно.

Методика нахождения соседей клетки a описывается таблицами 5 и 6. В первой из них для каждого из двенадцати возможных соседей, в зависимости от индекса $h(a)$, приведен индекс непосредственного соседа ячейки $H(a)$ в шестиугольной решетке, которому принадлежит искомый сосед треугольника. Если в соответствующей ячейке таблицы содержится знак "*", это означает, что сосед данного треугольника принадлежит тому же шестиугольнику, что и он сам.

Во второй таблице для каждого из двенадцати возможных соседей, в зависимости от индекса $h(a)$, приведен индекс соседа треугольника внутри соответствующего шестиугольника.

Таким образом, порядок действий при поиске m-го соседа клетки a должен быть следующим:

- по формулам (21) определить $h(a)$ и $H(a)$;
- выбрать в таблице 5 значение из ячейки, соответствующей m-му соседу для клетки с индексом $h(a)$. Если оно численное, то перейти к

[7] Операция a mod b обозначает остаток от деления a на b.

рассмотрению соответствующего ему соседа шестиугольника $H(a)$. Если оно – "*", то рассматривать шестиугольник $H(a)$;

- выбрать в таблице 6 значение из ячейки, соответствующей m-му соседу для клетки с индексом $h(a)$, и по формуле (22) определить координату соседа треугольника a, как клетки, принадлежащей рассматриваемому шестиугольнику и имеющей в нем индекс, полученный из последней таблицы.

Таблица 5. Индекс соседа шестиугольника, который содержит искомого соседа треугольника

h(a)	**Индекс непосредственного соседа**											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	1	*	*	1	2	2	*	*	*	0	0	1
1	*	*	2	*	*	*	1	1	2	2	3	3
2	*	3	*	2	2	3	3	4	4	*	*	*
3	4	*	*	4	5	5	*	*	*	3	3	4
4	*	*	5	*	*	*	4	4	5	5	0	0
5	*	0	*	5	5	0	0	1	1	*	*	*

Таблица 6. Индекс искомого соседа треугольника внутри объемлющего шестиугольника

h(a)	Индекс непосредственного соседа											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	3	1	5	2	5	4	2	3	4	2	1	4
1	2	0	4	3	4	5	3	2	5	3	0	5
2	1	5	3	4	3	0	4	1	0	4	5	0
3	0	4	2	5	2	1	5	0	1	5	4	1
4	5	3	1	0	1	2	0	5	2	0	3	2
5	4	2	0	1	0	3	1	4	3	1	2	3

Данный подход оказывается существенно более производительным, чем спиральные обобщенные координаты для треугольной решетки, так как поиск соседей для решетки шестиугольников не требует рекурсивных вызовов функций.

Кроме того, время, необходимое на поиск каждого из двенадцати возможных соседей, приблизительно одинаково, тогда как при использовании спиральных координат, описанных выше, поиск главных соседей осуществляется быстрее, чем остальных.

Отметим, что подобный механизм компоновки клеток в более крупные ячейки можно применять и для других пар решеток. Например, собирать треугольники в четырехугольники, формировать подобия шестиугольников из квадратов и так далее.

Кроме того, этот прием может оказаться полезным для других решеток, состоящих из клеток разных типов.

2.6. Обобщенные координаты для квадратной решетки, основанные на кривой Пеано

Кривые Пеано (*Peano*) [59] относятся к "кривым, заполняющим пространство" ("*Space-Filling Curves*"). Способность этой кривой покрыть все точки квадрата нашла свое применение во множестве отраслей науки и, в частности, численных методах (в методе конечных элементов, численном решении дифференциальных уравнений в частных производных [60, 61]), теории алгоритмов и вычислениях при помощи клеточных автоматов.

Сохраняя, как разновидность обобщенных координат, единообразие в смысле представления данных, нумерация клеток сообразно кривой Пеано, как показано на рис. 15, обеспечивает относительно плотное расположение соседей друг к другу в одномерном массиве. Однако основное их преимущество заключается, в первую очередь, в том, что они позволяют обеспечивать непрерывную нумерацию даже для иррегулярных решеток, обеспечивая уменьшение размера клетки, там, где это необходимо – в зонах, где требуется большая точность вычислений. Эта особенность кривой Пеано иллюстрируется на рис. 16

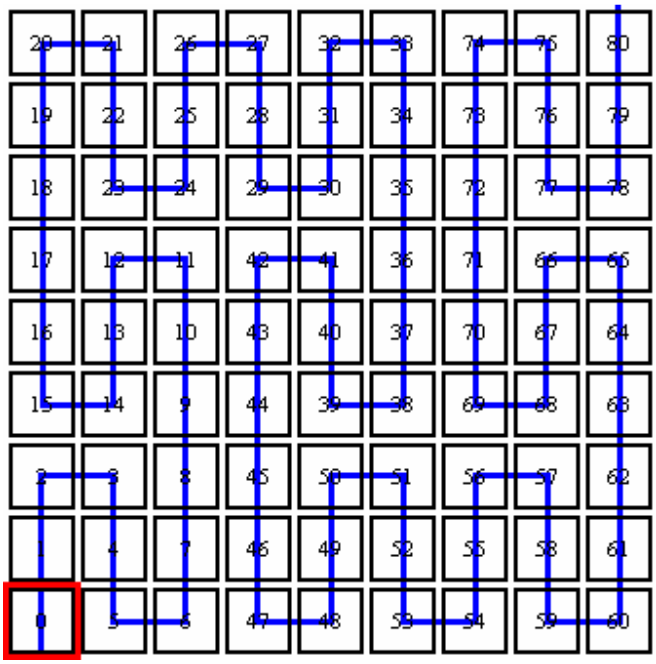


Рис. 15. Обобщенные координаты для квадратной решетки, основанные на кривой Пеано

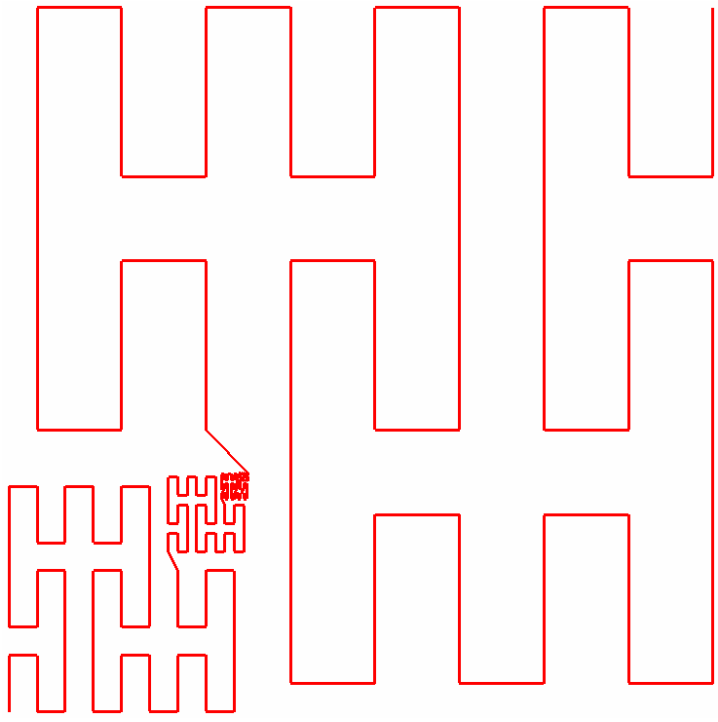


Рис. 16. Непрерывная ломанная, составленная из кривых Пеано разных порядков

Обобщенные координаты для квадратной решетки, основанные на кривой Пеано, реализованы для инструментального средства *CAME&L*, как компонент-метрика и могут быть использованы для проведения пользовательских экспериментов или реализации метрик на основе других кривых, заполняющих пространство.

2.7. Преимущества обобщенных координат. Изоморфизм двумерных решеток

Для сравнения эффективности предложенных способов введения обобщенных координат для различных решеток, они были реализованы и протестированы на разных вычислительных системах. Замерялось количество клеток, непосредственных соседей которых удавалось определить за равные промежутки времени. Результаты этого эксперимента приведены на рис. 17.

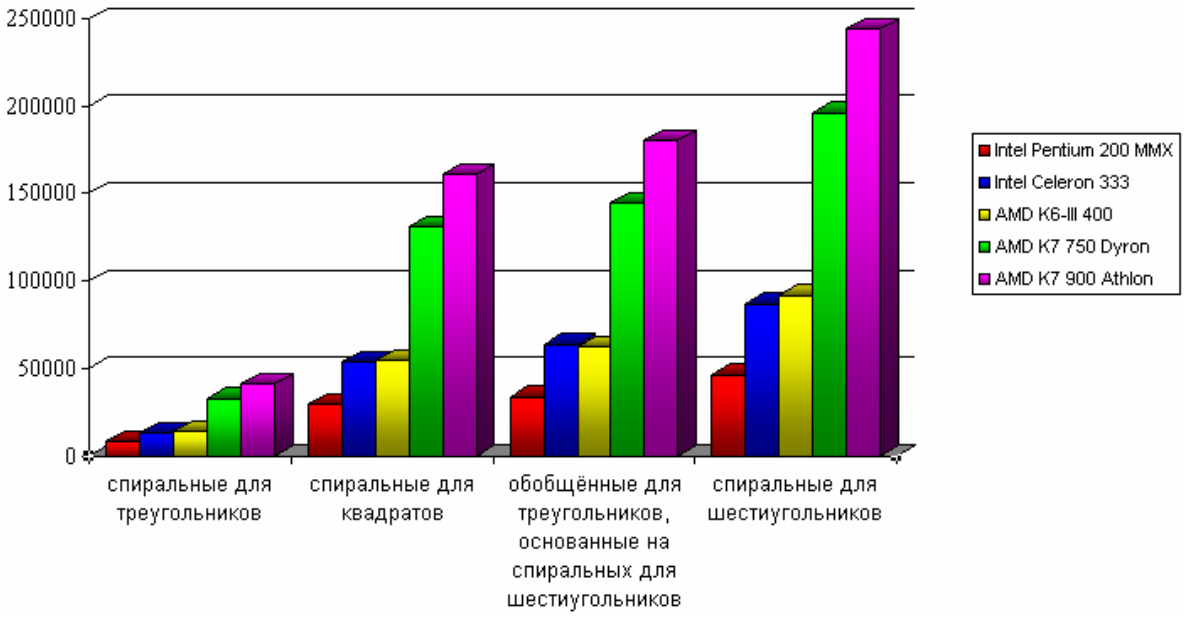


Рис. 17. Количество клеток, для которых удалось найти соседей за секунду, применяя различные однопроцессорные вычислительные системы

Приняв за единицу измерения производительности средние показатели самого медленного метода – спиральных координат для треугольной

решетки, автором экспериментально установлено, что реализация спиральных координат для квадратной решетки функционирует в 3,9 раза быстрее, реализация обобщенных координат для треугольной решетки, базирующихся на спиральных координатах для шестиугольной – в 4,4 раза быстрее, а реализация спиральных координат для шестиугольной решетки – быстрее в 6,2 раза.

Перечислим основные преимущества использования обобщенных координат:

1. Обобщенные координаты обеспечивают универсальность организации данных, позволяя хранить состояния различных решеток клеточных автоматов в одномерной структуре. При переходе от автомата с одной решеткой к автомату с другой необходимо сменить только набор функций для определения непосредственных соседей клетки, что, при реализации рассматриваемого подхода на объектно-ориентированном языке программирования может быть организовано посредством переприсваивания указателя на объекты, реализующие классы, имеющие общего предка, в котором эти функции являются виртуальными [62].
2. Существенно более удобным становится изменение размеров решетки. Расширение линейного массива и запись данных в его конец несравненно проще, чем изменение размера и перераспределение многомерной структуры, так как она тоже представляется в памяти, как линейная.

3. Одномерную структуру можно рассматривать, как последовательность данных, а последовательность данных удобно сериализовывать[8] и хранить.
4. Обобщенные координаты – это идея, которая может быть адаптирована под конкретную задачу с учетом, например, внутренней симметрии моделируемой системы и т.п. Кроме того, она может быть применена к решеткам, состоящим из других клеток или обладающим большей размерностью.
5. Отсутствие существенной зависимости от положения нуля позволяет перемещать его в соответствии с условиями задачи. Например, при моделировании может возникнуть необходимость поместить ноль в центр активности для того, чтобы оперировать там с меньшими числами. С другой стороны, можно, наоборот, отодвинуть его дальше от центра для того, чтобы основные вычисления происходили на длинных ребрах колец, где получение координат некоторых соседей выполняется за счет прибавления или вычитания единицы из координаты клетки.
6. Для некоторых решеток понятие декартовых координат неоднозначно. Например, для решетки шестиугольников неясно, почему клеткам необходимо присваивать одинаковые координаты вдоль той или иной оси, когда их центры не лежат на одной прямой параллельной этой оси. Для более "вычурных" решеток подобные проблемы еще актуальнее.

Кроме того, необходимо отметить, что клеточный автомат – модель параллельных вычислений, обладающая сколь угодно большой степенью

---

[8] "Удобно" потому, что, с точностью до языка, слова "последовательность" и "сериализовывать" – однокоренные.

параллелизма. Обобщенные координаты предоставляют метод его эмуляции с помощью конечного числа машин Тьюринга. Одна из них управляет состоянием решетки, хранимым на ее ленте, а остальные вычисляют вспомогательные функции. Точное количество машин Тьюринга, требуемое для моделирования, зависит от реализации.

Основным недостатком использования обобщенных координат является то, что скорость нахождения соседей ниже, чем при использовании декартовой метрики.

В заключение главы обратимся к первому из перечисленных выше преимуществ предложенного типа координат и поясним, в чем состоит и какой смысл перехода от автомата с одной решеткой к автомату с другой.

В подавляющем большинстве случаев для решения задач используют клеточные автоматы с квадратными решетками. Чаще всего такой выбор связан с простотой организации данных и визуализации состояния. Однако для некоторых задач треугольная и шестиугольная решетки подходят куда лучше. Тем не менее, при поверхностном рассмотрении, они "отпугивают" от себя сложностями, с которыми, якобы, связано их использование.

На рис. 18 и 19 показаны соответствия между решетками прямоугольников (не квадратов, а именно прямоугольников, так как расстояния между центрами клеток различны по разным направлениям, хотя принципиально это ничего не меняет) и треугольников, а также между решетками квадратов и шестиугольников. Эти соответствия используются при организации хранения данных, а иногда и визуализации "неквадратных" решеток. Для ясности, в клетках указаны их декартовы координаты.

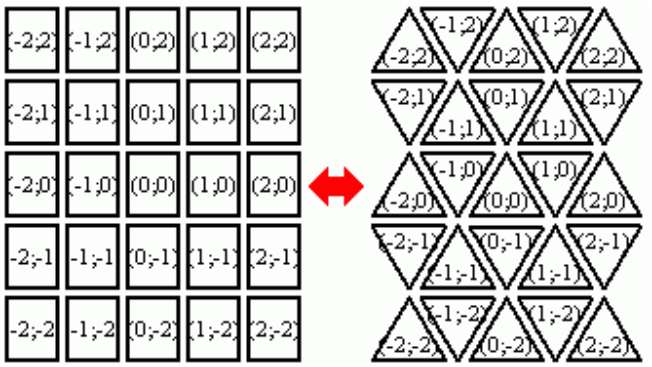


Рис. 18. Соответствие между решетками прямоугольников и треугольников

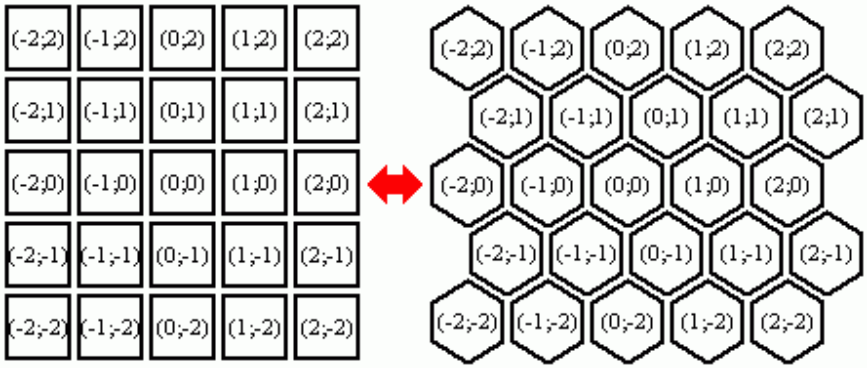
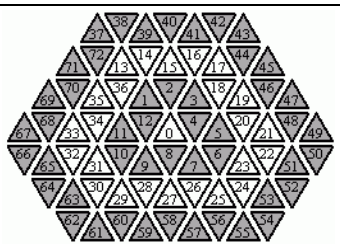
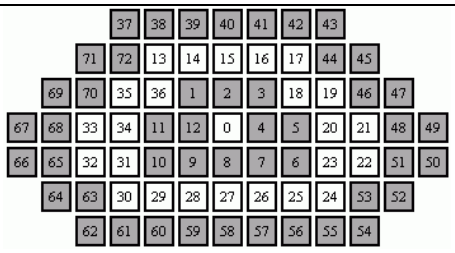
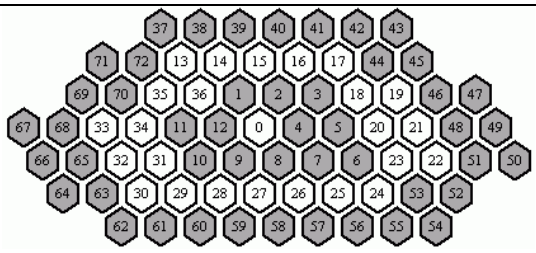
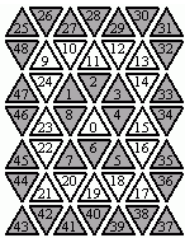
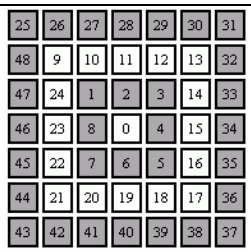
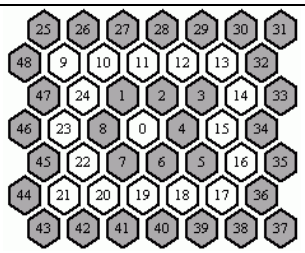
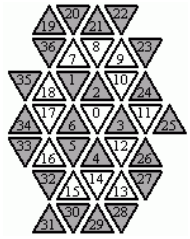
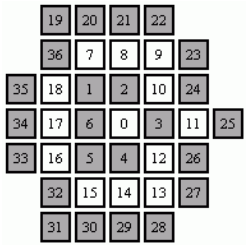
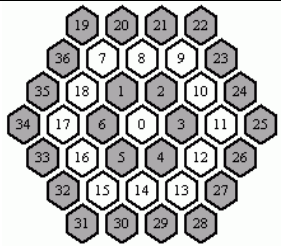


Рис. 19. Соответствие между решетками квадратов и шестиугольников

В таблице 7 показаны деформации колец при переходе от одной решетки к другой. Эта деформация происходит с учетом соответствий аналогичных показанным на рис. 18 и 19.

Таблица 7. Деформации колец при переходе от одной двумерной решетки из правильных многоугольников к другой

	Решетка		
	Треугольная	**Квадратная**	**Шестиугольная**
Треугольные координаты			

Квадратные координаты			
Шестиугольные координаты			

На рисунках в этой таблице смежные кольца выделены разными цветами. Первая строка содержит рисунки, показывающие, каким образом видоизменяются кольца треугольной решетки при переходе к другим решеткам рассматриваемого типа. Для большей наглядности во всех решетках на этой строке таблицы введены спиральные обобщенные координаты для треугольной решетки, так как для них известно, что клетки с номерами от 1 до 12 принадлежат первому кольцу, от 13 до 36 – второму и т.д.

Вторая строка содержит аналогичные рисунки для колец квадратной решетки, а третья – шестиугольной. На решетках второй строки введены спиральные координаты для квадратной решетки, а на третьей – для шестиугольной.

Диагональ из левого-верхнего угла в правый-нижний в таблице 7 содержит рисунки, демонстрирующие применение координат к решеткам, для которых они и были созданы. Эти изображения, по сути, совпадают с представленными на 12, 8 и 10, соответственно.

Основной вывод, который следует сделать из рассмотрения этой таблицы, состоит в том, что все решетки эквивалентны между собой с точностью до определения окрестности. Взаиморасположение нулевой клетки и элементов ее первого кольца наглядно показывает, какой должна быть окрестность для перехода от одной решетки к другой.

Например, из последних двух рисунков третьей строки таблицы 7 следует, что для того, чтобы выполнять итерации, описанные для клеточного автомата с шестиугольной решеткой на автомате с решеткой из квадратов, необходимо для последнего задать окрестность, состоящую из шести соседей, имеющих индексы 0, 1, 2, 3, 6 и 7 в соответствии с рис. 9. При этом всех этих соседей нужно считать главными.

То, что при этом может быть нарушено соотношение расстояний между центрами клеток, не существенно, так как это понятие ("расстояние между центрами клеток") имеет смысл только внутри функции переходов. Поэтому при сохранении этой функции смена решетки не приведет к изменению ее результирующего состояния.

Если говорить об удаленности клеток в терминах метрики решетки, как дискретного метрического пространства G, то в качестве меры расстояния между клетками a и b можно выбрать метрику, заданную формулой (3) (номер кольца клетки a, которому принадлежит клетка b, или наоборот). В результате расстояние между клетками будет сохраняться при переходе от одной решетки к другой в случае правильного выбора окрестности.

Необходимо отметить, что использование спиральных координат для шестиугольной решетки на треугольной решетке затруднено тем, что в этом случае зависимость ориентации клеток от координат неочевидна. Это иллюстрирует первый рисунок в третьей строке таблицы 7.

В таблице 8 приведены основные преимущества и недостатки каждого из рассмотренных типов решеток.

Таблица 8. Преимущества и недостатки двумерных решеток из правильных многоугольников

Решетка	Преимущества	Недостатки
Треугольная	• небольшое число главных соседей [63]; • имеет наибольшую анизотропию;	• сложность визуализации [63];
Квадратная	• простота визуализации и представления данных;	• для решения определенных задач не подходит в силу недостаточной изотропии [63];
Шестиугольная	• имеет наименьшую анизотропию [63]; • может быть "уточнена" путем деления клеток на треугольники.	• сложность визуализации [63].

Выбор той или иной решетки для решения конкретной задачи выполняется в зависимости от внутренних параметров и свойств модели, которая должна быть построена, а также от инструментальных средств доступных для решения.

Выводы по главе 2

1. Предложен метод введения обобщенных координат для решеток клеточных автоматов. Перечислены его преимущества и недостатки.
2. Продемонстрировано использование предложенного метода для двумерных решеток. Разработан математический аппарат спиральных обобщенных координатах для решеток из треугольников, квадратов и шестиугольников, а также координаты для треугольной решетки, базирующиеся на спиральных обобщенных координатах для шестиугольной решетки. Описаны обобщенные координаты для решеток из квадратов, основанные на кривой Пеано.
3. Показано, что компонентная модель декомпозиции клеточных автоматов позволяет реализовывать такие системы координат в виде компонента-метрики для инструментального средства *CAME&L*, что дает исследователю возможность легко использовать их свойствами в случае необходимости. Так, все иллюстрации настоящей работы, содержащие фрагменты решеток с введенными на них обобщенными координатами были получены при помощи инструментального средства *CAME&L*.
4. Показана эквивалентность двумерных решеток клеточных автоматов с точностью до метрики. На практике, в силу предыдущего вывода, это означает, что при использовании инструментального средства *CAME&L*, для смены решетки выполняемого автомата исследователю необходимо лишь выбрать другой компонент-метрику и, при желании изменить способ визуализации, компонент-решетку, входящие в эксперимент. Хранилище данных при этом остается неизменным и содержит все то же состояние решетки автомата.

Глава 3. Разработка инструментального средства *CAME&L* для автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов

В настоящей главе описываются важнейшие компоненты разработанного инструментального средства *CAME&L*: среду выполнения и библиотеку *CADLib* ("*Cellular Automata Development Library*" – "Библиотека для разработки клеточных автоматов").

В первом разделе описывается среда выполнения. Рассматриваются основные принципы работы в ней, назначения всех пунктов меню, формат файлов и т. д.

Второй раздел посвящен библиотеке *CADLib*. В нем приводятся описания большинства классов, функций, макроопределений и констант. Излагаются особенности программирования с использованием предложенной библиотеки. Описывается применение компонентной модели декомпозиции клеточных автоматов.

В третьем разделе описывается использование компонентного метода декомпозиции клеточных автоматов на практике. Все стандартные компоненты разрабатывались таким же образом.

3.1. Принципы проектирования предлагаемого инструментального средства

Одной из целей настоящей работы является создание универсального, расширяемого инструментального средства автоматизации проектирования программного обеспечения вычислительных экспериментов с

использованием клеточных автоматов на основе метода введения обобщённых координат, описанного в главе 2, а также компонентной модели декомпозиции клеточных автоматов, описанной в разд. 1.11. Кроме того, предлагаемое средство должно удовлетворять семи требованиям к продуктам этого типа, перечисленным в разд. 1.9.

Опишем, как обеспечивается выполнение каждого из них.

1. В состав инструментального средства входит среда выполнения экспериментов на основе клеточных автоматов, обладающая широким спектром возможностей. Интерфейс рассматриваемого инструментального средства соответствует современным требованиям. Пользователю предоставлены типичные функции, на которые он имеет основания рассчитывать: поддержка буфера обмена, отмены/повтора последней операции, печати и т.д. Среда имеет многодокументный интерфейс (*MDI*) для организации взаимодействия между экспериментами.
2. При использовании компонентной модели декомпозиции клеточных автоматов за визуализацию состояния решетки и навигацию по ней отвечает отдельный класс компонентов продукта – компоненты-решётки. В наборе стандартных компонентов, включенный в дистрибутив, имеется множество стандартных решёток, пригодных для решения большинства задач. Кроме того, любой пользователь имеет возможность расширять его, разработав свое средство, обладающее конкретными специфическими свойствами, с помощью библиотеки разработчика. Такой компонент, как и все другие, представляет собой *DLL*-библиотеку (*Dynamic-Link Library*) и будет динамически встраиваться в среду выполнения.

3. При использовании компонентной модели декомпозиции клеточных автоматов инструментальное средство *CAME&L* позволяет моделировать более широкий класс систем, чем класс "клеточные автоматы", так как выразительные способности совокупности четырех компонентов, образующих клеточный автомат, существенно шире. Универсальность и удобство переноса данных из решётки одного автомата в другой обеспечивается за счёт метода введения обобщённых координат для клеточных автоматов. Внутри инструментального средства, в любом случае клетки идентифицируются с помощью 64-битных целых чисел, обобщённых координат (даже в случае выбора декартовой метрики). Тем самым обеспечивается глобальное единообразие представления данных внутри инструментального средства, что является ключевым свойством при обеспечении универсальности.
4. Проект позволяет организовывать вычисления, как на отдельном персональном компьютере, так и на многопроцессорной системе и в вычислительном кластере. Это обеспечивается за счёт интеграции базовых функций взаимодействия вычислителей (процессоров в случае использования многопроцессорной системы или вычислительных узлов в случае использования кластерной) в базовые классы библиотеки разработчика. Множество поддерживаемых архитектур может быть расширено пользователем.
5. Средства для оптимизации вычислений также реализованы на уровне базовых классов библиотеки разработчика, включены в стандартный набор поставки программного обеспечения и могут быть использованы при решении пользовательских задач. Кроме того, набор этих средств может быть легко расширен пользователями.

6. Инструментальное средство позволяет удобно и надежно проводить продолжительные вычислительные эксперименты, сохраняя при этом интерактивность и предоставляя возможности управления его течением. За счёт разделения интерфейсных и вычислительных потоков в среде выполнения.
7. Инструментальное средство использует специальные компоненты-анализаторы для наблюдений и изучения динамики ключевых параметров эксперимента, построения графиков, файлов-отчетов и прочих задач такого рода.

В результате для обеспечения указанных свойств и характеристик было решено, что инструментальное средство автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов *CAME&L* должно быть реализовано в виде трёх составных частей:

- Среда выполнения – приложение с дружественным пользовательским интерфейсом. Базовой абстракцией в ней является понятие "эксперимент", вычислительная задача, описанная в терминах клеточных автоматов. В данном контексте это понятие – синоним термина "документ программы".

  Среда может работать на однопроцессорном компьютере, многопроцессорной системе или в вычислительном кластере . При этом продукт не требует никаких дополнительных инструментов для организации кластера, так как все необходимые возможности реализованы внутри него.

  Среда имеет многодокументный интерфейс для того, чтобы управлять несколькими экспериментами одновременно, а также реализовывать межавтоматные взаимодействия в процессе моделирования,

организовывать моделирование разных частей решётки в различных масштабах и т.п.

Помимо этого, среда обеспечивает сохранение описаний экспериментов в формате *XML* (файлы с расширением "*camel*"). Для уменьшения размера этих файлов информация о состоянии решетки автомата может быть сжата внутри документа с помощью алгоритма *BZip2*.

Для работы среды в вычислительном кластере был реализован специализированный сетевой протокол *Commands Transfer Protocol* (*CTP*).

- Библиотека для разработки клеточных автоматов *CADLib* ("*Cellular Automata Development Library*") – богатый набор *C++*-классов [62], которые служат для создания компонентов, элементов решения задач посредством программного обеспечения *CAME&L*.

  Библиотека предоставляется для использования и расширения пользователями. С точки зрения разработчика, каждый класс, реализующий компонент – потомок некого класса библиотеки *CADLib* или класса другого компонента [92].

  Иерархия классов компонентов восходит к классу `CAComponent`. От него наследуются классы, соответствующие всем пяти типам компонентов. Базовый класс решеток называется `CAGrid`, метрик – `CAMetrics`, хранилищ данных – `CADatum`, правил `CARules`, а анализаторов – `CAAnalyzer`.

  Разработка нового компонента заключается в создании потомка одного из этих классов. Потомок не обязательно должен являться непосредственным наследником одного из этих классов, так как в библиотеке имеется множество промежуточных классов,

унаследованных от них, облегчающих разработку более специфических компонентов, которые реализована часть функциональности.

Для того, чтобы быть использованным в среде, компонент, а также функции для его аутентификации, создания и удаления, помещаются в динамическую *DLL*-библиотеку.

Аналогично, иерархия классов, реализующих параметры компонентов, восходит к классу `Parameter`. Анализируемые параметры компонента, предоставляемые правилами автомата, с точки зрения реализации ничем не отличаются от обычных параметров.

Помимо классов, библиотека содержит функции, макроопределения и константы, делающие разработку компонентов настолько простой, насколько это возможно.

- Стандартные компоненты – базовый набор компонентов, из которых могут быть построены решения пользовательских задач.

Необходимо отметить, что модель компонентной декомпозиции клеточных автоматов позволяет "собирать" автоматы с различной функциональностью, посредством незначительных изменений переходить от одной задачи к другой, пробовать решать одну и ту же задачу на разных решетках, с различными метриками и т.п. Кроме того, она предоставляет возможность распределять разработку программного обеспечения эксперимента между различными исполнителями.

Каждый компонент декларирует список собственных параметров, изменение значений которых обеспечивает настройку компонента.

Компонент, реализующий правила автомата, декларирует также список анализируемых параметров, характеризующих работу системы. Среда позволяет наблюдать изменение их значений в динамике, строить графики, файлы отчетов и т. п.

В стандартный набор компонентов входит специализированный анализатор для построения графиков динамики производительности вычислений, что позволяет настраивать систему для достижения оптимальных результатов.

Реализация метрики в виде отдельного компонента позволяет использовать нестандартные системы координат, как, например, обобщенные координаты [**Error! Bookmark not defined.**, 91], что обеспечивает единообразие представления данных внутри инструментального средства.

При постановке эксперимента пользователь может воспользоваться компонентами, входящими в базовый набор или разработать свои собственные.

Структурная схема, показывающая основные составные части предложенного инструментального средства и эксперимента, выполняемого в нем, приведена на рис 20.

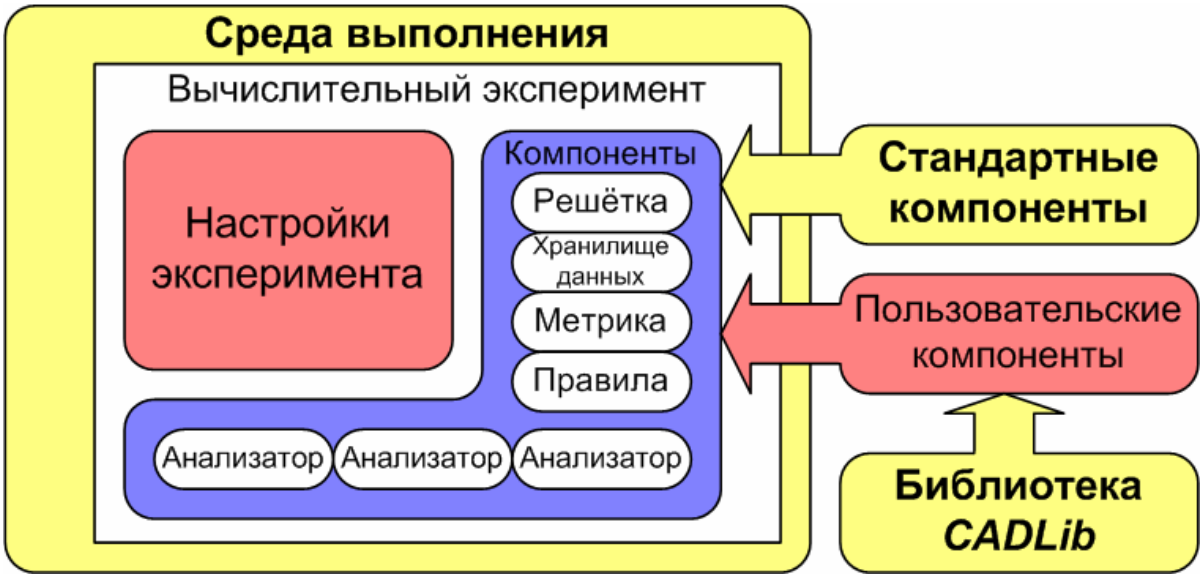


Рис. 20. Структурная схема предложенного инструментального средства

Жирным шрифтом на рисунке показаны три составные части инструментального средства *CAME&L*, перечисленные выше.

Вычислительный эксперимент, выполняемый в среде, характеризуется набором выбранных компонентов и установленными настройками. В набор компонентов может входить произвольное количество анализаторов, а также решетка хранилища данных, метрика и правила в единственном экземпляре. Компоненты выбираются из числа стандартных (поставляемых в составе инструментального средства) или пользовательских (разработанных с помощью библиотеки *CADLib*).

Предложенное средство доступно для свободного использования и опубликовано на сайте [**Error! Bookmark not defined.**].

3.2. Среда выполнения

3.2.1. Главное окно и окна документов

Среда является многодокументным приложением. Окно каждого документа служит для представления эксперимента, проводимого посредством клеточных автоматов.

На рис. 21 изображено главное окно среды с одним открытым экспериментом.

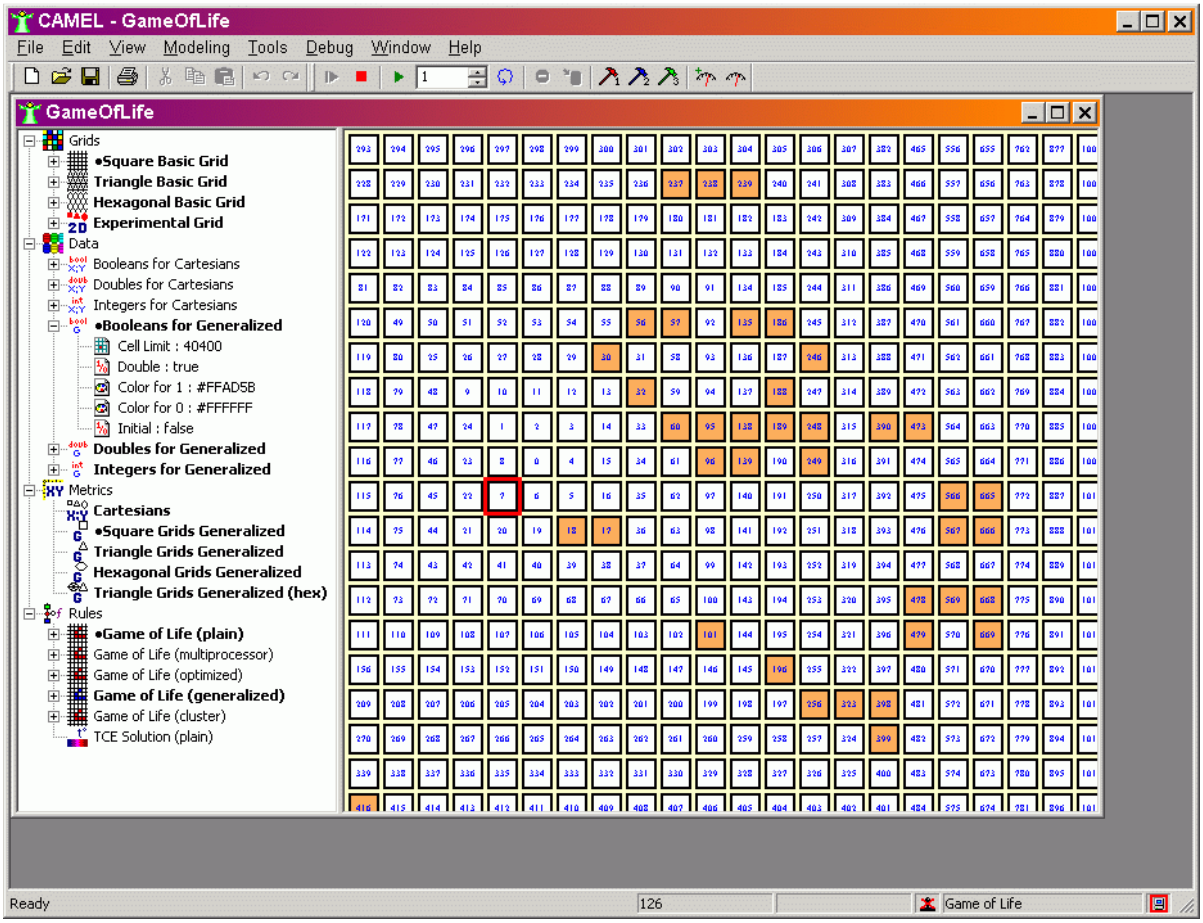


Рис. 21. Окно среды с одним открытым экспериментом

Окно эксперимента разделено на две части. В правой части находится дерево доступных компонентов. На первом уровне дерева компонентов перечислены имена четырех их типов (кроме анализаторов). На втором – имена компонентов, принадлежащих тем или иным типам. Используемые в данный момент компоненты выделены небольшим кружком в начале названия. Жирным начертанием показываются компоненты, совместимые с теми, которые выбраны в данный момент.

На третьем уровне дерева находятся параметры компонентов, изменение которых позволяет настраивать компонент.

Для изменения значения параметра необходимо щелкнуть на нем два раза. В результате появится диалог для задания нового значения. На рис. 22

изображен такой диалог, отображаемый при изменении параметра "*Double*" хранилища данных "*Booleans for Generalized*".

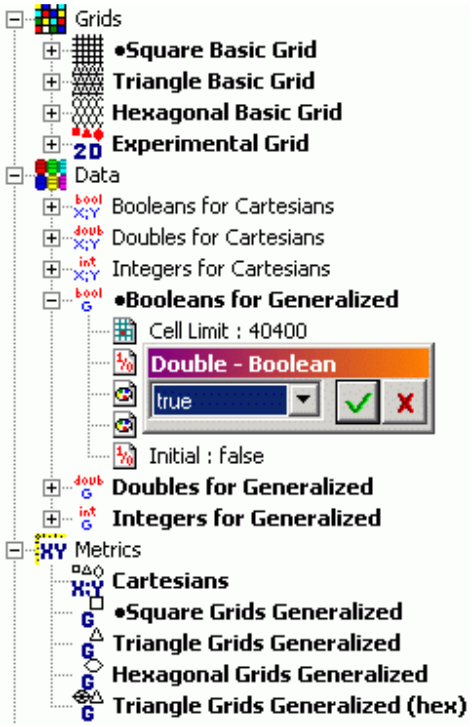


Рис. 22. Дерево доступных компонентов с открытым диалогом для изменения значения параметра

В левой части окна документа находится область визуализации решетки. Необходимо отметить, что она не будет отображаться до тех пор, пока пользователь не выберет совместимые компоненты трех типов: решетку, метрику и хранилище данных.

На рис. 23 изображено окно среды, в котором происходит один эксперимент с тремя анализаторами.

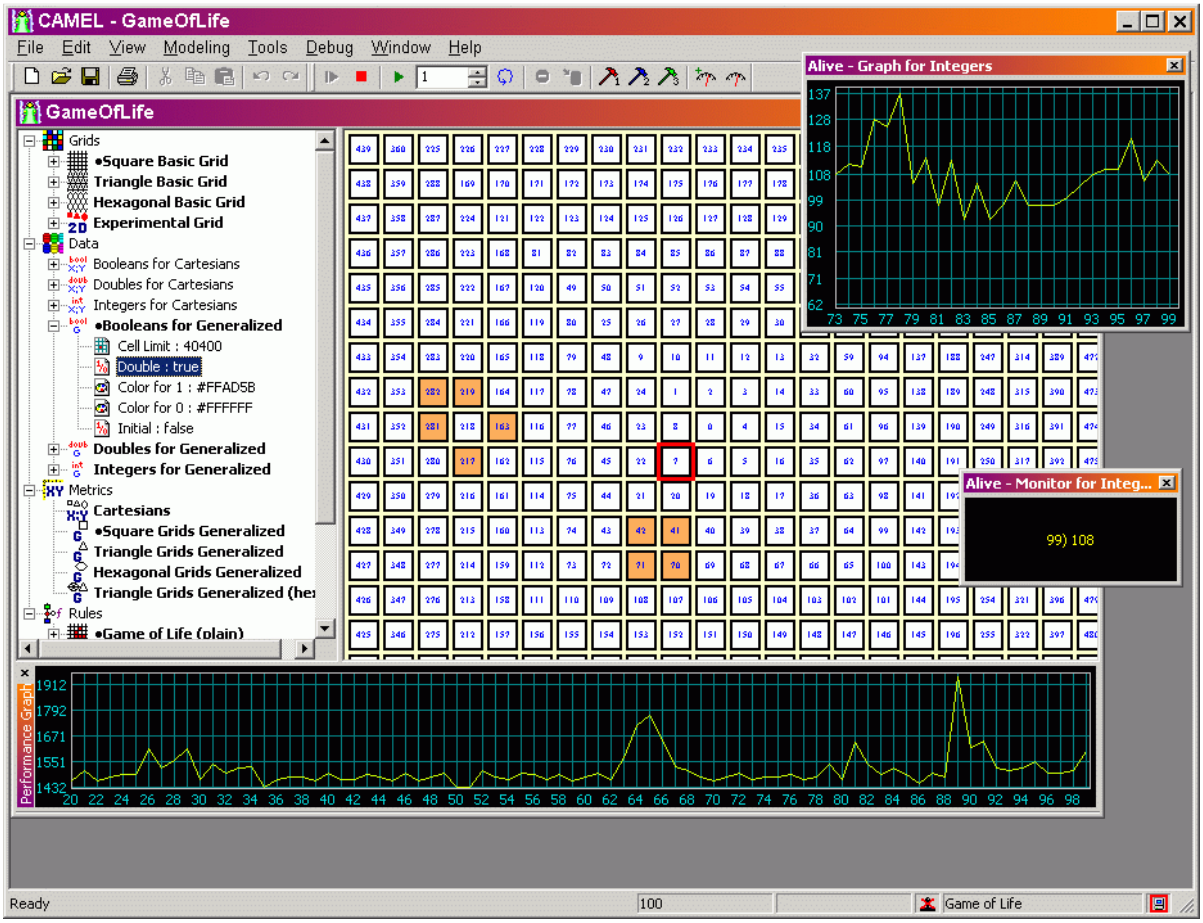


Рис. 23. Окно среды с одним открытым экспериментом и тремя анализаторами

Каждый эксперимент, для которого выбраны правила автомата, обладает именем, так называемым идентификатором правил (для документа, открытого на рисунке, это – строка "*GameOfLife*"), являющимся частью дескриптора автомата. Некая форма этого дескриптора отображается в заголовке окна. В большинстве случаев, если эксперимент не использует кластерные вычисления, представление дескриптора совпадает с идентификатором правил.

Верхняя строка основного окна программы представляет собой главное меню, описываемое ниже. Под ним располагаются панели управления, на

которые вынесены некоторые пункты меню. Внизу находится строка состояния.

Опишем справа налево поля строки состояния:

- Индикатор работы вычислительного кластера.
- Поле для вывода сообщений правилами автомата (на рисунке оно представляет собой строку "*Game of Life*").
- Индикатор перемещения курсора/решетки. Он показывает, приведет нажатие клавиш управления к перемещению курсора или визуализируемой области решетки.
- Поле для вывода координат клетки под указателем мыши.
- Поле для вывода номера итерации.
- Область для вывода информации и подсказок.

3.2.2. Главное меню

Опишем назначения всех пунктов главного меню. Необходимо отметить, что если в среде не открыт ни один документ, то далеко не все из перечисленных ниже опций (и даже меню) будут видны.

3.2.2.1. Меню "*File*"

- "*New*" – создать новый эксперимент.
- "*Open...*" – открыть файл эксперимента. Среда сохраняет документы в файлы с расширением "*camel*", представляющие собой *XML*-документы [64].
- "*Close*" – закрыть эксперимент.
- "*Save*" и "*Save As...*" – сохранить текущий эксперимент в файл.
- "*Save Visible as Picture...*" – сохранить видимую часть решетки, как картинку.

- "*Save Rendered Picture...*" – сгенерировать изображение некой области решетки и сохранить его, как картинку.
- "*Save Selected as Picture...*" – сохранить выделенную пользователем часть решетки, как картинку.
- "*Print...*" – напечатать часть решетки.
- "*Print Preview"* – предварительно просмотреть то, что может быть напечатано.
- "*Print Setup...*" – отобразить окно настройки печати.
- "*Exit*" – выйти из программы.

3.2.2.2. Меню "*Edit*"

- "*Undo*" – отменить последнее действие.
- "*Redo*" – повторить последнее действие.
- "*Cut*" – вырезать выделенное и поместить в буфер обмена.
- "*Copy*" – копировать выделенное в буфер обмена.
- "*Paste*" – вставить содержимое буфера обмена.
- "*Unselect*" – снять выделение.

3.2.2.3. Меню "*View*"

- "*Visualize Grid*" – включить/выключить визуализацию состояния решетки автомата или нет. При проведении эксперимента, после настройки рекомендуется отключить визуализацию состояния решетки автомата для повышения производительности системы.
- "*Components Tree*" – включить/выключить отображение дерева доступных компонентов.
- "*Cells Labels*" – включить/выключить отображение координат в клетках решетки.

- "*Reference Cell Must Exist*" – задать, должна ли реально существовать на решетке клетка, находящаяся в левом верхнем углу области визуализации. Если допускается использование несуществующей клетки, то появляется возможность перемещать отображаемую область за границы решетки.
- "*Cursor Position...*" – отобразить окно выбора позиции курсора.
- "*Position on Grid...*" – отобразить окно выбора позиции на решетке (запрашиваются координаты клетки, находящейся в левом верхнем углу области визуализации).
- "*Analyzers...*" – отобразить список анализаторов, выбранных для данного эксперимента.
- "*Toolbars*" – показать/скрыть те или иные панели инструментов. Доступны следующие панели:
  - "*Standard*" – стандартная панель инструментов.
  - "*Modeling*" – панель инструментов, служащая для управления экспериментами.
- "*Status Bar*" – показать/скрыть строку состояния.
- "*Information and Error Messages*" – показать/скрыть информативные сообщения и сообщения об ошибках.
- "*CTP Status Information*" – показать/скрыть информацию о состоянии механизма сетевого взаимодействия на основе протокола *CTP* (*Commands Transfer Protocol*).

3.2.2.4. Меню "*Modeling*"

- "*Start Experiment*" – начать эксперимент, выполнить инициализацию.
- "*Finish Experiment*" – окончить эксперимент, выполнить финализацию.

- "*Go*" – выполнить итерацию автомата или начать последовательность шагов, если включен режим автоматического выполнения итераций.
- "*Auto*" – включить/выключить режим автоматического выполнения итераций.
- "*Interrupt Waiting*" – прервать ожидание завершения некого процесса.
- "*Gather Data*" – собрать состояние всей решетки на данной машине, в данном документе (при использовании распределенных вычислений).
- "*Change Rules ID...*" – отобразить окно для изменения идентификатора правил автомата (имени эксперимента).
- "*First Tool Function*", "*Second Tool Function*", "*Third Tool Function*" – вызвать первую, вторую или третью функцию-инструмент, соответственно. Эти функции содержатся в компоненте правил автомата и могут быть определены пользователем.

3.2.2.5. Меню "*Tools*"

- "*Components Manager...*" – отобразить окно менеджера компонентов (рис. 24). Опишем этот инструмент.

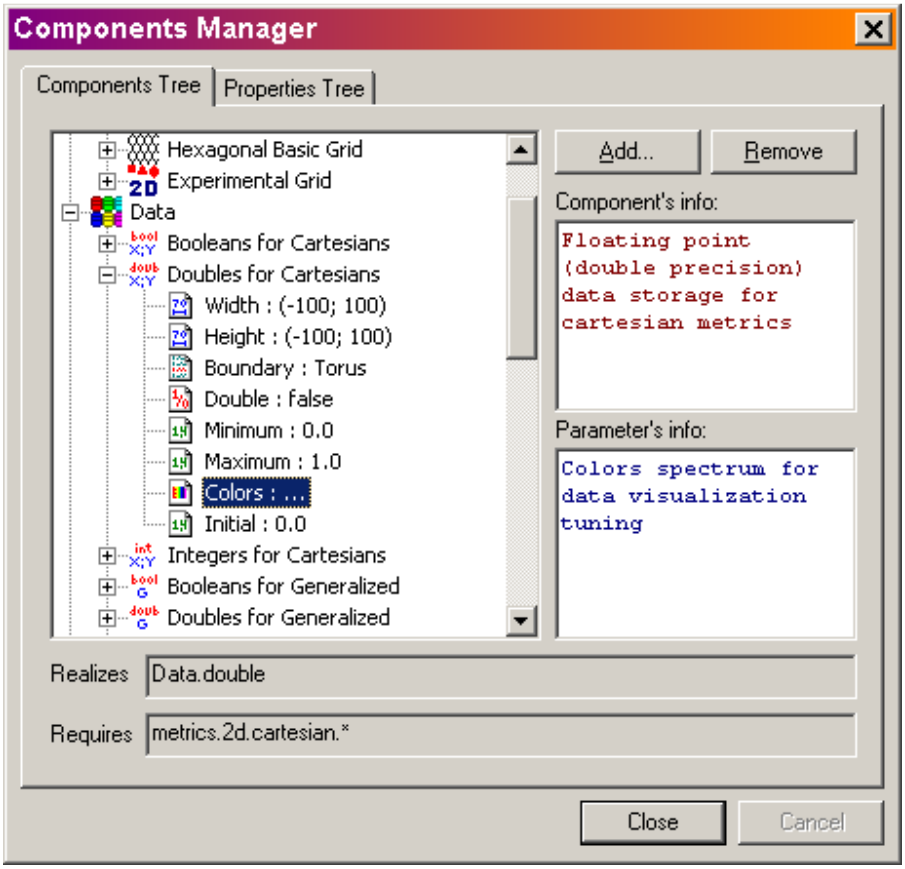


Рис. 24. Окно менеджера компонентов

Страница "*Components Tree*" содержит дерево компонентов такое же, как в любом окне документа среды. Это средство позволяет:

- просмотреть дерево;
- добавить в него компонент с помощью кнопки "*Add...*";
- удалить существующий компонент с помощью кнопки "*Remove*";
- получить описание компонента, его параметров, а также информацию о его условиях совместимости с другими. Для представления этих данных служат четыре текстовых поля.

Страница "*Properties Tree*" позволяет просмотреть дерево свойств, используемых компонентами при обмене условиями совместимости.

- "*Add Analyzer...*" – добавить анализатор эксперимента.

- "*Workstation's Characteristics...*" – отобразить окно характеристик данной рабочей станции. В частности, оно позволяет определить или задать фактор производительности ("*throughput factor*") машины.
- "*Available Automata...*" – отобразить список доступных автоматов, включая те, что выполняются на удаленных машинах.
- "*Arrange Cluster...*" – отобразить окно средств формирования кластера.
- "*Document's Options...*" – отобразить окно настройки документа.
- "*Environment's Options...*" – отобразить окно настройки среды.

3.2.2.6. Меню "*Debug*"

Данный пункт меню предоставляет доступ к операциям, которые могут оказаться полезными для отладки компонентов:

- "*Swap Datum*" – выполнить смену внутреннего хранилища данных (фактически при выборе этого пункта меню происходит вызов функции `Swap`, члена класса `CADatum`).
- "*Draw HitTest Map*" – нарисовать карту идентификации каждого пикселя визуализируемой области решетки. На ней зеленым цветом будут показаны точки, не принадлежащие клеткам или их границам, синим – точки границ клеток, а красным – самих клетки.

3.2.2.7. Меню "*Window*"

- "*New Window*" – создать новое окно для данного документа.
- "*Cascade*" – уложить открытые документы "каскадом".
- "*Tile*" – уложить открытые документы "черепицей".
- "*Arrange Icons*" – упорядочить окна документов.

3.2.2.8. Меню "*Help*"

- "*Command Line Usage...*" – отобразить информацию об опциях программы, доступных из командной строки.
- "*About CAMEL*" – отобразить информацию о программе.

3.2.3. Формат файлов документов

Camel-файл представляет собой *XML*-документ и имеет приводимую ниже структуру. Указанные без скобок атрибуты являются необходимыми. Те из них, которые приведены в квадратных скобках, являются необязательными. Атрибуты, взятые в круглые скобки, также не необходимы, однако желательны, так как их отсутствие будет проинтерпретировано существенным образом:

```
<?xml version="1.0"?>
<!-- CAMEL Document File.
  Framework Version 1.0 -->

<Document>
  ...
  <Property Name="..." Value="..."/>
  ...
  <Components>
    <Metrics Name="..." [Version="?,?,?,?"]>
      ...
      <Parameter [Name="..."] Index="?" Value="..."/>
      ...
    </Metrics>
```

```
<Datum Name="..." [Version="?,?,?,?"] (Data="...")>
  ...
  <Parameter [Name="..."] Index="?" Value="..."/>
  ...
</Datum>
<Grid Name="..." [Version="?,?,?,?"]>
  ...
  <Parameter [Name="..."] Index="?" Value="..."/>
  ...
</Grid>
<Rules Name="..." [ID="..."] [Version="?,?,?,?"]
  (Step="...") [Started="..."]>
  ...
  <Parameter [Name="..."] Index="?" Value="..."/>
  ...
  <AParameter [Name="..."] Index="?" Value="..."
    (WasSet="...")/>
  ...
</Rules>
...
<Analyzer Name="..." [AParameter="..."]
  [Position="?,?,?,?"] [Dock="..."] [Visible="..."]
  [Version="?,?,?,?"]>
  ...
  <Parameter [Name="..."] Index="?" Value="..."/>
  ...
</Analyzer>
```

```
    ...
  </Components>

  <Data (Name="...") [Format="..."] [Portion="?"]
    [Skip]>
    ...\0
  </Data>
</Document>
```

Тело документа заключается между тегами <Document> ... </Document>. Внутри документа может находиться любое количество тегов <Property/>, <Components> ... </Components> и <Data> ... </Data>.

Теги <Property/> служат для задания свойств документа (настроек среды). Пара атрибутов "Name" и "Value" определяет имя и значение свойства, соответственно.

Между тегами <Components> ... </Components> должны размещаться теги, описывающие выбранные компоненты: <Metrics> ... </Metrics>, <Datum> ... </Datum>, <Grid> ... </Grid>, <Rules> ... </Rules> и <Analyzer> ... </Analyzer>. При этом первые четыре тега могут быть использованы не более чем один раз, а пятый – произвольное число раз.

У каждого из них есть атрибуты "Name" – имя компонента и "Version" – версия библиотеки компонента в формате "?,?,?,?". Последний из них необязательный, но если он указан и версия имеющегося в

системе компонента более ранняя, то при открытии будет выведено предупреждающее сообщение.

Кроме того, у тега <Datum> ... </Datum> имеется дополнительный атрибут "Storage", указывающий имя тега <Data> ... </Data>, данные из которого следует использовать. Значение этого атрибута по умолчанию – "Default".

У тега <Rules> ... </Rules> имеется три дополнительных атрибута: "ID" – идентификатор правил, "Step" – номер шага на момент сохранения и "Started" – флаг, указывающий был ли начат (инициализирован) эксперимент или нет. Последний атрибут, как и другие булевы флаги, может принимать значения "true" или "false".

У тега <Analyzer> ... </ Analyzer> имеются четыре дополнительных атрибута: "AParameter" – имя анализируемого параметра, "Position" – положение окна анализатора (в формате "x1,y1,x2,y2"), "Dock" – к чему пристыковано окно (может принимать значения "top", "bottom", "left", "right" и "float", если не к чему) и "Visible" – флаг, определяющий видим анализатор или скрыт.

Между тегами, описывающими компоненты, могут размещаться теги <Parameter/>, содержащие значения их параметров. Они имеют два обязательных атрибута: "Index" – индекс данного параметра в списке параметров компонента и "Value" – значение параметра. Кроме того, для них предусмотрен необязательный атрибут "Name" – имя параметра. Он необязателен, так как не принимается в расчет при открытии файлов, однако может оказаться весьма информативным для человека, редактирующего *camel*-файл в текстовом редакторе.

Между тегами, описывающими компонент правил, могут также размещаться теги <AParameter/>, описывающие анализируемые параметры. Как и теги <Parameter/>, эти теги имеют два обязательных атрибута: "Index" – индекс в списке анализируемых параметров компонента правил и "Value" – значение параметра, а также необязательный атрибут "Name" – имя параметра. Кроме того, для тегов <AParameter/> предусмотрен необязательных атрибут "WasSet" – флаг, показывающий, присваивалось ли этому параметру какое-либо значение.

Тег <Data> ... </Data> предназначен для хранения состояния решетки. У него есть атрибут "Name", содержащий имя хранилища, по которому на него может ссылаться атрибут "Storage" тега <Datum> ... </Datum>. Если в качестве значения атрибута "Name" указана строка "Autoload", и тег <Datum> ... </Datum> не ссылается ни на какое другое хранилище, то будет использовано именно этот состояние решетки.

Кроме того, у тега <Data> ... </Data> существуют три необязательных параметра: "Portion" – указывает, сколько символов содержится в строке записи состояния решетки, "Format" – указывает формат, в котором хранятся данные (например, "plain" или "bzip2") и "Skip" – если указан этот атрибут тег будет пропущен при чтении файла. Необходимо отметить, что последний атрибут не является флагом и значение для него указывать не требуется.

Для описания состояния решетки между тегами могут быть использованы только символы с *ASCII*-кодами от 64 до 192. Любые другие символы игнорируются. Последним должен быть символ с нулевым кодом.

В документе может быть несколько хранилищ данных с разными именами.

Camel-файл интерпретируется и выполняется средой по мере прочтения.

3.3. Библиотека для разработки клеточных автоматов *CADLib*

3.3.1. Диаграмма классов библиотеки

Библиотека содержит набор *C++*-классов, которые служат для создания так называемых "компонентов", элементарных частей эксперимента.

Помимо классов, она также содержит функции, макроопределения и константы, которые делают разработку компонентов настолько простой, насколько это возможно.

Полная диаграмма классов библиотеки *CADLib* и стандартных компонентов приведена на рис. 25.

Параллелограммы с пунктирными границами обозначают шаблоны классов [62]. Серые фигуры показывают классы стандартных компонентов. Они распространяются вместе с исходным кодом, однако формально в библиотеку *CADLib* не входят. Остальные параллелограммы обозначают обычные классы.

Большинство классов разрабатывалось так, чтобы они не зависели от платформы на уровне исходного кода. В них используются только стандартные библиотеки и библиотека *STL* (*Standard Template Library*) [62]. Однако некоторые классы, преимущественно, имеющие отношение к пользовательскому интерфейсу, имеют связи с библиотекой *MFC* (*Microsoft Foundation Classes*) [65]. Эти классы отмечены на диаграмме знаком "*". При переносе библиотеки *CADLib* на другую платформу их необходимо изменить

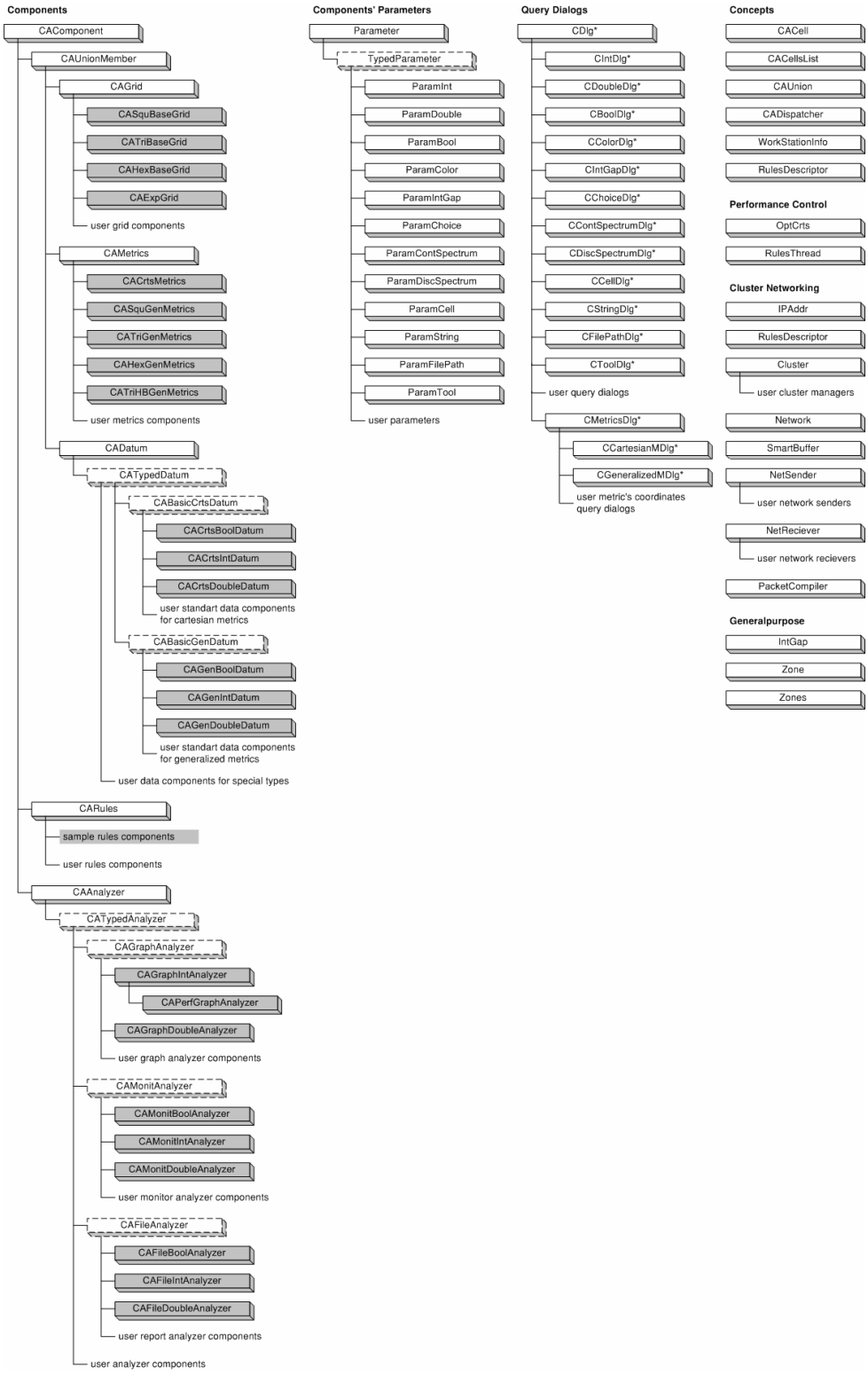


Рис. 25. Диаграмма классов библиотеки классов *CADLib*

для работы с соответствующей библиотекой, предоставляющей средства создания пользовательского интерфейса.

Кроме того, на диаграмме специально показаны места, которые могут занять пользовательские классы в зависимости от их назначения.

3.3.2. Ключевые классы

Одним из ключевых типов данных является тип `CACell`, предназначенный для представления идентификаторов клеток решетки в произвольной метрике как 64-разрядных целых чисел.

Необходимо описать также чрезвычайно востребованный класс `Zone`, описывающий некоторую область на трехмерной решетке автомата, так называемую "зону". Экземпляр класса хранит шесть значений типа `CACell`, определяющих три отрезка вдоль трех осей пространства [`a1`, `b1`], [`a2`, `b2`] и [`a3`, `b3`].

В случае необходимости описания зоны на двумерной решетке `a3` полагается равным `b3`. Для одномерного случая еще и `a2` приравнивается `b2`.

Кроме того, класс `Zone` реализует большой набор функций зональной алгебры.

3.3.3. Класс `CAComponent`

Как отмечалось выше, данный класс является базовым для всех компонентов. Опишем его члены:

- `public CAComponent()` – конструктор.
- `public ~CAComponet()` – деструктор.
- `protected int m_iType` – переменная, хранящая тип компонента. Может принимать одно из следующих значений:

- CT_COMPONENT – абстрактный тип, среда игнорирует такой компонент;
- CT_GRID – решетка;
- CT_DATUM – хранилище данных;
- CT_METRICS – метрика;
- CT_RULES – правила;
- CT_ANALYZER – анализатор.

Значение присваивается в потомках.

- `public inline UINT GetType()` – функция возвращает тип компонента.

Далее описываются функции, предоставляющие информацию о компоненте. Их можно или нужно переопределять в наследниках.

- `public virtual inline STRING GetName()` – функция возвращает имя компонента. Она должна быть переопределена в потомке. Для переопределения предусмотрен макрос `COMPONENT_NAME(<имя>)`.
- `public virtual inline STRING GetInfo()` – функция возвращает описание компонента. Она должна быть переопределена в потомке. Для переопределения предусмотрен макрос `COMPONENT_INFO(<информация>)`.
- `public virtual inline LPTSTR GetIconID()` – функция возвращает идентификатор ресурса, представляющего собой иконку данного компонента. Если она возвращает `NULL`, то среда будет использовать иконку по умолчанию. Функция может быть переопределена в потомке. По умолчанию она возвращает `NULL`. Для

переопределения предусмотрен макрос COMPONENT_ICON(<идентификатор>).

- public virtual inline STRING GetRequire() – функция возвращает выражение, описывающее набор свойств, которыми должны обладать компоненты для того, чтобы быть совместимыми с данным, так называемые "условия совместимости". Выражение принадлежит языку, определяемому следующей грамматикой:

```
<EXPRESSION> ::= (<EXPRESSION>)
<EXPRESSION> ::= !<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>^<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>|<EXPRESSION>
<EXPRESSION> ::= <EXPRESSION>&<EXPRESSION>
<EXPRESSION> ::= <TOKEN>
```

Каждая лексема <TOKEN> может представлять собой:

  - конкретное свойство, набор слов, разделенных точкой (например, "Data.bool" или "Metrics.2D.cartesian"). Каждое следующее слово уточняет предыдущее или, иными словами, является "подсвойством" предшествующего;
  - начало свойства и групповой символ "*" (например, "Metrics.2D.*"), обозначающий любое, в том числе и нулевое количество подсвойств;
  - начало свойства и групповой символ "?", обозначающий любое ненулевое количество подсвойств.

В качестве имен свойств и подсвойств могут быть использованы произвольные слова. При их проверке на совпадение регистр не учитывается.

Функция может быть переопределена в потомке, по умолчанию возвращает "*", что соответствует совместимости со всеми компонентами. Для переопределения предусмотрен макрос COMPONENT_REQUIRES(<выражение>).

- public virtual inline STRING GetRealize() – функция возвращает набор конкретных свойств, которые реализует настоящий компонент. Если свойств несколько, то они разделяются точкой с запятой. Функция может быть переопределена в потомке, по умолчанию возвращает пустую строку, что приводит к тому, что данный компонент будет совместим со всеми другими. Для переопределения предусмотрен макрос COMPONENT_REALIZES(<выражение>).

Далее описываются функции для поддержки работы с параметрами, наследниками класса Parameter. Средой обеспечивается возможность изменения их значений для настройки компонента.

- public virtual inline unsigned short GetParametersCount() – функция возвращает число параметров данного компонента. Она может быть переопределена в потомке, по умолчанию возвращает ноль.
- public virtual inline Parameter* GetParameter(unsigned short n) – функция возвращает указатель на *n*-ый параметр компонента, где *n* принимает значение от нуля до значения, которое вернет функция GetParametersCount,

уменьшенного на единицу. Функция может быть переопределена в потомке. По умолчанию она возвращает NULL.

Для удобства реализации двух последних функции предусмотрены следующие макроопределения:

- PARAMETERS_COUNT(<число_параметров>) – для задания числа параметров;
- BEGIN_PARAMETERS – для того, чтобы начать, так называемую, "карту параметров";
- PARAMETER(n,<указатель_на_n-ый_параметр>) – для добавления *n*-го параметра в карту;
- END_PARAMETERS – для того, чтобы закончить карту.

Например, если у компонента *N* параметров, которые называются p_Par1, p_Par2, ..., p_ParN, то карта параметров будет иметь вид:

```
PARAMETERS_COUNT(N)
BEGIN_PARAMETERS
    PARAMETER(0,&p_Par1)
    PARAMETER(1,&p_Par2)
            ...
    PARAMETER(N-1,&p_ParN)
END_PARAMETERS
```

- public virtual void ParameterChanged(Parameter* pPar) – функция, вызываемая средой, когда произошло изменение значения некого параметра данного компонента. При этом указатель на изменившийся параметр передается в функцию. При инициализации

компонента ей передается значение NULL. Функция может быть переопределена в потомке. По умолчанию она ничего не делает.

3.3.4. Классы CAUnion и CAUnionMember

Три компонента (решетка, метрика и хранилище данных), в силу специфики возложенных на них функций, должны функционировать слаженно, взаимодействуя друг с другом, образуя, так называемый, союз компонентов. Функциональность этого союза поддерживается на уровне их общего предка – класса CAUnionMember.

В библиотеке *CADLib* имеется класс CAUnion, обеспечивающий упомянутый выше союз компонентов. Опишем этот класс:

- public CAGrid* Grid – переменная, хранящая указатель на компонент-решетку;
- public CADatum* Datum – переменная, хранящая указатель на компонент-хранилище;
- public CAMetrics* Metrics – переменная, хранящая указатель на компонент-метрику;
- public CAUnion() – конструктор инициализирует указатели на все компоненты значением NULL;
- public inline bool IsTrioOK() – функция возвращает true, если указатели на все компоненты не равны NULL – существует союз выбранных Пользователем компонентов. Возвращает false в противном случае.

Опишем теперь класс CAUnionMember, являющийся потомком класса CAComponent. К средствам поддержки работы в союзе компонентов относятся следующие:

- `protected void* pUnion` – переменная, в которой хранится указатель на соответствующий объект класса `CAUnion`;
- `public void SetUnion(void* pUnion)` – функция устанавливает значение переменной `pUnion`.

  Кроме того, класс имеет следующие члены:
- `public CAUnionMember()` – конструктор;
- `public ~CAUnionMember()` – деструктор;
- `private bool bWasInit` – переменная, определяющая проводилась ли инициализация данного компонента. Одной из основных операций, сопутствующих инициализации, является вызов функции-члена `ParameterChanged` с параметром `NULL`. Потребность в данном поле объясняется тем, что эта операцию должна выполняться однократно;
- `public inline bool GetWasInit()` – функция возвращает значение переменной `bWasInit`. Она используется средой и не должна вызываться пользовательскими библиотеками;
- `public inline void ResetWasInit()` – функция присваивает переменной `bWasInit` значение `true`. Обратное присвоение уже невозможно. Функция используется средой и не должна вызываться пользовательскими библиотеками.

3.3.5. Класс `CAGrid`

Класс `CAGrid` является наследником класса `CAUnionMember`. Он базовый для всех классов, реализующих компоненты решеток. Компоненты данного типа отвечают за визуализацию состояния решетки. Опишем этот класс:

- `public CAGrid()` – конструктор;
- `protected CDC* pDC` – переменная для хранения контекста устройства для отображения решетки;
- `protected CSize* pszSize` – переменная для хранения размера изображения решетки в пикселях;
- `protected CSize* pszVArea` – переменная для хранения размера видимой части изображения решетки в пикселях. Ясно, что его величина не может превосходить размеров, хранящихся в переменной `pszSize`;
- `public void SetOutput(CDC* pDC, CSize* pszSize, CSize* pszVArea)` – функция устанавливает значения предыдущих переменных `pDC`, `pszSize` и `pszVArea`. Она не должна вызываться пользовательскими библиотеками, а вызывается только средой;
- `public CACell RefCell` – переменная для хранения идентификатора клетки, находящейся в левом верхнем углу в видимой области решетки, так называемой "ссылочной клетки". Именно изменение значения этой переменной обеспечивает перемещение по решетке;
- `public CACell CurCell` – переменная для хранения идентификатора клетки, на которой находится курсор;
- `public CACellsList SelCells` – переменная для хранения списка идентификаторов клеток, выделенных в настоящий момент. Класс `CACellsList` реализует список объектов класса `CACell`;
- `public bool bShowLabels` – переменная, определяющая требуется ли показывать координаты клеток (если она равна `true`) или нет (если она равна `false`);

- `public bool bRefCellMustExist` – переменная, определяющая должна ли клетка, находящаяся в левом верхнем углу области визуализации реально существовать на решетке (если значение равно `true`). Если допускается использование несуществующей клетки, то появляется возможность перемещать отображаемую область за границы решетки.

Далее описываются функции, которые должны быть переопределены в потомке:

- `public virtual void Paint(bool onlysel=false, bool hidecur=false)` – функция должна обеспечивать отображение решетки. Параметр `onlysel` определяет, требуется ли изобразить все видимые клетки (если значение параметра равно `false`) или только выделенные (если значение параметра равно `true`). Параметр `hidecur` определяет, прятать ли курсор (если `true`) или нет (если `false`).

  Детали рисования задаются значениями переменных `pDC`, `pszSize`, `pszVArea`, `RefCell`, `CurCell`, `SelCells`, `bVizualize` и `bShowLabels`, а также параметров компонента.
- `public virtual CACell HitTest(int x,int y,int& type)` – функция идентифицирует каждый пиксель визуализируемой области решетки. Для каждой точки с координатами (`x`; `y`), через ссылку `type`, функция передает одно из следующих значений и, в зависимости от него, возвращает соответствующий результат:
  - `HT_NONE` – точка соответствует промежутку между клетками. Значение, которое вернет сама функция, не будет приниматься во внимание;

- HT_FRAME – точка принадлежит границе клетки, при этом сама функция должна вернуть идентификатор этой клетки;
- HT_CELL – точка принадлежит клетке, при этом сама функция должна вернуть идентификатор этой клетки.

- `public virtual void MoveGrid(int dir, bool repaint=true)` – функция должна обеспечивать перемещение видимой области решетки в направлении, указанном параметром `dir`. Он может принимать одно из следующих значений:
  - DIR_UP – вверх;
  - DIR_DOWN – вниз;
  - DIR_LEFT – вправо;
  - DIR_RIGHT – влево;
  - DIR_PAGEUP – вверх на страницу;
  - DIR_PAGEDOWN – вниз на страницу;
  - DIR_PAGELEFT – влево на страницу;
  - DIR_PAGERIGHT – вправо на страницу.

  Перемещение обеспечивается изменением значения ссылочной клетки, идентификатор которой хранится в переменной `RefCell`.

  Если значение параметра `repaint` равно `true`, то после выполнения сдвига необходимо выполнить перерисовку решетки (вызвать функцию `Paint`).
- `public virtual void MoveCursor(int dir)` – функция должна обеспечивать сдвиг курсора в направлении, указанном параметром `dir`. Его возможные значения приведены в описании предыдущей функции. По возможности, следует реализовать эту функцию так, чтобы курсор всегда был в видимой части решетки.

Далее следуют функции, используемые при сохранении изображения фрагмента решетки, как картинки. Их можно не переопределять, но тогда эта функция не будет работать корректно.

- `virtual inline unsigned int GetPicWidth(unsigned int cells)` и `virtual inline unsigned int GetPicHeight(unsigned int cells)` – функции должны возвращать ширину и высоту картинки, изображающей `cells` клеток, стоящих в ряд, соответственно.

3.3.6. Класс `CAMetrics`

Класс `CAMetrics` является наследником класса `CAUnionMember`. Он базовый для всех классов, реализующих компоненты метрик. Данный тип компонентов отвечает за присвоение клеткам координат, обеспечение разрешения отношения соседства, а также вычисление расстояний между клетками. Опишем функции этого класс:

- `public CAMetrics()` – конструктор.

Далее следуют функции, которые могут или должны быть переопределены в потомке:

- `public virtual long GetDistance(CACell c1, CACell c2)` – функция определяет расстояние между клетками `c1` и `c2`. По умолчанию она возвращает ноль. Функция может быть переопределена в потомке. Среда не пользуется этой функцией, она реализуется для пользователя.
- `public virtual inline long GetDistanceX(CACell c1, CACell c2)` – функция определяет расстояние между клетками `c1` и

`c2` вдоль оси абсцисс. Она должна быть переопределена в потомке, так как используется средой для навигации по решетки.

- `public virtual inline long GetDistanceY(CACell c1, CACell c2)` – функция определяет расстояние между клетками `c1` и `c2` вдоль оси ординат. Она должна быть переопределена в потомке, так как используется средой для навигации по решетки.
- `public virtual inline long GetDistanceZ(CACell c1, CACell c2)` – функция определяет расстояние между клетками `c1` и `c2` вдоль оси аппликат. По умолчанию она возвращает ноль. Функция может быть переопределена в потомке, но не обязательно, так как трехмерные решетки поддерживаются в качестве дополнительной возможности, а не основной.
- Шесть следующих функций служат для определения направлений увеличения (incrementing) и уменьшения (decrementing) координат вдоль той или иной оси относительно клетки с идентификатором `c`, передаваемым в функцию в качестве параметра:
  - `virtual inline unsigned short GetXIncDir(CACell c);`
  - `virtual inline unsigned short GetXDecDir(CACell c);`
  - `virtual inline unsigned short GetYIncDir(CACell c);`
  - `virtual inline unsigned short GetYDecDir(CACell c);`
  - `virtual inline unsigned short GetZIncDir(CACell c);`

- `virtual inline unsigned short GetZDecDir(CACell c);`

Первые четыре функции должны быть переопределены в потомке, а последние две по умолчанию возвращают ноль, но могут быть переопределены. Трехмерные решетки поддерживаются в качестве дополнительной возможности, а не основной.

- `public virtual inline CACell GetOrigin()` – функция возвращает координаты клетки, принятой за начало координат. По умолчанию функция возвращает ноль. Она может быть переопределена в потомке.
- `public virtual inline unsigned short GetNeighboursCount()` и `public virtual inline unsigned short GetMainNeighboursCount()` – функции возвращают число соседей и главных соседей клеток, соответственно. Должны быть переопределены в потомке.
- `public virtual CACell GetNeighbour(CACell c, unsigned short neig)` – функция для клетки с идентификатором `c` возвращает ее соседа, имеющего индекс, определяемый параметром `neig`. Этот параметр – целое число от нуля до результата, возвращенного функцией `NeighboursCount`, уменьшенного на единицу. Причем первые `GetMainNeighboursCount` соседей должны быть главными.

  Удобные идентификаторы соседних клеток для стандартных решеток можно найти в файле *%CAMEL_PATH%*[9]*\Lib\Neighbours.h*.

[9] Будем обозначать через `%CAMEL_PATH%` путь к установленному пакету *CAME&L*.

Функция должна быть переопределена в потомке.

- `public virtual void GetNeighbours(CACell c,CACell* pNeig)` – функция для клетки с идентификатором `c` возвращает через указатель `pNeig` массив идентификаторов ее соседей. Как привило, вызов этой функции приведет к получению результата быстрее, чем в случае запроса координат всех соседей по одному с помощью предыдущей функции.

  Размер массива равен числу, возвращаемому функцией `GetNeighboursCount`, причем первые `GetMainNeighboursCount` элементов должны соответствовать главным соседям.

  Функция может быть переопределена в потомке. Реализация по умолчанию вызывает `GetNeighbour` необходимое число раз, поэтому пользователям рекомендуется разрабатывать свой вариант реализации.

- `public virtual bool Query(CWnd* pParent, CACell& c)` – функция отображает окно для запроса координаты клетки в данной метрике. Параметр `c` определяет ссылку на переменную, хранящую идентификатор клетки, значение которого необходимо изменить. Параметр `pParent` определяет родительское окно. Функция возвращает `true` в случае удачного выполнения и `false` в противном случае.

  Для создания диалога необходимо разработать наследника класса `CMetricsDlg`.

  Функция должна быть переопределена в потомке.

- `public virtual void ToString(CACell c,STRING[10] s)` – функция возвращает через параметр `s` строковое представление координат клетки `c`. Функция должна быть переопределена в потомке.
- `virtual unsigned int CellType(CACell c)` и `virtual unsigned int PlaceType(CACell c)` – функции возвращают целочисленные типы клетки и места, соответственно. Функция используется для того, чтобы различать их, при необходимости, как, например, в треугольной решетке. По умолчанию функции возвращают ноль. Функции могут быть переопределены в потомке.
- `virtual inline CACell ToCell(CACell c1, CACell c2, CACell c3)` – функция преобразует координаты вдоль трех осей в идентификатор клетки. Координаты передаются через переменные типа `CACell`, но их требуется представлять, как обычные числа. Функция должна быть переопределена в потомке.
- `virtual inline void ToAxis(CACell c, CACell& i1, CACell& i2, CACell& i3)` – функция преобразует идентификатор клетки `c` в координаты вдоль трех осей. Координаты возвращаются, как переменные типа `CACell`, но представляют собой обычные числа. Функция должна быть переопределена в потомке.

[10] Макроопределение `STRING` обозначает указатель на первый символ строки с завершающим нулем.

3.3.7. Класс `CADatum` и шаблоны `CATypedDatum`, `CABasicCrtsDatum`, `CABasicGenDatum`

Класс `CADatum` является наследником класса `CAUnionMember`. Он базовый для всех классов, реализующих компоненты хранилища данных. Как упоминалось выше, компоненты этого типа отвечают за хранение состояния решетки в памяти, обеспечение двойной буферизации, сохранение и восстановление из файла, а также некоторые аспекты визуализации данных, а именно – преобразование значения, хранящегося в клетке, в два дескриптора цвета [65]. Опишем этот класс:

- `public CADatum()` – конструктор.
- `public bool bComputation` – переменная равна `true` во время итерации и `false` – в остальное время. Она используется средой, а также шаблоном классов `CATypedDatum`, который обсуждается далее.
- `public Zone z` – переменная хранит границы рабочей зоны решетки, состояния клеток которой подлежат вычислению на итерации.
- `public Zone zfull` – переменная хранит границы полной зоны решетки, информация о состоянии которой содержится в данном экземпляре компонента. Она включает в себя рабочую зону, а также "поля" (области решетки, клетки которых используются при вычислении новых состояний других клеток, но сами не обновляются).
- `public __int64 iRequestedCells` – переменная хранит количество клеток, состояния которых были запрошены по сети у удаленных подзадач.
- `public bool bSubdata` – переменная, определяющая хранит ли данный компонент состояние подрешетки, входит в "подэксперимент",

(если значение равно `true`) или полноценной решетки (если значение равно `false`).

- `public IPAddr ipOwner` – переменная хранит *IP*-адрес владельца задачи, если данный экземпляр компонента содержит состояние подрешетки.

Далее следуют функции этого класса, которые могут или должны быть переопределены в потомке:

- `public virtual inline COLORREF GetCellColor(CACell c)` – функция возвращает дескриптор цвета, который будет использоваться для отображения клетки с идентификатором `c`. Должна быть переопределена в потомке.
- `public virtual inline COLORREF GetPlaceColor(CACell c)` – функция возвращает дескриптор цвета, который будет использоваться для отображения места клетки с идентификатором `c`. По умолчанию она возвращает дескриптор, соответствующий черному цвету. Может быть переопределена в потомке.
- `public virtual bool Query(CWnd* pParent, long x, long y, CACell c, bool selected)` – функция отображает диалог для запроса значения состояния некой клетки или состояний множества клеток. Параметр `pParent` определяет родительское окна. Параметры `x` и `y` задают положение диалога относительно родительского окна. Если значение параметры `selected` равно `false`, то `c` определяет клетку, состояние которой запрашивается. Если значение `selected` равно `true`, то результат запроса будет присвоен всем, выделенным в данный момент, клеткам.

Функция возвращает `true` в случае удачного выполнения и `false` в противном случае. Она должна быть переопределена в потомке.

- `public virtual void Swap()` – функция обеспечивает переключение между двумя буферами данных при моделировании классического клеточного автомата. По умолчанию функция не делает ничего. Если хранилище не должно поддерживать двойную буферизацию данных, то не стоит менять реализацию по умолчанию, хотя она может быть переопределена в потомке.
- `public virtual inline bool CellExists(CACell c)` – функция возвращает `true`, если клетка с координатой `c` существует и `false` в противном случае. Эта функция позволяет использовать решетки сложным взаимным расположением клеток. По умолчанию функция возвращает значение `true`. Она может быть переопределена в потомке.
- `public virtual void Save(IOStream* st)` – функция обеспечивает сохранение состояние решетки в поток, на который указывает переменная `st`. Функция должна быть переопределена в потомке.
- `public virtual bool Load(IOStream* st)` – функция обеспечивает восстановление состояния решетки из потока, на который указывает переменная `st`. Возвращает значение `true` в случае удачного выполнения и `false` – в противном случае. Должна быть переопределена в потомке.
- `virtual inline void GetToMem(CACell c, char*& ptr, bool move=true)` – функция записывает состояние клетки `c` в память, куда указывает указатель `ptr`. Если параметр `move` равен

`true`, то указатель будет перемещен в конец записанных данных. Функция должна быть переопределена в потомке.

- `virtual inline void SetFromMem(CACell c, char*& ptr, unsigned int size=0)` – функция устанавливает состояния клетки `c` в значение, на которое указывает указатель `ptr`. По указанному адресу считывается число байтов, равное `size`. Если значение последнего параметра равно нулю, используется число байтов переменной состояния, определяемое функцией `SizeOfValue`. Она должна быть переопределена в потомке.
- `virtual unsigned int GetSerialToMem(Zone& z, SmartBuffer`[11]`& sb)` – функция копирует значения состояний клеток зоны `z` в "умный буфер" `sb`. Должна быть переопределена в потомке.
- `virtual void SetSerialFromMem(Zone& z ,char* data, unsigned int size)` – функция устанавливает клеткам зоны `z` значения состояний, хранящиеся в памяти, начиная с ячейки на которую указывает указатель `data`. Считается, что состояние каждой клетки занимает в памяти `size` байтов. Если значение последнего параметра равно нулю, используется размер состояния, определяемый функцией `SizeOfValue`. Должна быть переопределена в потомке.
- `virtual inline unsigned int SizeOfValue()` – функция возвращает размер значения состояния кленки в байтах. Должна быть переопределена в потомке.

---

[11] Класс `SmartBuffer` представляет средства для обмена данными посредствам сетевого протокола *CTP* [88, **Error! Reference source not found.**].

- `virtual inline void SetDefValue(CACell c)` – функция устанавливает данной клетке значение состояния по умолчанию. Вызывается средой, например, при вырезании множества выделенных клеток в буфер обмена. Реализация функции в рассматриваемом классе не делает ничего, но она может быть переопределена в потомке.
- `virtual inline bool NormalizeCell(CACell& c)` – функция приводит идентификатор клетки `c` к нормальному значению, в соответствии с граничными условиями – если клетка лежит за границами хранимой зоны, ее координаты будут, если это возможно, преобразованы так, чтобы она попадала в зону. Возвращает значение `false`, если нормализация не требуется и значение `true` в противном случае. По умолчанию функция возвращает значение `false`, но может быть переопределена в потомке. Она вызывается средой при организации распределенных вычислений.
- `virtual void CreateSubDatum(Zone& zfull, Zone& z, char* data, unsigned int size)` – функция обеспечивает создание подзадачи. Параметры имеют следующие назначения:
  - `zfull` – границы полной зоны подзадачи;
  - `z` – границы рабочей зоны подзадачи;
  - `data` – указатель на данные, хранящие состояния клеток подрешетки;
  - `size` – размер данных.

Однако непосредственных наследников класса `CADatum` создавать не следует. Так как хранилище данных, очевидно, тесно связано с типом данных, которые оно хранит, возникла необходимость создания шаблона `CATypedData<class type>`, добавляющего к описанному выше классу

функции, необходимые для работы с типом данных `type`. Опишем этот класс:

- `public virtual inline type Get(CACell c)` – функция возвращает значение, хранящееся в клетке `c`. Должна быть переопределена в потомке.
- `public virtual inline void Set(CACell c, type value)` – функция помещает в клетку `c` значение `value`. Возвращает значение `true` в случае удачного выполнения и значение `false` в противном случае. Должна быть переопределена в пользовательской библиотеке.

Кроме того, данный шаблон реализует описанные выше функции `SizeOfValue`, `GetSerialToMem` и `SetSerialFromMem`.

Написание компонентов хранилищ данных – непростая задача, при решении в которой, однако, можно выделить ряд стандартных приемов. Именно эти приемы использовались при написании шаблонов классов `CABasicCrtsDatum<class type, class dlg, int idd>` и `CABasicGenDatum <class type, class dlg, int idd>`. Они позволяют создавать хранилища для структур данных с двумерной декартовой и обобщенной метрикой, соответственно. Параметр `type` определяет тип хранимых данных. Параметр `dlg` – класс диалога (наследник класса `CDlg`) для запроса значений типа `type`. Параметр `idd` – идентификатор ресурса библиотеки, содержащего этот диалог.

Эти два шаблона берут на себя большую часть работы и реализуют функции члены `Swap`, `CellExists`, `Get`, `Set`, `Query`, `Save` и `Load`. Помимо этого они вводят ряд важных для компонента хранилища параметров. Если не требуется некой специфической, экономичной

структуры хранения данных, то эти шаблоны – большое подспорье при разработке компонентов. На откуп разработчику отдается лишь функция `GetCellColor(c)`. Однако, справедливости ради, надо отметить, что порой приходится переопределять и другие функции-члены, обращаясь при этом к их реализации в шаблонах.

3.3.8. Класс `CARules`

Данный класс, наследник класса `CAComponent`, является базовым для всех компонентов, реализующих правила клеточного автомата. Класс обладает широким спектром возможностей. Он включает в себя, в частности, средства для межавтоматных взаимодействий, кластерных вычислений и т.д. Опишем здесь только основные его члены:

- `public CARules()` – конструктор.
- `protected CAUnion* m_pUnion` – переменная хранит указатель на "союз компонентов", с которыми должны работать данные правила автомата. Союзом компонентов, как объяснялось в разд. 3.3.4, называется набор из совместимых друг с другом решетки, метрики и хранилища данных.
- `public void SetUnion(CAUnion* pUnion)` – функция устанавливает значение указателя на союз компонентов.
- `public CAUnion* GetUnion()` – функция возвращает значение указателя на союз компонентов.
- `public EnvInfo* m_pEnvInfo` – переменная хранит объект класса `EnvInfo`, который предназначен для описания платформы и окружения вычислений. Эта информация может быть проанализирована и учтена функциями класса `CARules`.

- `public void SetEnvInfo(EnvInfo* pEnvInfo)` – функция устанавливает описания платформы и окружения.
- `public bool m_bFinalizeIfChanged` – если значение этой переменной равно `true`, то после любого изменения состояния решетки пользователем будет произведена финализация эксперимента и перед следующим шагом потребуется повторная инициализация. Эта возможность оказывается полезной при использовании средств оптимизации производительности, кластерных вычислениях и т.п.

Далее описываются функции для управления итерациями, которые могут быть переопределены в потомках:

- `public virtual bool SubCompute(Zone& z)` – функция выполняет шаг автомата в области решетки, ограниченной значением первой координаты от `z.a1` до `z.b1`, второй – от `z.a2` до `z.b2` и третьей – `z.a3` до `z.b3`. Эта функция является вспомогательной. Она вызывается функцией `Compute` при распараллеливании вычислений на многопроцессорной системе, например, с помощью вспомогательного класса `RulesThread` на многопроцессорной, а также кластерной платформах.

  Возвращаемое функцией значение может быть как-то проинтерпретировано в контексте решаемой задачи.

  Необходимо отметить, что идеологически правильно применять ее и в тех случаях, когда вычисления не распределяются.

  Клеточные автоматы характеризуются тем, что результирующее состояние решетки не зависит от порядка обхода клеток или областей. При выполнении этого свойства вызов данной функции

для множества дизъюнктных зон, объединение которых покрывает всю решетку, приведет к тому же результату, что и вызов ее для единственной области, охватывающей всю решетку целиком.

Функция может быть переопределена в потомке, по умолчанию возвращает значение `true`.

Необходимо отметить, что, например, при использовании обобщенных координат [**Error! Bookmark not defined.**, 91] даже для двумерных и трехмерных решеток область ограничивается только вдоль одной оси – значения переменных `z.a2`, `z.b2`, `z.a3` и `z.b3` учитываться в этом случае не должны.

- `public bool SubCompute()` – функция вызывает предыдущую функцию, передавая в качестве параметра описание всей зоны, охватываемой текущим хранилищем данных. Оно хранится в переменной-члене `z`, наследуемой каждым хранилищем от класса `CADatum`.
- `public virtual bool Initialize()` – функция выполняет инициализацию эксперимента перед его стартом или выполнением шага после финализации. Если функция возвращает значение `true`, то инициализация считается прошедшей успешно и эксперимент может быть начат. Она может быть переопределена в потомке, по умолчанию возвращает значение `true`.
- `public virtual bool Compute()` – функция выполняет шаг автомата. Если функция возвращает значение `true`, то эксперимент может быть продолжен после данного шага. В противном случае он будет закончен и финализирован. Таким образом, можно задавать критерии окончания эксперимента. Функция может быть

переопределена в потомке. По умолчанию она вызывает функцию `SubCompute`, передавая в качестве параметра описание всей зоны, охватываемой текущим хранилищем данных.

- `public virtual void Finalize()` – функция выполняет финализацию эксперимента. Она может быть переопределена в потомке, по умолчанию не делает ничего.

Порядок вызова последних трех функций при проведении эксперимента иллюстрируется на рис. 26.

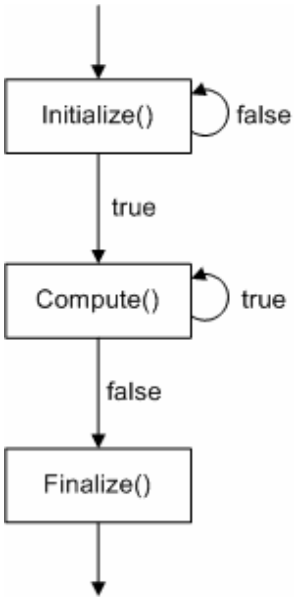


Рис. 26. Схема вызова функций класса `CARules` при проведении эксперимента

Кроме того, для управления состоянием решетки в классе предусмотрены следующие функции, которые могут быть переопределены в потомке.

- `public virtual void Tool1()`, `public virtual void Tool2()` , `public virtual void Tool3()` – функции реализуют три "пользовательских инструмента", которые могут быть вызваны из среды. Эти инструменты позволяют преобразовывать

или анализировать состояние решетки. По умолчанию функции отображают сообщения о том, что они не были переопределены.

В случае если пользователю потребуется больше трех инструментов, можно использовать параметры-инструменты (объекты класса `ParamTool`), реализуя необходимое поведение внутри функции `ParameterChanged`.

Для определения анализируемых параметров, характеризующих эксперимент, предусмотрены две функции:

- `public virtual inline unsigned short GetAnalyzableParametersCount()` – функция возвращает число анализируемых параметров данного компонента правил. Она может быть переопределена в потомке. По умолчанию она возвращает ноль.
- `public virtual inline Parameter* GetAnalyzableParameter(unsigned short n)` – функция возвращает указатель на *n*-ый анализируемый параметр компонента, где *n* принимает значение от нуля до значения, которое вернет функция `GetAnalyzableParametersCount`, уменьшенного на единицу. Функция может быть переопределена в потомке, по умолчанию возвращает `NULL`.

  Для удобства реализации двух последних функции, как и в случае обычных параметров, предусмотрены макроопределения:

  - `APARAMETERS_COUNT(<число_параметров>)` – для задания числа анализируемых параметров;
  - `BEGIN_APARAMETERS` – для того, чтобы начать карту анализируемых параметров;

- END_APARAMETERS – для того, чтобы закончить карту.

Параметры в карту добавляются с помощью описанного в разд. 3.3.3 макроопределения PARAMETER.

Опишем две переменные и две функции, входящие в класс CARules, которые предназначены для организации кластерных вычислений.

- public Network* pNetwork – переменная хранит указатель на описание сетевого контекста и интерфейса рабочей станции.
- public Cluster* pCluster – переменная хранит указатель на менеджер кластера.
- public virtual unsigned int CreateCluster(unsigned int overlap, unsigned char type, bool usethis, int remoteness) – функция разделяет задачу на подзадачи, отбирает машины, участвующие в эксперименте и рассылает команды для создания вспомогательных экспериментов. Функция имеет следующие параметры:
  - overlap – определяет величину полей. Они должны быть не меньше наибольшего расстояния от данной клетки до клетки окрестности (радиуса окрестности);
  - type – определяет способ разделения задачи. В данный момент поддерживается следующие значения:
    - 0 – отладочный способ, который делит решетку на вертикальные полосы. Количество полос определяется параметром remoteness. Все части эксперимента создаются на главной машине;
    - 1 – разделение на равные вертикальные полосы. Их количество зависит от числа участвующих машин;

- 2 – разделение на вертикальные полосы, пропорциональные факторам производительности машин ("*throughput factor*")[12] . Их количество зависит от числа участвующих машин;
- другие значения могут быть приняты пользователем для обозначения разнообразных стратегий разделения при переопределении данной функции.

o `usethis` – определяет, учитывать ли главную машину при разделении задачи. Если значение параметра равно `true`, то на ней тоже будет создан вспомогательный эксперимент.

o `remoteness` – определяет удаленность машин, которые следует привлекать к эксперименту (в настоящее время не поддерживается).

Функция может быть переопределена в потомке.

- `public virtual bool ComputeCluster(bool gather=false)` – функция посылает всем участникам эксперимента команды для осуществления "подитераций" (выполнения шага при решении подзадач) и ожидает ответов о ее завершении на каждой из них. Параметр `gather` определяет, требуется ли собрать состояние всей решетки на главной машине (владельце эксперимента) после данной итерации или нет. Функция может быть переопределена в потомке.

[12] Величина фактора производительности определяется средой для каждой машины, на которой она запускается. Пользователь может самостоятельно определить его, выбрав в меню среды "*Tools*" | "*Workstation Characteristics*...".

3.3.9. Классы параметров компонентов и диалогов для запросов значений параметров

Базовым классом для всех классов параметров компонентов является класс `Parameter`. Опишем его:

- `protected STRING sName` – переменная, которая хранит имя параметра;
- `protected STRING sInfo` – переменная, которая хранит описание параметра;
- `protected void* pComp` – переменная, которая хранит указатель на компонент, владеющий этим параметром;
- `public Parameter(STRING name,STRING info,void* pComp=NULL)` – конструктор, присваивающий значения переменным `sName`, `sInfo` и `pComp`;
- `public ~Parameter()` – деструктор;
- `public virtual inline STRING GetName()` – функция возвращает имя компонента;
- `public virtual inline STRING GetInfo()` – функция возвращает описание компонента;
- `public virtual HICON GetIcon()` – функция возвращает дескриптор иконки, соответствующей компоненту. Если она вернет `NULL`, то будет использована иконка по умолчанию. Для разработки единообразных иконок, в качестве основы для новых изображений пользователю рекомендуется использовать файл *%CAMEL_PATH%\Pics\Param.ico*;

- `protected bool bWasSet` – переменная, определяющая было ли значение данного параметра установлено или нет;
- `inline bool GetWasSet()` – функция возвращает значение переменной `bWasSet`;
- `inline void SetWasSet(bool ws)` – функция устанавливает значение переменной `bWasSet`;
- `protected short iAnalizers` – переменная, которая хранит количество анализаторов данного параметра;
- `inline bool IsAnalyzed()` – функция позволяет определить анализируется ли данный параметр;
- `inline void AnalyzerAdded()` – функция увеличивает число анализаторов данного параметра на единицу;
- `inline void AnalyzerRemoved()` – функция уменьшает число анализаторов данного параметра на единицу.

Далее перечислим функции, которые подлежат обязательному переопределению в классах-наследниках.

- `public virtual void GetString(STRING str)` – функция возвращает через параметр `str` строковое представление значения параметра для отображения в дереве компонентов. Должна быть переопределена в потомке.
- `public virtual void Store(STRING s)` – функция возвращает через параметр `s` строковое представление значения параметра для его сохранения в файле. Должна быть переопределена в потомке.
- `public virtual bool Restore(STRING s)` – функция восстанавливает значение параметра по его строковому представлению,

полученному с помощью предыдущей функции. Должна быть переопределена в потомке.

- `public virtual bool Query(CWnd* pParent, long x, long y)` – функция отображает диалог для запроса значения параметра над родительским окном `pParent` с верхним левым углом в точке с координатами (`x`; `y`). Возвращает значение `true` в случае успешного присвоения параметру нового значения и значение `false` в противном случае. Должна быть переопределена в потомке.
- `virtual inline int GetTypeID()` – функция возвращает целочисленный идентификатор типа параметра. Возможные возвращаемые значения приведены в таблице 9. Кроме них функция может возвращать определенные пользователем значения, не меньшие `PARAM_NEXT_TYPE_ID`.
- `virtual inline void ReleaseIcon()` – функция должна обеспечивать освобождение ресурса, хранящего иконку, соответствующую компоненту.

Однако не следует наследовать новые классы параметров непосредственно от класса `Parameter`. Для этих целей служит его наследник, шаблон классов `TypedParameter<class type, int icon, int dlg, class dlgclass, int type_id>`, позволяющий быстро разрабатывать параметры для новых типов данных. Аргумент шаблона `type` определяет тип значения параметра, `icon` – идентификатор ресурса, хранящего соответствующую иконку в библиотеке ресурсов *CADLib*, `dlg` – идентификатор ресурса, хранящего диалог для запроса значений, `dlgclass` – класс диалога (наследник класса `CDlg`), а `type_id` – целочисленный идентификатор типа параметра.

В описываемом шаблоне представлены стандартные реализации конструктора, а также функций `ReleaseIcon`, `GetIcon`, `Query`. Кроме того, в этом классе дополнительно вводятся следующие функции:

- `virtual inline type Get()` – функция возвращает значение параметра.
- `virtual inline bool Set(type value, bool notify=true)` – функция устанавливает значение параметра равным `value`. Если значение флага `notify` равно `true`, то при этом будет вызвана функция `ParameterChanged` компонента, владеющего данным параметром, иначе – не будет. Функция возвращает значение `true`, если новое значение корректно, и оно было присвоено. В противном случае она возвращает значение `false`. Функция должна быть переопределена в потомке.

Для опрашивания пользователя следует использовать диалоги, функциональность которых реализуют наследники класса `CDlg`. Опишем этот класс:

- `public bool bResult` – переменная, получающая значение `true` в случае успешного запроса и `false` – в противном случае.
- `public bool bParam` – переменная, определяющая диалог был создан для запроса параметра (если значение равно `true`) или состояния некой клетки (если значение равно `false`). Конструкторы класса `CDlg` присваивают полю истинное значение. Если класс-наследник реализует функциональность диалога для запроса состояния клетки, то в нем требуется предусмотреть соответствующий конструктор.

- `public CWnd* pParent` – переменная, которая хранит указатель на родительское окно.
- `protected long x` – переменная, которая хранит абсциссу верхнего левого угла окна диалога.
- `protected long y` – переменная, которая хранит ординату верхнего левого угла окна диалога.
- `protected CDlg(CWnd* pParent)` – конструктор, создающий диалог с родительским окном `pParent`.
- `public CDlg(CWnd* pParent, long x, long y)` – конструктор, создающий диалог с родительским окном `pParent` и верхним левым углом в точке (*x*; *y*).
- `private virtual BOOL OnInitDialog()` – функция, вызываемая для обработки инициализации диалога. Не должна вызываться пользователем.
- `private afx_msg void OnDestroy()` – функция, вызываемая для обработки уничтожения диалога. Не должна вызываться пользователем.

Далее следуют функции, переопределяемые пользователем:

- `public virtual void PrePrompt()` – функция, вызываемая перед отображением диалога. В ней следует инициализировать элементы управления и т.п.
- `private virtual void OnOK()` – функция, вызываемая после завершения запроса (пользователь нажал кнопку "*OK*"). Эта функция должна анализировать введенное значение и присваивать его параметру или состоянию клетки.

- `public virtual void PostPrompt()` – функция, вызываемая после закрытия диалога.

В таблице 9 приведены имена классов, реализующих стандартные параметры, имена классов соответствующих им диалогов, идентификаторы их типов, а также словесные описания.

Таблица 9. Стандартные классы параметров, соответствующие им классы диалогов, идентификаторы типов и описания

Класс параметра	Класс диалога	Идентификатор типа	Описание типа параметра
ParamInt	CIntDlg	PARAM_INT	Целое число
ParamDouble	CDoubleDlg	PARAM_DOUBLE	Число с плавающей точкой (двойной точности)
ParamBool	CBoolDlg	PARAM_BOOL	Булева переменная
ParamColor	CColorDlg	PARAM_COLOR	Цветовой дескриптор
ParamIntGap	CIntGapDlg	PARAM_INT_GAP	Отрезок с целочисленными границами
ParamChoice	CChoiceDlg	PARAM_CHOICE	Конечное множество альтернатив
ParamContSpectrum	CContSpectrumDlg	PARAM_CONT_SPECTRUM	Непрерывный цветовой спектр
ParamDiscSpectrum	CDiscSpectrumDlg	PARAM_DISC_SPECTRUM	Дискретный цветовой спектр
ParamCell	CCellDlg	PARAM_CELL	Идентификатор

			клетки решетки
ParamString	CStringDlg	PARAM_ STRING	Строка
ParamFilePath	CFilePathDlg	PARAM_ STRING	Путь к файлу
ParamTool	CToolDlg	PARAM_TOOL	Инструмент

Помимо наследников, перечисленных в таблице, от класса CDlg происходит класс CMetricsDlg, используемый для запроса координат клеток решетки в виде, сообразном текущей метрике. В этом классе добавляется конструктор и поля для хранения идентификатора выбранной клетки и союза компонентов. Существуют дочерние классы этого класса для декартовых (класс CCartesianMDlg) и обобщенных (класс CGeneralizedMDlg) координат.

3.4. Разработка библиотеки, содержащей компонент

Как отмечалось выше, компонент должен быть помещен в динамическую библиотеку *DLL* (*Dynamic-Link Library*). Помимо класса, реализующего компонент, в ней должны находиться три функции, описываемые ниже:

- EXPORTIT[13] STRING GetCompatibilityInfo() – функция аутентификации библиотеки и компонента. Возвращает строку вида "CAMEL Component|<версия_ядра>|<тип_компонента>", где

[13] Использование макроопределения EXPORTIT обеспечивает экспортирование функции из библиотеки.

- <версия_ядра> – версия ядра инструментального средства, с которой совместим данных компонент, например "1.0". Совместимость "вниз", по возможности, должна обеспечиваться;
- <тип_компонента> – строка "Grids", "Data", "Metrics", "Rules" или "Analyzers", в зависимости от типа компонента, содержащегося в библиотеке.

Для удобства определения этой функции предусмотрены макроопределения COMPATIBLE_GRID(<версия_ядра>), COMPATIBLE_METRICS(<версия_ядра>), COMPATIBLE_DATUM(<версия_ядра>), COMPATIBLE_RULES(<версия_ядра>) и COMPATIBLE_ANALYZER(<версия_ядра>).

- Функция создания компонента, возвращающая указатель на новый экземпляр. В зависимости от типа она может иметь вид:
  - EXPORTIT CAGrid* CreateGrid();
  - EXPORTIT CAMetrics* CreateMetrics();
  - EXPORTIT CADatum* CreateDatum();
  - EXPORTIT CARules* CreateRules();
  - EXPORTIT CAAnalyzer* CreateAnalyzer().
- Функция удаления компонента, освобождающая память, занимаемую им. В зависимости от типа функция удаления может иметь вид:
  - EXPORTIT DestroyGrid(CAGrid* pGrid);
  - EXPORTIT DestroyMetrics(CAMetrics* pMetrics);
  - EXPORTIT DestroyDatum(CADatum* pDatum);
  - EXPORTIT DestroyRules(CARules* pRules);
  - EXPORTIT DestroyAnalyzer(CAAnalyzer* pAnalyzer).

Для удобства реализации двух последних функций предусмотрены макроопределения `GRID_COMPONENT(<имя_класса>)`, `METRICS_COMPONENT(<имя_класса>)`, `DATUM_COMPONENT(<имя_класса>)`, `RULES_COMPONENT(<имя_класса>)` и `ANALYZER_COMPONENT(<имя_класса>)`.

В результате, например, при написании библиотеки, содержащей класс компонента решетки `CAMyGrid`, совместимого с ядром версии 1.0[14], помимо реализации самого класса достаточно добавить две строки:

```
COMPATIBLE_GRID(1.0)
GRID_COMPONENT(CAMyGrid)
```

Для создания компонентов рекомендуется использовать компилятор и среду разработки *Microsoft Visual C++* 6, так как пакет *CAME&L* создавался с помощью этого средства. Таким образом, весь набор примеров, стандартных компонентов, также ориентирован на него.

Для реализации нового компонента следует создать в среде разработки проект, представляющий собой динамическую библиотеку (выбрать в меню "*File*" | "*New...*" | "*Projects*" | "*Win32 Dynamic-Link Library*", указать имя и директорию). В появившемся после этого мастере выбрать вариант "*An empty DLL project*". После этого необходимо подключить к проекту библиотеку

---

[14] В настоящее время ведется разработка ядра версии 1.1, которое будет обладать существенно более широким спектром возможностей, чем предыдущая версия. Развитие ядра вызвано, в первую очередь ростом популярности и активным использованием инструментального средства.

CADLib (выбрать в меню "*Project*" | "*Settings*" | "*Link*" и в категории "*General*" в поле "*Object/library modules*" добавить "*CADLib.lib*", а в категории "Input" в поле "*Additional library path*" добавить "*%CAMEL_PATH%\Lib*"). Теперь можно приступать к разработке, создавать новые файлы, реализующие компонент, а также его иконку и прочие ресурсы.

Однако, для того, чтобы избавить пользователя от необходимости настраивать среду для каждого нового компонента был разработан скрипт *ccomp.sh*, служащий для копирования существующих компонентов. Под операционной системой семейства *Windows* он может быть выполнен с помощью пакета *Cygwin* [66], средства эмуляции оболочки операционных систем семейства *UNIX*/*Linux*.

Данный скрипт доступен с сайта [**Error! Bookmark not defined.**], созданного автором. Перед использованием скрипта его необходимо поместить в директорию "*%CAMEL_PATH%\CompsSrc*\<тип_компонента>", в которой обычно размещаются исходные коды компонентов.

Скрипт поддерживает следующие параметры из командной строки:

```
$>  ccomp.sh  <директория_из_которой_копировать>
<директория_в_которую_копировать> <опции>
```

Поддерживаются следующие опции (в алфавитном порядке):

- `-c <имя_класса>` – установить соответствующее имя класса для копии компонента;
- `-i <описание>` – установить соответствующее описание для копии компонента;

- `-n <имя>` – установить соответствующее имя[15] для копии компонента.

Для получения информации о параметрах командной строки можно также воспользоваться встроенной в скрипт помощью, вызвав

```
$> ccomp.sh -h
```

Например, если в директории *%CAMEL_PATH%\CompsSrc\Rules\MyRules1* находится компонент, который можно использовать в качестве основы для нового компонента правил, то следует воспользоваться описанным выше скриптом, поместив его в директорию *%CAMEL_PATH%\CompsSrc\Rules* и вызвав

```
$> ccomp.sh MyRules1/ MyRules2/ -c CAMyRules2 -i "This is my second rules component" -n MyRules2
```

В результате компонент будет скопирован, в директорию *%CAMEL_PATH%\CompsSrc\Rules\MyRules2*. Для копии будет установлено имя класса `CAMyRules2`, описание "`This is my second rules component`" и имя компонента "`MyRules2`".

Описанный выше скрипт для копирования компонентов позволяет существенно увеличить скорость разработки новых решений.

Все стандартные компоненты представляют собой аналогичные библиотеки, которые для среды никак не отличаются от библиотек, разработанных пользователем.

---

[15] Здесь имеется в виду имя, под которым компонент виден в среде, возвращаемое его функцией-членом `GetName`, а не имя класса.

Примеры решения задач, приводимые в главе 4, иллюстрируют тот факт, что библиотека *CADLib* предоставляет разработчику богатейший набор средств. При работе с инструментальным средством *CAME&L* знание языка программирования *C++* позволит решать широкий спектр задач.

Последний тезис может показаться странным, так как владения языком *C++* само по себе позволяет решать вычислительные задачи, не применяя специализированных пакетов. Кроме того, это – нетривиальный навык, рассчитывать на наличие которого у пользователя не правомерно.

Однако, во-первых, организация параллельных и распределенных вычислений с помощью любого из существующих средств требует от пользователя существенных программистских навыков.

Во-вторых, пакет *CAME&L* берет на себя реализацию огромного количества вспомогательной функциональности, от сетевого взаимодействия до визуализации и сохранения экспериментов, что в полной мере оправдывает его использование.

В-третьих, программный интерфейс библиотеки *CADLib* построен из настолько общих соображений, что позволяет, например, разработать компонент правил, который в качестве параметра принимал бы описание клеточного автомата на неком специализированном языке, как *FORTH* [1, 35] или *CARPET* [41]. В результате, конечный пользователь будет избавлен от необходимости владеть языком *C++*.

Все приводимые в данной работе компоненты, а также инструментальное средство *CAME&L* доступны с сайта [**Error! Bookmark not defined.**].

Выводы по главе 3

1. Обоснована структура инструментального средства *CAME&L*.

2. Разработана среда выполнения вычислительных экспериментов на основе клеточных автоматов на различных вычислительных архитектурах, входящая в состав инструментального средства *CAME&L*.
3. Разработана библиотека разработки клеточных автоматов *CADLib*, входящая в состав инструментального средства *CAME&L*.
4. Разработан набор стандартных компонентов для типовых вычислительных экспериментов, входящий в состав инструментального средства *CAME&L*.
5. Показаны и описаны средства для создания библиотеки, содержащей пользовательский компонент.

Глава 4. Применение инструментального средства *CAME&L* для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов

В настоящей главе приводятся примеры применения рассматриваемого инструментального средства автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов.

На примере игры "Жизнь" демонстрируется разработка простейшего вычислительного эксперимента, а также использование зональной оптимизации, поддержка многопроцессорной и кластерной вычислительных систем, обобщенных координат и создание универсальных правил, не зависящих от платформы.

В следующем разделе этой главы приводится пример численного решения уравнения теплопроводности, который показывает применение предложенного инструментального средства для решения физических задач.

Еще один раздел этой главы посвящен проведению научного исследования с помощью инструментального средства *CAME&L*, которое позволило провести классификацию структур, порождаемых одномерными двоичными клеточными автоматами из точечного зародыша. При этом подсчитано количество классов автоматов при использовании различных операций инвариантности, что вносит вклад в теорию клеточных автоматов.

В заключительном разделе главы описан набор средств, разработанный для того, чтобы организовывать вычисления на *FPGA*-процессорах (*Field*

Programmable Gate Array) с помощью инструментального средства *CAME&L*. Это расширяет множество вычислительных платформ, на которых могут быть организованы вычислительные эксперименты с помощь метода и инструментального средства, разработанных в диссертации.

4.1. Проектирование вычислительных экспериментов с помощью инструментального средства *CAME&L*

Любой эксперимент, проводимый с помощью инструментального средства *CAME&L*, представляет собой выполнение некого клеточного автомата. Поэтому подход к решению задач с помощью предложенного средства представляет собой, в первую очередь, описание процесса построения набора компонентов, реализующих этот автомат. Так как непосредственная разработка библиотек, содержащих компоненты, представляет собой сугубо технический вопрос, который иллюстрируется приводимыми в настоящей главе примерами, то в настоящем разделе обсудим соображения, которыми следует руководствоваться при выборе компонентов всех четырех типов, которые формирую клеточный автомат.

Стандартный набор компонентов решеток, содержит все возможные двумерные решетки из правильных многоугольников, а также трехмерную решетку из кубов[16]. Этого достаточно для решения подавляющего большинства задач. Задачи, требующие каких-либо иных решеток, представляются настолько специфическими, что, в любом случае, включать такие решетки в набор стандартных компонентов было бы не рационально. Если пользователь столкнулся с одной из таких задач, то ему придется

[16] Этот компонент появился, начиная с версии 1.1 библиотеки *CADLib*.

реализовывать свою решетку, воспользовавшись одной из стандартных в качестве примера.

Выбор или создание нового хранилища данных представляется чрезвычайно простым. В стандартный набор компонентов включено немало хранилищ для элементарных типов данных, которые могут быть востребованы при решении огромного множества учебных задач. Однако, в научных задачах моделирования систем, распределенных по некому параметру, как правило, состояние клетки автомата представляет собой структуру, содержащую в себе набор свойств или, иными словами, переменных различных типов. Библиотека *CADLib* позволяет очень просто создавать хранилища для новых типов данных, посредством *C++*-шаблонов `CABasicCrtsDatum` (для обобщенных координат или одномерного пространства клеток), `CABasicCrtsDatum` (для двумерного пространства клеток), `CABasicCrts3DDatum`[17] (для трехмерного пространства клеток). При этом библиотека компонента будет представлять собой лишь объявление пользовательского класса, как наследника одного из шаблонов. В общей сложности, исходный код библиотеки будет занимать десять строк. Так как в стандартной библиотеке невозможно предусмотреть все варианты типов данных, которые могут понадобиться пользователю в ходе решения задачи, то решение с использованием шаблонов представляется оптимальным. В результате инструментальное средство *CAME&L* предоставляет пользователю полную свободу в части выбора типа данных для представления состояния клетки в одномерном, двумерном или трехмерном пространствах. Поддержка пространств большей размерности также возможна, однако это требует более трудоемкой реализации.

---

[17] Этот класс появился в библиотеке *CADLib*, начиная с версии 1.1.

Варианты разнообразных обобщенных координат, а также декартовы метрики для двумерных и трехмерных[18] решеток включены в стандартный набор компонентов. Как и в случае с решетками, этого достаточно для решения подавляющего большинства задач. Задачи, требующие каких-либо иных метрик, представляются весьма специфическими (подчас, задача может заключаться, собственно, в построении той или иной метрики). Если пользователь столкнулся с настолько специфической задачей, то ему придется реализовывать свою метрику, воспользовавшись одной из стандартных в качестве примера. Особняком стоит вопрос о моделировании одномерных клеточных автоматов. Так как обычно представляет интерес изучение структуры, порождаемой ими во времени, то их следует моделировать с помощью двумерного клеточного автомата. Именно таким образом реализован одномерный клеточный автомат для проведения исследования, обсуждаемого в разд. 4.4.

Разработка компонента правил является неотъемлемой частью построения решения пользовательской задачи. Именно в этом компоненте и описывается суть вычислительного эксперимента: функция переходов, используемые методы оптимизации и параллелизации вычислений, перечисляются анализируемые параметры эксперимента. Для создания универсального компонента правил, позволяющего эффективно использовать однопроцессорную, многопроцессорную и кластерную вычислительные платформы, библиотека *CADLib* предоставляет базовый класс `CAUniRules`, использование которого обсуждается в разд. 4.2.6.

Таким образом, в подавляющем большинстве случаев, пользователь реализует только компонент правил и, возможно, хранилище данных, что не

---

[18] Этот компонент появился, начиная с версии 1.1 библиотеки *CADLib*.

представляет трудностей, в виду богатства и простоты предоставляемых библиотекой *CADLib* вспомогательных средств.

Как отмечалось выше, в эксперименте также могут принимать участие компоненты пятого типа, анализаторы. В стандартный набор компонентов входят анализаторы-мониторы (отображающие текущие значение параметров булева, целочисленного и вещественного типов), анализаторы-построители файлов отчетов (сохраняющие значения параметров булева, целочисленного и вещественного типов в файлы) и анализаторы-построители графиков (отображающие динамику изменения целочисленных и вещественных параметров в виде графиков). Кроме этого имеется специальный анализатор производительности, позволяющий строить график продолжительности каждой итерации от времени для изучения и настройки вычислительной системы. Задачи, требующие создания специфических анализаторов, выходящих за рамки перечисленных здесь, могут возникнуть не столь редко, как те задачи, которые требуют решеток или метрик, не входящих в набор стандартных компонентов. В любой эксперимент может быть вовлечено произвольное количество анализаторов.

В приводимых далее примерах решения задач и проведения исследования с помощь инструментального средства *CAME&L* не возникает потребности в разработке специфических решеток, хранилищ данных или метрик, поэтому основное внимание будет уделяться именно созданию компонентов правил. В разд. 4.4 будет обсуждаться суть исследования, проводимого с помощью инструментального средства *CAME&L*, в то время как исходный код соответствующего компонента правил вынесен в приложение 1.

4.2. Проектирование и реализация клеточного автомата "Жизнь" для различных вычислительных платформ

Клеточный автомат Джона Хортона Конвея "Жизнь" [27–29] широко известен.

Данный раздел посвящен шести вариантам реализации этого автомата: простейшей реализации "в лоб" (без указания вычислительной архитектуры, оптимизации и прочих подобных вопросов), реализации с зональной оптимизацией, реализации для многопроцессорной и кластерной вычислительных систем, для обобщенных координат [91] а также – универсальной, не зависящей от вычислительной платформы реализации[19].

4.2.1. Простая реализация, без указания вычислительной архитектуры и оптимизации

Приведем реализацию игры "Жизнь" для картезианской метрики и без указания вычислительной архитектуры, то есть для однопроцессорной системы и без оптимизации.

При этом необходимо разработать класс, представляющий собой соответствующий компонент правил. Назовем его `CALifeRules`. В качестве решетки автомата можно выбрать любую из стандартных двумерных решеток (наиболее привычна квадратная, стандартный компонент "*Square Basic Grid*"). В качестве метрики – картезианские координаты (стандартный компонент "*Cartesians*"), а хранилища данных – хранилище для булевых величин (стандартный компонент "*Booleans for Cartesians*").

---

[19] При этом используется класс `CAUniRules`, который не входит в официальную поставку *CADLib* версии 1.0, но, несомненно, будет включен в релиз 1.1.

Необходимо отметить, что перед проведением эксперимента потребуется включить хранение данных на двух последовательных шагах, двойную буферизацию (параметр "*Double*" хранилища перевести в значение `true`). В результате внутри компонента будут созданы две копии структуры данных для размещения состояния решетки. Значения будут получаться из одной из них, а помещаться в другую. После каждого шага структуры будут меняться местами. Это обеспечит независимость результирующего состояния решетки от порядка перебора клеток, неотъемлемое свойство "классических" клеточных автоматов.

На рис. 27 изображен фрагмент решетки клеточного автомата, реализующего игру "Жизнь" в инструментальном средстве *CAME&L*. В эксперименте участвуют описанные выше решетка, хранилище данных и метрика, входящие в набор стандартных компонентов, а также разрабатываемые ниже правила.

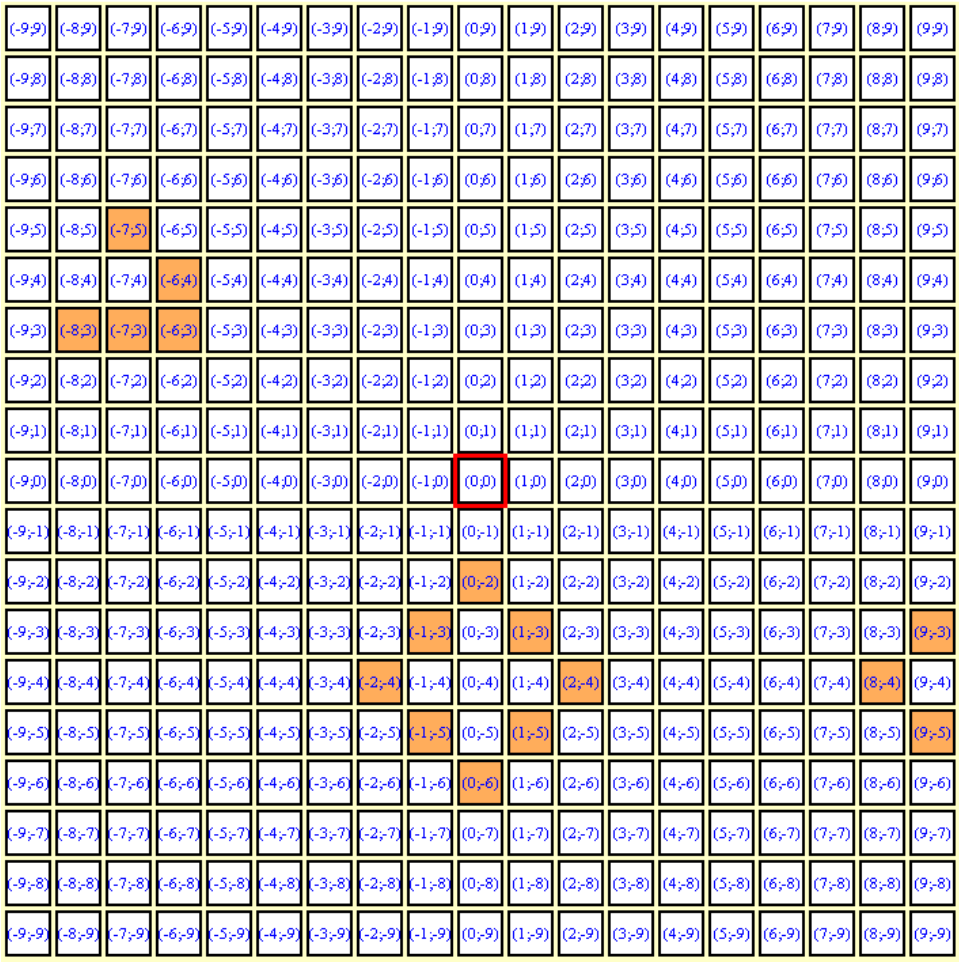


Рис. 27. Фрагмент решетки клеточного автомата, реализующего игру "Жизнь" без указания вычислительной архитектуры и оптимизации

Введем в разрабатываемом компоненте правил два набора по 13 параметров булевского типа (массивы `p_bBorn`[20] и `p_bDie`), которые позволят определять функцию переходов автомата. Если значение параметра `p_bBorn[i]` равно `true`, то мертвая клетка оживает при наличии *i* живых соседей. Если значение параметра `p_bDie[i]` равно `true`, то живая клетка умирает при наличии *i* живых соседей. Значение *i* лежит от нуля до

[20] Названия переменных-членов классов компонентов, представляющих собой параметры, имеет смысл начинать с префикса "`p_`".

двенадцати (наибольшего числа непосредственных соседей на треугольной решетке), однако на квадратной решетке будут использованы лишь первые девять из них (от нуля до восьми).

Кроме того, введем целочисленный анализируемый параметр `p_iAlive`, который позволит наблюдать за динамикой изменения количества живых клеток на решетке.

Объявление класса `CALifeRules` может иметь следующий вид:

```
class CALifeRules:public CARules
{
public:
    // Конструктор и деструктор
    CALifeRules();
    virtual ~CALifeRules();

    // Определение имени, описания, иконки и условий
    // совместимости
    COMPONENT_NAME(Game of Life (plain))
    COMPONENT_INFO(Conways game of Life rules
      (plain variant))
    COMPONENT_ICON(IDI_ICON)
    COMPONENT_REQUIRES(Data.bool.*&
      Metrics.2D.cartesian.*)

    // Следующие функций класса CARules будут
    // переопределены
    virtual bool Initialize();
    virtual bool SubCompute(Zone& z);
```

```
public:
    // Параметры
    ParamBool* p_bBorn[13];
    ParamBool* p_bDie[13];

    // Анализируемый параметр
    ParamInt p_iAlive;

public:
    // Карта параметров
    PARAMETERS_COUNT(26)
    BEGIN_PARAMETERS
        PARAMETER(0,p_bBorn[0])
        PARAMETER(1,p_bBorn[1])
        PARAMETER(2,p_bBorn[2])
        PARAMETER(3,p_bBorn[3])
        PARAMETER(4,p_bBorn[4])
        PARAMETER(5,p_bBorn[5])
        PARAMETER(6,p_bBorn[6])
        PARAMETER(7,p_bBorn[7])
        PARAMETER(8,p_bBorn[8])
        PARAMETER(9,p_bBorn[9])
        PARAMETER(10,p_bBorn[10])
        PARAMETER(11,p_bBorn[11])
        PARAMETER(12,p_bBorn[12])
        PARAMETER(13,p_bDie[0])
```

```
        PARAMETER(14,p_bDie[1])
        PARAMETER(15,p_bDie[2])
        PARAMETER(16,p_bDie[3])
        PARAMETER(17,p_bDie[4])
        PARAMETER(18,p_bDie[5])
        PARAMETER(19,p_bDie[6])
        PARAMETER(20,p_bDie[7])
        PARAMETER(21,p_bDie[8])
        PARAMETER(22,p_bDie[9])
        PARAMETER(23,p_bDie[10])
        PARAMETER(24,p_bDie[11])
        PARAMETER(25,p_bDie[12])
    END_PARAMETERS

    // Карта анализируемых параметров
    APARAMETERS_COUNT(1)
    BEGIN_APARAMETERS
        PARAMETER(0,&p_iAlive)
    END_APARAMETERS

private:
    // Вспомогательная переменная для вычисления
    // значения p_iAlive
    int iq;
};
```

Три необходимых каждой библиотеке компонента функции можно создать с помощью описанных выше макроопределений:

```
COMPATIBLE_RULES(1.0)
RULES_COMPONENT(CALifeRules)
```

Основное назначение конструктора и деструктора компонента состоит в создании, инициализации, а также удалении параметров. В данном случае они будут иметь вид:

```
CALifeRules::CALifeRules():
    p_iAlive("Alive","Amount of alive cells",0)
{
    p_bBorn[0]=new ParamBool("Born if 0","Is dead cell
      to become alive if it has 0 alive neighbors",
      false, this);
    p_bBorn[1]=new ParamBool("Born if 1","Is dead cell
      to become alive if it has 1 alive neighbors",
      false, this);
    p_bBorn[2]=new ParamBool("Born if 2","Is dead cell
      to become alive if it has 2 alive neighbors",
      false, this);
    p_bBorn[3]=new ParamBool("Born if 3","Is dead cell
      to become alive if it has 3 alive neighbors",
      true, this);
    p_bBorn[4]=new ParamBool("Born if 4","Is dead cell
      to become alive if it has 4 alive neighbors",
```

```
  false, this);
p_bBorn[5]=new ParamBool("Born if 5","Is dead cell
  to become alive if it has 5 alive neighbors",
  false, this);
p_bBorn[6]=new ParamBool("Born if 6","Is dead cell
  to become alive if it has 6 alive neighbors",
  false, this);
p_bBorn[7]=new ParamBool("Born if 7","Is dead cell
  to become alive if it has 7 alive neighbors",
  false, this);
p_bBorn[8]=new ParamBool("Born if 8","Is dead cell
  to become alive if it has 8 alive neighbors",
  false, this);
p_bBorn[9]=new ParamBool("Born if 9","Is dead cell
  to become alive if it has 9 alive neighbors",
  false, this);
p_bBorn[10]=new ParamBool("Born if 10","Is dead
  cell to become alive if it has 10 alive
  neighbors", false, this);
p_bBorn[11]=new ParamBool("Born if 11","Is dead
  cell to become alive if it has 11 alive
  neighbors", false, this);
p_bBorn[12]=new ParamBool("Born if 12","Is dead
  cell to become alive if it has 12 alive
  neighbors", false, this);

p_bDie[0]=new ParamBool("Die if 0","Is alive cell
```

```
  to become dead if it has 0 alive neighbors",
  true, this);
p_bDie[1]=new ParamBool("Die if 1","Is alive cell
  to become dead if it has 1 alive neighbors",
  true, this);
p_bDie[2]=new ParamBool("Die if 2","Is alive cell
  to become dead if it has 2 alive neighbors",
  false, this);
p_bDie[3]=new ParamBool("Die if 3","Is alive cell
  to become dead if it has 3 alive neighbors",
  false, this);
p_bDie[4]=new ParamBool("Die if 4","Is alive cell
  to become dead if it has 4 alive neighbors",
  true, this);
p_bDie[5]=new ParamBool("Die if 5","Is alive cell
  to become dead if it has 5 alive neighbors",
  true, this);
p_bDie[6]=new ParamBool("Die if 6","Is alive cell
  to become dead if it has 6 alive neighbors",
  true, this);
p_bDie[7]=new ParamBool("Die if 7","Is alive cell
  to become dead if it has 7 alive neighbors",
  true, this);
p_bDie[8]=new ParamBool("Die if 8","Is alive cell
  to become dead if it has 8 alive neighbors",
  true, this);
p_bDie[9]=new ParamBool("Die if 9","Is alive cell
```

```
      to become dead if it has 9 alive neighbors",
      true, this);
    p_bDie[10]=new ParamBool("Die if 10","Is alive cell
      to become dead if it has 10 alive neighbors",
      true, this);
    p_bDie[11]=new ParamBool("Die if 11","Is alive cell
      to become dead if it has 11 alive neighbors",
      true, this);
    p_bDie[12]=new ParamBool("Die if 12","Is alive cell
      to become dead if it has 12 alive neighbors",
      true, this);

    // Определение комментария, отображаемого в строке
    // состояния
    sComment[21]="Game of Life";
}

CALifeRules::~CALifeRules()
{
    for(unsigned char b=0;b<13;b++) delete p_bBorn[b];
    for(b=0;b<13;b++) delete p_bDie[b];
}
```

Потребность в обработчике инициализации (перегрузке функции `Initialize`) возникает только из-за наличия анализируемого параметра,

[21] Переменная `sComment` – член класса `CARules`.

значение которого должно быть известно перед выполнением первой итерации. В результате эта функция примет вид:

```
bool CALifeRules::Initialize()
{
    // Инициализацию следует производить, только если
    // параметр анализируется
    if (!p_iAlive.IsAnalyzed()[22]) return true;

    DATUM[23](CACrtsBoolDatum);
    CACell c; // Текущая клетка
    iq=0;

    for(int i=(int)datum->z.a1;
      i<=(int)datum->z.b1; i++)
        for(int j=(int)datum->z.a2;
          j<=(int)datum->z.b2; j++)
        {
            // Получение идентификатора текущей клетки
            // по декартовым координатам
            c=GetUnion()->Metrics->ToCell(i,j,0);

            if (datum->Get(c)) iq++;
```

[22] Функция `IsAnalyzed()` – член класса `Parameter`.

[23] Данное макроопределение создает локальную переменную `datum`, указатель на хранилище данных, с которым работает компонент правил, приводится его к типу указателя на класс, передаваемый макроопределению в качестве параметра.

```
        }
    // Присваивание значения анализируемому параметру
    p_iAlive.Set(iq);

    return true;
}
```

Необходимо отметить, что переменная-член `Zone z` класса `CADatum` описывает зону, значения состояний клеток из которой содержатся в хранилище. Эта переменная используется в приведенной выше функции для организации перебора всех клеток в цикле.

Далее приводится основная функция, осуществляющая итерации. Как отмечалось выше, даже для однопроцессорной системы, не предусматривающей распараллеливания вычислений, реализацию шага лучше осуществлять в функции `SubCompute`. Пример такого решения приводится ниже:

```
bool CALifeRules::SubCompute(Zone& z)
{
    DATUM(CACrtsBoolDatum);

    CACell c; // Текущая клетка
    // Массив для хранения всех соседей
    CACell neig[12];
    // Переменная, которая хранит количество соседей
    unsigned short ncount=
```

```
  GET_METRICS[24]->GetNeighboursCount();
// Количество живых соседей текущей клетки
unsigned char alive;
iq=0;

for(CACell i=z.a1; i<=z.b1; i++)
    for(CACell j=z.a2; j<=z.b2; j++)
    {
        alive=0;

        // Получение идентификатора текущей клетки
        // по декартовым координатам
        c=GetUnion()->Metrics->ToCell(i,j,0);

        // Получение всех соседей и размещение их в
        // массиве
        pUnion->Metrics->GetNeighbours(c,neig);

        // Подсчет числа живых соседей текущей
        // клетки
        for (int k=0; k<ncount; k++) {
            if (datum->Get(neig[k])) alive++;
        }
```

[24] Макроопределение GET_METRICS обеспечивает быстрый доступ к метрике, с которой работает данный компонент. Аналогично имеются макроопределения GET_GRID и GET_DATUM.

```
            // Применение функции переходов и
            // определение значения анализируемого
            // параметра
            if (datum->Get(c)) {
                datum->Set(c,!p_bDie[alive]->Get());
                if (!p_bDie[alive]->Get()) iq++;
            } else {
                datum->Set(c,p_bBorn[alive]->Get());
                if (p_bBorn[alive]->Get()) iq++;
            }
        }
    // Присваивание значения анализируемому параметру
    p_iAlive.Set(iq);

    return true;
}
```

Таким образом, пример простейшей реализации клеточного автомата "Жизнь" построен. Однако необходимо отметить, что перед его использованием компонент необходимо скомпилировать и уставить в среду, для этого требуется выбрать в главном меню "*Tools*" | "*Components Manager...*" (или воспользоваться горячей клавишей *<F9>*), затем нажать в появившемся окне кнопку "*Add...*" и открыть файл библиотеки компонента.

Отметим, что в настоящей главе не будут обсуждаться такие вопросы, как включение необходимых заголовочных файлов и подключение библиотек в силу громоздкости описания этой темы, а также для того, чтобы не обижать

читателей предположением, что они не смогут сделать этого самостоятельно. Базовые сведения по этому поводу приведены в разд. 3.4.

В заключение раздела отметим также, что вычисление анализируемых параметров в процессе выполнения шага автомата позволяет существенно сэкономить вычислительное время и избавиться от повторных переборов всех клеток решетки.

4.2.2. Реализация с зональной оптимизацией

Наблюдая за моделированием рассматриваемого клеточного автомата, нетрудно заметить, что почти всегда изменения состояний клеток происходят лишь в некоторых локализованных областях. При этом большие зоны, состоящие сплошь из мертвых клеток можно исключать из рассмотрения, так как на данном шаге жизнь в них зародиться не сможет. Отбрасывание таких областей будем называть зональной оптимизацией вычислений.

На рис. 28 изображен фрагмент решетки клеточного автомата, реализующего игру "Жизнь" с зональной оптимизацией в инструментальном средстве *CAME&L*. В эксперименте участвуют описанные в разд. 4.2.1 решетка, хранилище данных и метрика, входящие в набор стандартных компонентов, а также разрабатываемые ниже правила. Светло-серым выделены "мертвые" клетки, жизнь в которых не может появиться на следующем шаге в силу того, что у них нет живых соседей. В результате, их можно исключить из рассмотрения на данной итерации. Отбрасывание таких зон и называется зональной оптимизацией вычислений.

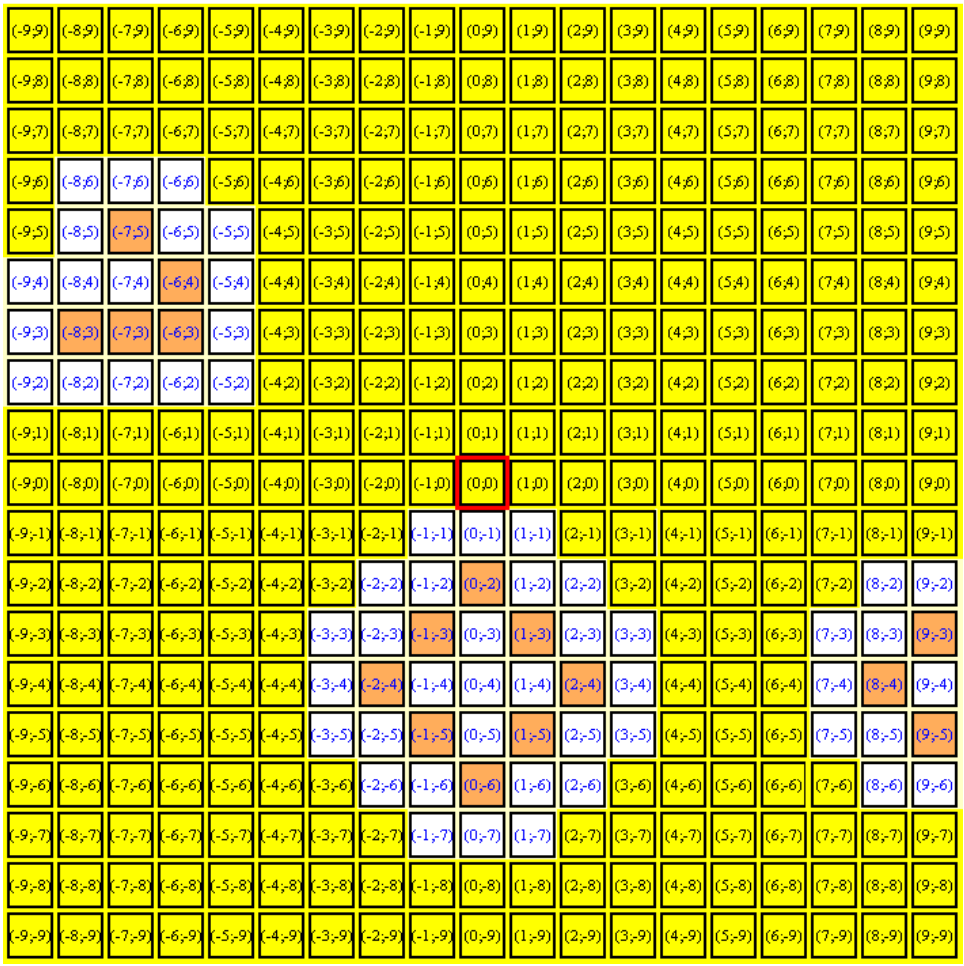


Рис. 28. Фрагмент решетки клеточного автомата, реализующего игру "Жизнь" с зональной оптимизацией

Для осуществления такой оптимизации в библиотеке *CADLib* имеется класс `OptZonal`. На каждой итерации объект этого класса запоминает клетки, состояние которых может повлиять на соседей на следующих итерациях (назовем их значимыми клетками). Пользуясь этой информацией, он формирует зоны, которые следует принимать в расчет при вычислениях на следующем шаге.

Приведем описание нескольких членов этого класса, которые будут использованы для реализации клеточного автомата "Жизнь" с зональной оптимизацией:

- `public OptZonal(unsigned int diffusion,unsigned int unite)` – конструктор. Параметр `diffusion` определяет величину "полей" зон, расстояние от крайних значимых клеток до границы зоны. Фактически, этот параметр должен быть равен радиусу окрестности клеток [91]. Параметр `unite` определяет наибольшее расстояние между клетками двух зон, при котором их следует объединять в одну.
- `public inline void Reset()` – функция инициализирует объект.
- `public void AddCell(CACell x,CACell y=0,CACell z=0)` – функция добавляет информацию о значимой клетке с соответствующими координатами вдоль трех осей.
- `public void NewStep()` – функция переключает указатель на рабочее хранилище информации о зонах между двумя внутренними хранилищами, а также проверяет, перекрываются ли зоны в них и т.п. Она должна вызываться перед каждым новым шагом для того, чтобы объект данного класса вычислял границы зон для следующего шага.
- `public bool ReleaseZone(Zone& z)` – функция возвращает через параметр `z` границы зоны, которую необходимо обработать. Ее следует вызывать до тех пор, пока результат, возвращаемый самой функцией, не станет равным `false`.

Для того чтобы воспользоваться зональной оптимизацией, в решение, разработанное в разд. 4.2.1, необходимо внести следующие изменения.

Во-первых, потребуется добавить переменную-член класса, реализующего компонент правил, `OptZonal oOpt`. В конструкторе класса `CALifeRules` необходимо вызвать конструктор для этой переменной с соответствующими параметрами, например, `oOpt(1,4)`, а также присвоить

значение `true` переменной `bFinalizeIfChanged`[25]. В результате изменение состояние решетки вручную приведет к финализации эксперимента. Это необходимо для обеспечения целостности информации о зонах, которые следует обрабатывать.

Во-вторых, в функции `Initialize` класса, реализующего правила, необходимо выполнить обнуление второго буфера данных, если включено двойное хранилище. Это необходимо для обеспечения корректности информации об обновляемых зонах. В результате функция примет вид:

```
bool CALifeRules::Initialize()
{
    // Инициализация оптимизатора
    oOpt.Reset();

    DATUM(CACrtsBoolDatum);
    CACell c; // Текущая клетка
    iq=0;

    for(CACell i=datum->z.a1; i<=datum->z.b1; i++)
        for(CACell j=datum->z.a2; j<=datum->z.b2; j++)
        {
            // Получение идентификатора текущей клетки
            // по декартовым координатам
            c=GetUnion()->Metrics->ToCell(i,j,0);
```

[25] Переменная `bFinalizeIfChanged` – член класса `CARules`.

```
            if (datum->Get(c)) {
                iq++;
                // Данная клетка живая, а
                // следовательно - значимая
                oOpt.AddCell(i,j);
            }
        }
    // Присвоение значения анализируемому параметру
    p_iAlive.Set(iq);

    // Обнуление второе буфера данных, если включено
    // двойное хранение
    if (datum->p_bDouble.Get()) {
        for(CACell i=datum->z.a1; i<=datum->z.b1; i++)
            for(CACell j=datum->z.a2;
              j<=datum->z.b2; j++)
            {
                c=GetUnion()->Metrics->ToCell(i,j,0);
                datum->Set(c,false);
            }
    }

    return true;
}
```

В-третьих, в функции `SubCompute` необходимо добавить вызов функции `oOpt.NewStep` и производить вычисления только для тех зон,

которые возвращаются функцией `oOpt.ReleaseZone`. Кроме этого, как и в обработчике инициализации, необходимо вызывать функцию `oOpt.AddCell` для всех оживших клеток.

В результате этих изменений производительность вычислений повысится, в среднем, на порядок.

4.2.3. Реализация для многопроцессорной системы

Идея выполнения клеточного автомата "Жизнь" на многопроцессорной вычислительной системе заключается в создании большого числа потоков [67], каждый из которых будет выполнять функцию `SubCompute` для некоторой части пространства. Потоки, в свою очередь, распределяются между процессорами. Итерация считается законченной, когда завершились вычисления во всех потоках.

На рис. 29 изображен фрагмент решетки клеточного автомата, реализующего игру "Жизнь" для многопроцессорной системы в инструментальном средстве *CAME&L*. В эксперименте участвуют описанные в разд. 4.2.1 решетка, хранилище данных и метрика, входящие в набор стандартных компонентов, а также разрабатываемые ниже правила. При выполнении на многопроцессорной системе вычисления на разных участках решетки возлагается на разные процессоры (такие участки на рис. 29 показаны прямоугольниками различных оттенков серого цвета). Так как процессоры имеют доступ к общей памяти, то заботиться о доступности состояний граничных клеток нескольким процессорам не требуется. Например, новое состояние клетки с координатами (–3; 7) вычисляет процессор №1, однако ее состояние требуется и процессору №2 для вычисления новых состояний клеток (–2; 6), (–2; 7) и (–2; 8) (при условии использования окрестности Мура).

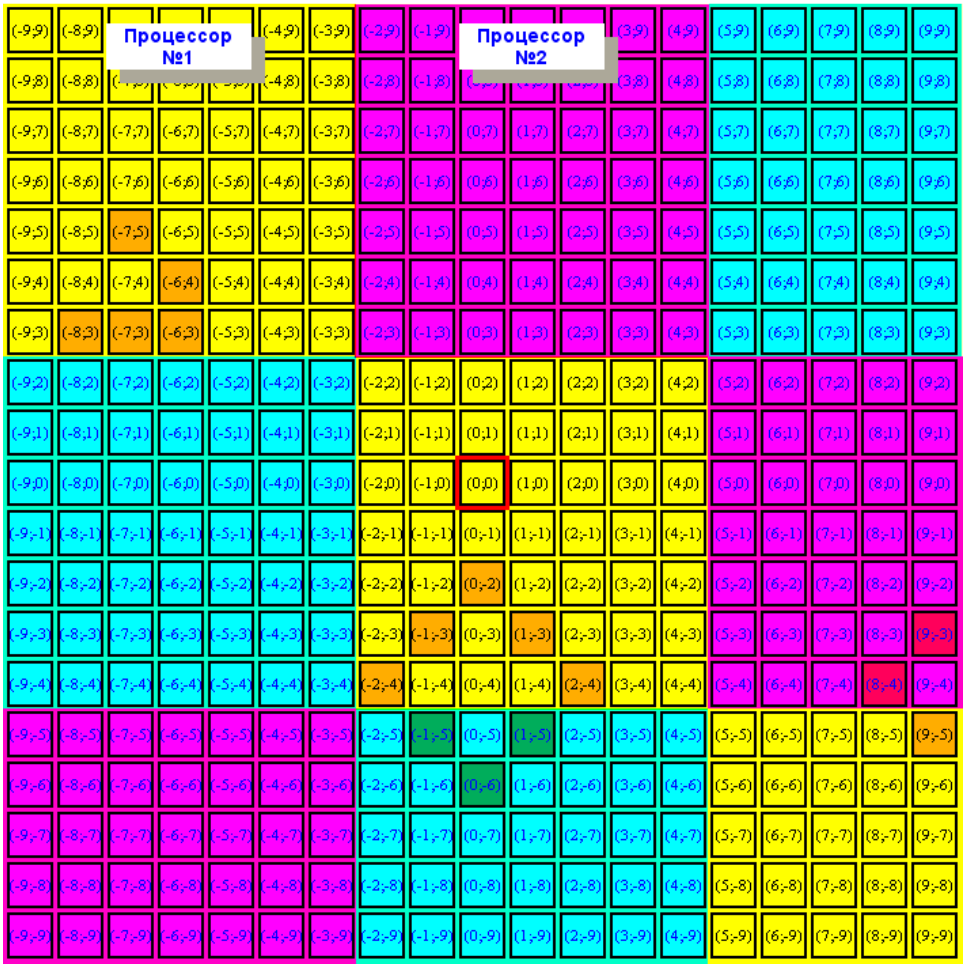


Рис. 29. Фрагмент решетки клеточного автомата, реализующего игру "Жизнь" для многопроцессорной системы

Для того чтобы рассматриваемый клеточный автомат выполнялся на многопроцессорной системе в решение, разработанное в разд. 4.2.1, необходимо внести ряд изменений.

Во-первых, удалить из функции `SubCompute` обнуление временной переменной `iq`, а также добавить присвоение нового значения анализируемому параметру `p_iAlive`.

Во-вторых, реализовать функцию `Compute` следующим образом (все используемые в данном фрагменте кода макроопределения реализованы в библиотеке *CADLib*).

```
bool CALifeRules::Compute()
{
    DATUM(CACrtsBoolDatum);

    // Создание счетчиков стартовавших и завершившихся
    // потоков
    COUNTER(started);
    COUNTER(finished);

    // Инициализация будущего значения анализируемого
    // параметра
    iq=0;

    // Разделение задачи на подзадачи для разных
    // потоков
    MULTIFOR2(k1,
      4*(int)pNetwork->GetThis()->uCPUsCount[26],
      datum->z.a1,datum->z.b1,
      k2,4*(int)pNetwork->GetThis()->uCPUsCount,
      datum->z.a2,datum->z.b2,started,finished);
    // Возможен и такой вариант:
    // MULTIFOR2_FIXED(k1,100,datum->z.a1,datum->z.b1,
    //   k2,100,datum->z.a2,
```

[26] Переменная `pNetwork->GetThis()->uCPUsCount` хранит число процессоров на данной машине.

```
    //   datum->z.b2,started,finished);

    // Ожидание совпадения счетчиков. Проверка каждые
    // 20 миллисекунд
    WAIT_COUNTER(started,finished,20);

    // Присвоение значения анализируемому параметру
    p_iAlive.Set(iq);

    return true;
}
```

Комментарии делают ясным назначение каждой строки приведенной выше функции. Отдельно следует рассмотреть только назначение макроопределений `MULTIFOR2` и `MULTIFOR2_FIXED`. Запишем их в общем виде.

Макроопределение `MULTIFORn(k1, count1, a1, b1, ..., kn, countn, an, bn, started, finished)` осуществляет деление *n*-мерной задачи (*n* лежит от одного до трех) на подзадачи, причем отрезок [`ai`, `bi`] вдоль *i*-й оси делится на `counti` частей. Параметр `ki` определяет имя создаваемой локальной переменной цикла вдоль *i*-й оси (*i* лежит от одного до *n*). Таким образом, для *n*=3 будет создано `count1*count2*count3` потоков. Последние два параметра передают имена счетчиков стартовавших и завершившихся потоков соответственно. Счетчиком здесь называется переменная, созданная с помощью макроопределения `COUNTER(<имя_переменной>)`. Использование двух

различных счетчиков (вместо одного) оправдывается рядом весомых соображений, обсуждение которых в настоящей работе нецелесообразно.

Макроопределение `MULTIFORn_FIXED(k1, size1, a1, b1, ..., kn, sizen, an, bn, started, finished)` осуществляет деление *n*-мерной задачи (*n* лежит от одного до трех) на подзадачи, причем отрезок `[ai, bi]` вдоль *i*-й оси делится на отрезки по `sizei` клеток (последний из них может быть меньше). Параметр `ki` определяет имя создаваемой локальной переменной цикла вдоль *i*-й оси (*i* лежит от одного до *n*). Таким образом, для *n*=3 будет создано `((b1-a1+1)/size1)*((b2-a2+1)/size2)*((b3-a3+1)/size3)` потоков (при делении здесь необходимо округлять результаты "вверх"). Последние два параметра передают имена счетчиков стартовавших и завершившихся потоков соответственно.

4.2.4. Реализация для вычислительного кластера

Для выполнения клеточного автомата в вычислительном кластере на всех машинах, входящих в распределенную систему, должна быть запущена среда из пакета *CAME&L*. Пусть на одной из машин (назовем ее главной) создается эксперимент (главный эксперимент), в процессе которого будут выполняться кластерные вычисления.

При выполнении инициализации эксперимента на главной машине всем остальным (или не всем, в зависимости от стратегии разделения задачи) посылается команда, создать вспомогательный эксперимент, который представляет собой часть главного, подзадачу. Причем состояние фрагмента решетки для данной подзадачи отправляется с учетом "полей", зон, которые не будут обрабатываться, но состояния клеток из которых могут потребоваться для вычисления новых значений для клеток основной области.

На рис. 30 изображен фрагмент решетки клеточного автомата, реализующего игру "Жизнь" для вычислительного кластера в инструментальном средстве *CAME&L*. В эксперименте участвуют описанные в разд. 4.2.1 решетка, хранилище данных и метрика, входящие в набор стандартных компонентов, а также разрабатываемые ниже правила. При выполнении на кластерной системе вычисления на разных участках решетки возлагается на разные процессоры (такие участки на рис. 30 показаны прямоугольниками различных оттенков серого цвета). При этом, если используется окрестность единичного радиуса, то столбец с координатой абсцисс равной единице будет относиться к полям подрешетки, хранимой на машине №1, а столбец с координатой абсцисс равной нулю будет относиться к полям подрешетки, хранимой на машине №2. Обмен полями необходимо выполнять, так как разные вычислительные узлы не имеют доступа к общей памяти. Если исходить из предположения, что в задаче используются константные граничные условия, то полей с других сторон подрешеток не будет.

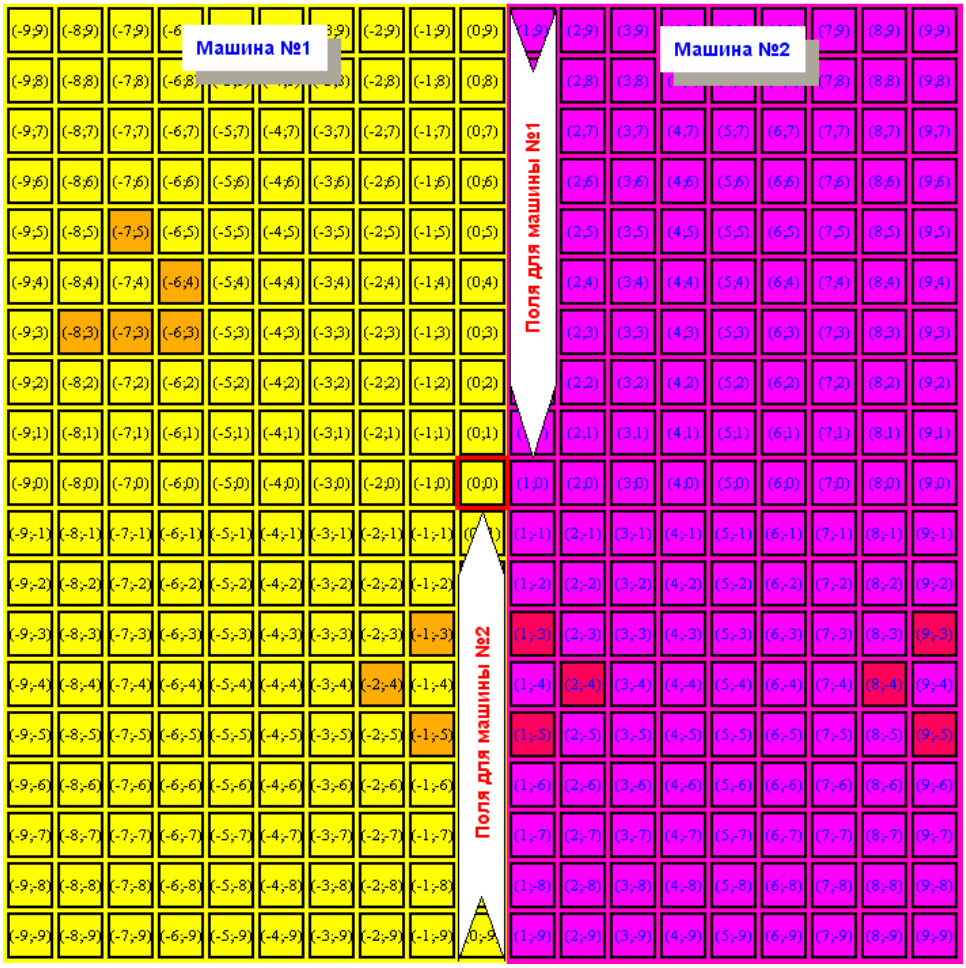


Рис. 30. Фрагмент решетки клеточного автомата, реализующего игру "Жизнь" для вычислительного кластера

Каждый шаг на главной машине состоит в отправке команд всем участникам эксперимента обновить поля (главная машина сообщает каждой, кто владеет информацией о состоянии клеток их полей) и произвести итерацию (шаг во вспомогательных экспериментах).

Необходимо отметить, что каждый эксперимент идентифицируется по так называемому дескриптору, в который входят строковое имя эксперимента (для его изменения можно выбрать в меню среды "*Modeling*" | "*Change Rules ID...*" или воспользоваться "горячим" сочетанием клавиш <*Shift*>+<*F12*>), адрес главной машины эксперимента и номер подзадачи. Данный дескриптор должен быть уникальным, чтобы однозначно идентифицировать

эксперимент. Неповторимость строкового имени легко обеспечить на каждой машине в отдельности. В результате обеспечивается уникальность пары имени и адреса главной машины. Номер используется при вычислениях в кластере. Для главного эксперимента он равен нулю. Все вспомогательные эксперименты получают уникальные номера последовательно, начиная с единицы.

Когда главная машина сочтет необходимым, она может запросить у остальных состояние их подрешеток (исключая поля). В результате состояние всей решетки будет "собрано" на одной машине. После этого можно продолжить эксперимент или прекратить его.

Реализация автомата "Жизнь" для вычислительного кластера, как и для многопроцессорной системы, не сильно отличается от решения, приведенного в разд. 4.2.1. Потребуется внести следующие изменения.

Для удобства введем еще один параметр, `p_iGather`, целое число, определяющее количество шагов, по истечении которого необходимо собрать состояние всей решетки на главной машине. В случае, если он равен нулю, сбор не осуществляется. В конструкторе класса `CALifeRules` необходимо будет вызвать конструктор этого параметра, а так же, как и в разд. 4.2.2, присвоить значение `true` переменной `bFinalizeIfChanged` для обеспечения целостности данных.

Функция `SubCompute` останется неизменной, однако функция `Initialize` существенно изменится. Кроме того, потребуется определить функции `Compute` и `Finalize`. Три новые функции будут иметь следующий вид:

```
bool CALifeRules::Initialize()
```

```
{
    // Проверка того, работает ли сетевой интерфейс
    if (!pNetwork->IsWorking()) return false;
    // Разделение задачи между машинами
    CreateCluster(1,1,true,2);
    // Ожидание, пока все участники кластера подтвердят
    // создание экспериментов и готовность
    return WaitForClusterReady();
}

void CALifeRules::Finalize()
{
    // Удаление экспериментов на всех участниках
    DestroyCluster();
}

bool CALifeRules::Compute()
{
    // Осуществление итерации
    return ComputeCluster(p_iGather.Get() &&
        (Step+1-pCluster->Step>=p_iGather.Get()));
}
```

Как следует из фрагмента кода, приведенного выше, кластерные вычисления организуются с помощью пакета *CAME&L* чрезвычайно просто в виду наличия богатого набора средств, содержащихся в классе `CARules`.

Во фрагменте кода, приводимом выше, при определении того, нужно ли собирать состояние решетки использовалось выражение "Step+1-pCluster->Step>=p_iGather.Get()". Здесь значение переменной Step – номер шага в главном эксперименте. К нему прибавлена единица, так как внутри функции Compute его увеличение еще не произошло. Значение переменной pCluster->Step – номер шага, на котором осуществлялась последняя сборка состояния всей решетки. Если эти две величины расходятся не меньше, чем на значение параметра p_iGather, то состояние всей решетки будет собрано.

4.2.5. Реализация для обобщенных координат

Компонент правил для клеточного автомата "Жизнь" в обобщенных координатах [91] даже проще, чем приведенный в разд. 4.2.1, так как задача становится одномерной. Функции Initialize и SubCompute примут следующий вид:

```
bool CALifeRules::Initialize()
{
    if (!p_iAlive.IsAnalyzed()) return true;

    DATUM(CAGenBoolDatum);
    iq=0;

    for(CACell c=datum->z.a1; c<=datum->z.b1; c++)
    {
        if (datum->Get(c)) iq++;
    }
```

```
    p_iAlive.Set(iq);

    return true;
}

bool CALifeRules::SubCompute(Zone& z)
{
    DATUM(CAGenBoolDatum);

    // Массив для хранения всех соседей
    CACell neig[12];
    // Переменная, которая хранит количество соседей
    unsigned short ncount=
      GET_METRICS->GetNeighboursCount();
    // Количество живых соседей текущей клетки
    unsigned char alive;
    iq=0;

    for(CACell c=z.a1; c<=z.b1; c++)
        {
            alive=0;

            // Получение всех соседей и размещение их в
            // массиве
            pUnion->Metrics->GetNeighbours(c,neig);

            // Подсчет числа живых соседей текущей
```

```
            // клетки
            for (int k=0; k<ncount; k++) {
                if (datum->Get(neig[k])) alive++;
            }

            // Применение функции переходов и
            // определение значения анализируемого
            // параметра
            if (datum->Get(c)) {
                datum->Set(c,!p_bDie[alive]->Get());
                if (!p_bDie[alive]->Get()) iq++;
            } else {
                datum->Set(c,p_bBorn[alive]->Get());
                if (p_bBorn[alive]->Get()) iq++;
            }
        }
    // Присваивание значения анализируемому параметру
    p_iAlive.Set(iq);

    return true;
}
```

В дополнение к этому требуется изменить условия совместимости данного компонента, так как он требует обобщенной метрики. Таким образом, выражение его условий примет вид "Data.bool.*&Metrics.2D.generalized.*".

Для постановки эксперимента теперь придется использовать обобщенную метрику (например, стандартный компонент "*Square Grids Generalized*") и соответствующее хранилище данных (стандартный компонент "*Booleans for Generalized*").

На рис. 31 изображен фрагмент решетки клеточного автомата, реализующего игру "Жизнь" для обобщенных координат в инструментальном средстве *CAME&L*. В эксперименте участвуют описанная в разд. 4.2.1 решетка, упомянутые выше хранилище данных и метрика, входящие в набор стандартных компонентов, а также разработанные правила.

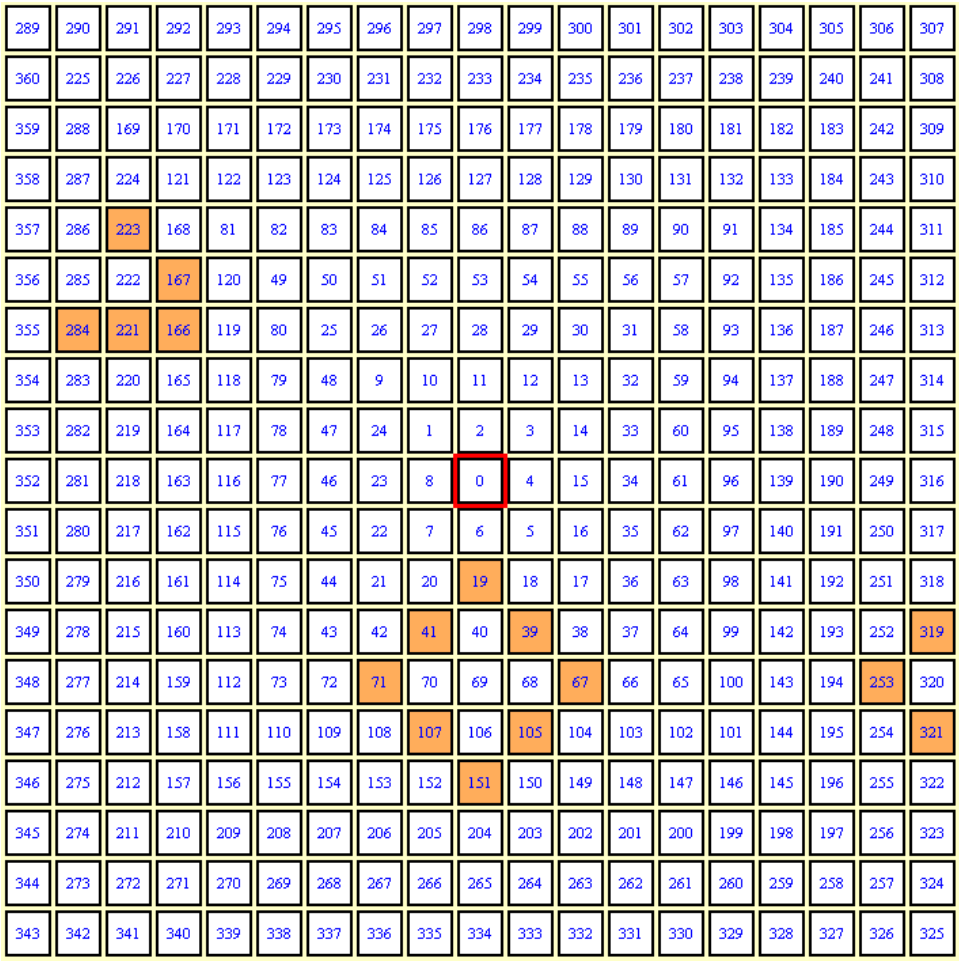


Рис. 31. Фрагмент решетки клеточного автомата, реализующего игру "Жизнь" для обобщенных координат

Необходимо отметить, что компонент, реализованный в разд. 4.2.1, тоже может работать с обобщенной метрикой, если указать условия совместимости "`Data.bool.*&Metrics.2D.*`". Этого не было сделано изначально только для ясности изложения.

Таким образом, описанный в настоящем разделе компонент лишь оптимизирован под работу с одномерными координатами, однако принципиально новой функциональности он не реализует.

4.2.6. Универсальная реализация, не зависящая от вычислительной платформы

Примеры, приведенные в предыдущих подразделах настоящего раздела, показали, что функция `SubCompute` практически не изменяется при переходе от одной вычислительной системы к другой (если не считать вариант для обобщенных координат, требующий ряда изменений). Таким образом, функциональность базового класса `CARules` может быть расширена так, что реализация правил для однопроцессорной, многопроцессорной и кластерной вычислительных систем будет осуществляться одним компонентом. Такой класс уже разработан. Он называется `CAUniRules`, однако не входит в библиотеку *CADLib* версии 1.0, но будет включен в один из ближайших релизов.

Для того чтобы данный компонент правил мог работать на произвольной вычислительной платформе, класс, реализующий его функциональность, необходимо унаследовать от базового класса `CAUniRules`.

На уровне базового класса уже реализована поддержка шести параметров компонента:

- `p_cPlatform` – параметр, позволяющий выбрать на какой платформе происходят вычисления: однопроцессорной, многопроцессорной или кластерной (по умолчанию выбрана однопроцессорная платформа).
- `p_iDivDim` – параметр имеет значение только в случае вычислений на многопроцессорной платформе. Определяет, вдоль какого количества измерений требуется производить разделение задачи на подзадачи: одного, двух или трех (по умолчанию используется значение два).
- `p_iDivX` – параметр имеет значение только в случае вычислений на кластерной платформе. Определяет наибольший размер подрешетки вдоль оси *X* (по умолчанию используется значение сто).
- `p_iDivY` – параметр имеет значение только в случае вычислений на кластерной платформе. Определяет наибольший размер подрешетки вдоль оси *Y* (по умолчанию используется значение сто).
- `p_iDivZ` – параметр имеет значение только в случае вычислений на кластерной платформе. Определяет наибольший размер подрешетки вдоль оси *Z* (по умолчанию используется значение сто).
- `p_iGather` – параметр имеет значение только в случае вычислений на кластерной платформе. Определяет период (количество шагов), по истечении которого состояния всех подрешеток будут собираться на вычислительном узле, запустившем задачу. Тем самым, формируется состояние всей решетки в целом (по умолчанию используется значение ноль. Поэтому автоматический сбор результирующего состояния выполняться не будет).

Помня о том, что реализуемые правила и без того имеют 26 параметров, их карта примет следующий вид:

```
PARAMETERS_COUNT(32)
BEGIN_PARAMETERS
    PARAMETER(0,&p_cPlatform)
    PARAMETER(1,&p_iDivDim)
    PARAMETER(2,&p_iDivX)
    PARAMETER(3,&p_iDivY)
    PARAMETER(4,&p_iDivZ)
    PARAMETER(5,&p_iGather)
    PARAMETER(6,p_bBorn[0])
    PARAMETER(7,p_bBorn[1])
    PARAMETER(8,p_bBorn[2])
    PARAMETER(9,p_bBorn[3])
    PARAMETER(10,p_bBorn[4])
    PARAMETER(11,p_bBorn[5])
    PARAMETER(12,p_bBorn[6])
    PARAMETER(13,p_bBorn[7])
    PARAMETER(14,p_bBorn[8])
    PARAMETER(15,p_bBorn[9])
    PARAMETER(16,p_bBorn[10])
    PARAMETER(17,p_bBorn[11])
    PARAMETER(18,p_bBorn[12])
    PARAMETER(19,p_bDie[0])
    PARAMETER(20,p_bDie[1])
    PARAMETER(21,p_bDie[2])
    PARAMETER(22,p_bDie[3])
    PARAMETER(23,p_bDie[4])
    PARAMETER(24,p_bDie[5])
```

```
    PARAMETER(25,p_bDie[6])
    PARAMETER(26,p_bDie[7])
    PARAMETER(27,p_bDie[8])
    PARAMETER(28,p_bDie[9])
    PARAMETER(29,p_bDie[10])
    PARAMETER(30,p_bDie[11])
    PARAMETER(31,p_bDie[12])
END_PARAMETERS
```

Также потребуется внести изменения в код в случае обращения к параметрам по индексам.

Если бы в разработанных правилах был реализован обработчик `OnParameterChanged`, то при изменениях первых шести параметров необходимо было бы вызывать аналогичный обработчик предка.

Функция `SubCompute` при этом будет иметь такой же вид, как, например, для случая простейшей реализации, позволяя, однако, организовывать вычисления, как на однопроцессорной, так и на многопроцессорной и кластерной платформах.

Таким образом, показано, что предложенное средство пригодно для проведения вычислительных экспериментов на однопроцессорной, многопроцессорной и кластерной вычислительных архитектурах, в том числе с применением оптимизации и разнообразных метрик.

4.3. Проектирование и реализация клеточного автомата для решения уравнения теплопроводности

Приведем пример решения физической задачи – уравнения теплопроводности с помощью инструментального средства *CAME&L*. Как известно, уравнение имеет вид:

$$\frac{\partial T}{\partial t} = a \cdot \Delta T, \tag{23}$$

где $a = \frac{\lambda}{\rho \cdot c_p}$;

λ – коэффициент теплопроводности материала;

ρ – плотность материала;

c_p – теплоемкость материала при постоянном давлении.

Каждая клетка автомата будет хранить значение температуры в соответствующей точке пространства, представляемое числом с плавающей точкой. Таким образом, автомат будет моделировать поле температур.

Для постановки эксперимента снова потребуется двумерная решетка из квадратов (хотя может использоваться и любая другая решетка), картезианская метрика. Помимо этого необходимо воспользоваться хранилищем вещественных данных для картезианских координат (стандартный компонент "*Doubles for Cartesians*"), а соответствующие правила требуется разработать.

На рис. 32 изображен фрагмент решетки клеточного автомата, реализующего решение уравнения теплопроводности без указания вычислительной архитектуры и оптимизации в инструментальном средстве *CAME&L*. В эксперименте участвуют описанные выше решетка, хранилище данных и метрика, входящие в набор стандартных компонентов, а также разрабатываемые ниже правила.

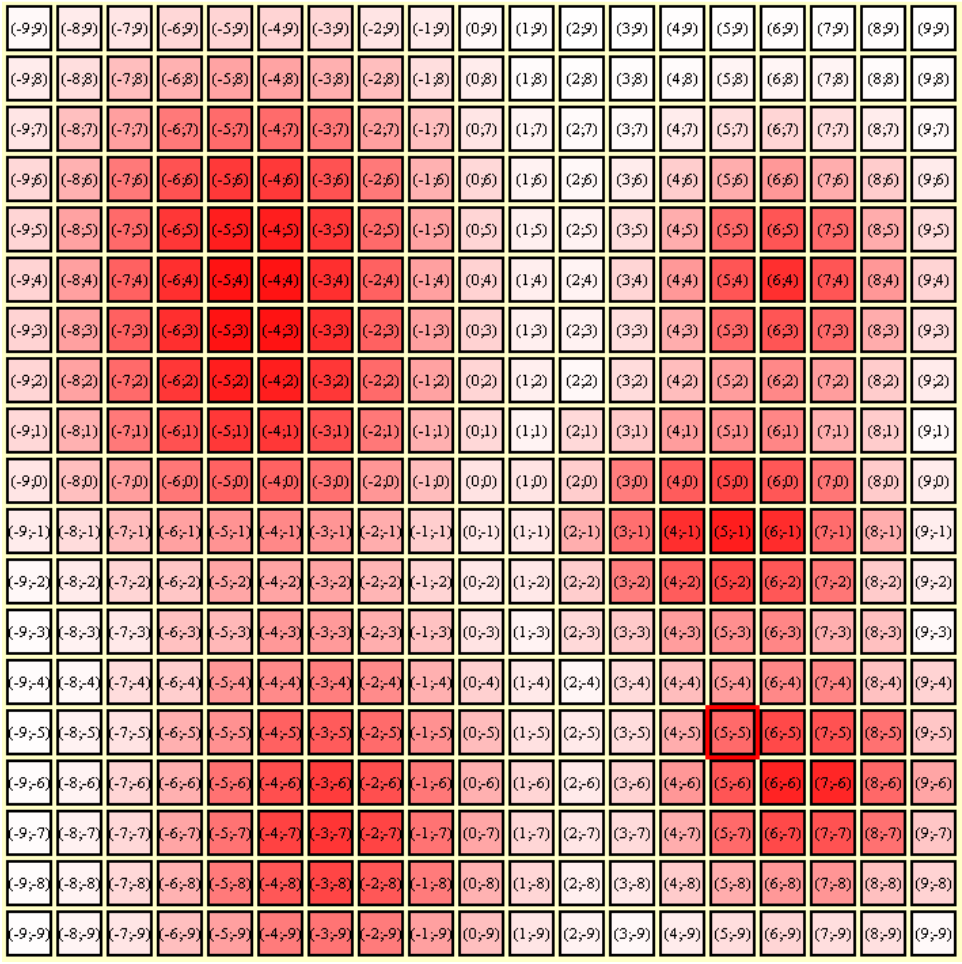


Рис. 32. Фрагмент решетки клеточного автомата, реализующего решение уравнения теплопроводности без указания вычислительной архитектуры и оптимизации

Класс, реализующий эти правила, назовем `CATCERules`. Для того, чтобы не загромождать реализацию физическими величинами будем считать параметр *a* из выражения (23), а также величины *dx*, *dy* и *dt*, равными единице. В результате в качестве функции переходов будем использовать следующее выражение в конечных разностях:

$$T'(x,y) = \frac{T(x+1,y) + T(x-1,y) + T(x,y+1) + T(x,y-1)}{4}. \qquad (24)$$

В случае если для хранилища данных будет включено хранение состояния на двух последовательных шагах, то можно считать, что в левой

части выражения (24) стоит температура на следующем временном шаге, по сравнению с теми значениями, что стоят в правой части. Если хранение на двух шагах не включено, то значения температуры в обеих частях выражения считаются для одного и того же временного шага, а тогда порядок обхода клеток при вычислении новых состояний становится существенным.

Предусмотрим в компоненте правил четыре анализируемых вещественных параметра: `p_dAverageT` – средняя температура на решетке, `p_dMaxT` – максимальная температура на решетке, `p_dMinT` – минимальная температура на решетке, `p_dAverSelT` – средняя температура среди выделенных пользователем в среде клеток.

Описание класса `CATCERules` будет иметь следующий вид:

```
class CATCERules:public CARules
{
public:
    // Конструктор и деструктор
    CATCERules();
    virtual ~CATCERules() {};

    // Определение имени, описания, иконки и условий
    // совместимости
    COMPONENT_NAME(TCE Solution (plain))
    COMPONENT_INFO(Thermal conductivity equation
      solution rules)
    COMPONENT_ICON(IDI_ICON)
    COMPONENT_REQUIRES(Data.double.*&
      Metrics.2D.cartesian.*)
```

```
    // Следующие функций класса CARules будут
    // переопределены
    virtual bool Initialize();
    virtual bool SubCompute(Zone& z);

public:
    // Анализируемые параметры
    ParamDouble p_dAverageT;
    ParamDouble p_dMaxT;
    ParamDouble p_dMinT;
    ParamDouble p_dAverSelT;

public:
    // Карта анализируемых параметров
    APARAMETERS_COUNT(4)
    BEGIN_APARAMETERS
        PARAMETER(0,&p_dAverageT)
        PARAMETER(1,&p_dMaxT)
        PARAMETER(2,&p_dMinT)
        PARAMETER(3,&p_dAverSelT)
    END_APARAMETERS
};
```

Функции `Initialize` и `SubCompute` могут быть реализованы следующим образом:

```
bool CATCERules::Initialize()
{
    // Инициализацию следует производить, только если
    // хотя бы один из параметров анализируется
    if (!p_dAverageT.IsAnalyzed() &&
      !p_dMaxT.IsAnalyzed() && !p_dMinT.IsAnalyzed() &&
      !p_dAverSelT.IsAnalyzed())
        return true;

    DATUM(CATypedDatum<double>);
    CACell c; // Текущая клетка
    double val=0.0; // Значение из текущей клетки

    // Переменные для вычисления значений анализируемых
    // параметров
    double avt=0.0,avselt=0.0;
    double maxt=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));
    double mint=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));

    for(CACell i=datum->z.a1; i<=datum->z.b1; i++)
        for(CACell j=datum->z.a2; j<=datum->z.b2; j++)
        {
            // Получение идентификатора текущей клетки
            // по декартовым координатам
            c=GET_METRICS->ToCell(i,j,0);
```

```
            val=datum->Get(c);

            avt+=val;
            if (val<mint) mint=val;
            if (val>maxt) maxt=val;
            if (p_dAverSelT.IsAnalyzed()) {
                if (find(GET_GRID->SelCells[27].begin(),
                  GET_GRID->SelCells.end(),c)!=
                  GET_GRID->SelCells.end())
                    avselt+=val;
            }
        }
    // Присваивание значений анализируемым параметрам
    p_dAverageT.Set(avt/(int)datum->z.GetVolume());
    p_dMaxT.Set(maxt);
    p_dMinT.Set(mint);
    if (GET_GRID->SelCells.size()) p_dAverSelT.Set(
      avselt/GET_GRID->SelCells.size()); else
        p_dAverSelT.Set(0.0);

    return true;
}
```

[27] Переменная `SelCells` класса `CAGrid` хранит список клеток, выделенных пользователем. Таким образом, данное условие, накладываемое на результат, возвращаемый функцией `find`, позволяет определить принадлежит ли клетка `c` к множеству выделенных или нет.

```
bool CATCERules::SubCompute(Zone& z)
{
    DATUM(CATypedDatum<double>);
    CACell c; // Текущая клетка
    double val=0.0; // Значение из текущей клетки

    // Переменные для вычисления значений анализируемых
    // параметров
    double avt=0.0,avselt=0.0;
    double maxt=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));
    double mint=datum->Get(GET_METRICS->ToCell(
        datum->z.a1,datum->z.a2,0));

    for(CACell i=z.a1; i<=z.b1; i++)
        for(CACell j=z.a2; j<=z.b2; j++)
        {
            // Получение идентификатора текущей клетки
            // по декартовым координатам
            c=GET_METRICS->ToCell(i,j,0);

            // Вычисление нового состояния клетки по
            // формуле (24)
            val=datum->Get(GET_METRICS->ToCell(i-1, j,
              0));
            val+=datum->Get(GET_METRICS->ToCell(i+1, j,
              0));
```

```
            val+=datum->Get(GET_METRICS->ToCell(i, j-1,
              0));
            val+=datum->Get(GET_METRICS->ToCell(i, j+1,
              0));
            val/=4.0;
            datum->Set(c,val);

            avt+=val;
            if (val<mint) mint=val;
            if (val>maxt) maxt=val;
            if (p_dAverSelT.IsAnalyzed()) {
                if (find(GET_GRID->SelCells.begin(),
                  GET_GRID->SelCells.end(),c)!=
                  GET_GRID->SelCells.end())
                    avselt+=val;
            }
        }
    // Присваивание значений анализируемым параметрам
    p_dAverageT.Set(avt/(int)z.GetVolume());
    p_dMaxT.Set(maxt);
    p_dMinT.Set(mint);
    if (GET_GRID->SelCells.size()) p_dAverSelT.Set(
      avselt/GET_GRID->SelCells.size()); else
        p_dAverSelT.Set(0.0);

    return true;
}
```

Как следует из приведенного выше фрагмента кода, реализации функций `Initialize` и `SubCompute` различаются только отсутствием в первой из них вычисления нового значения клетки и наличием проверки того, анализируется ли хотя бы один из параметров.

В данном разделе приведено простейшее решение уравнения теплопроводности. Однако, по аналогии с игрой "Жизнь", можно разработать также варианты для многопроцессорной и кластерной вычислительных систем, с поддержкой зональной оптимизации и обобщенных координат. Кроме того, нетрудно ввести параметры, определяющие физические характеристики системы, упомянутые в выражении (23).

Помимо этого, для большего соответствия физической сути задачи имеет смысл отредактировать спектр, отображающий соответствия между температурой и цветом (параметр "*Colors*" компонента хранилища данных).

Таким образом, показано, что предложенное средство пригодно для проведения вычислительных экспериментов на основе клеточных автоматов, функция переходов которых содержит вычисления с плавающей точкой и предоставляет инструментарий для наблюдения и анализа течения таких вычислительных экспериментов.

4.4. Проектирование и реализация вычислительного эксперимента для классификации структур, порождаемых одномерными двоичными клеточными автоматами из точечного зародыша

В настоящем разделе приводятся классификации структур, порождаемых одномерными двоичными клеточными автоматами из точечного зародыша [93, 94].

Под структурой понимается пошаговая последовательность состояний одномерной решетки – набор конфигураций автомата, располагаемых одна под другой.

В работе [2] приведены все 256 возможных структур, порождаемых в результате функционирования одномерного двоичного клеточного автомата, содержащего на нулевом шаге единственный точечный зародыш. Среди этих структур существуют не только одинаковые, но и эквивалентные относительно таких инвариантных операций, как зеркальное отображение, инвертирование, сдвиг на одну клетку и т.д.

В этой работе отсутствовала классификация этих структур, что не позволяло определить число разнотипных структур. Цель настоящего раздела состоит в демонстрации результатов применения разработанного инструментального средства *CAME&L* для проведения научного исследования – классификации структур, порождаемых одномерными клеточными автоматами из точечного зародыша, относительно инвариантных операций, а также подсчет числа классов при использовании различных инвариантов.

4.4.1. Задание функции переходов одномерного двоичного клеточного автомата

Будем рассматривать такие клеточные автоматы, окрестность каждой клетки которых состоит из ее правого и левого соседей, с учетом тороидальных граничных условий. В силу того, что в настоящей главе поведение автомата рассматривается при выполнении числа шагов, значительно меньшего ширины решетки, то краевые эффекты не проявляют себя.

Функцию переходов автомата, определяющую изменения каждого состояния клетки, целесообразно задавать таблицей. В ее левой части в

лексикографическом порядке перечисляются значения состояний ячеек с номерами i–1, i и i+1 на текущем шаге, а в правой – состояние ячейки с номером i на следующем шаге.

Эта таблица имеет восемь строк, каждой из которых соответствует одно из двух значений следующего состояния. Таким образом, существует 256 различных функций переходов. Это позволяет создать 256 клеточных автоматов указанного типа для любых начальных условий.

Каждому автомату может быть сопоставлен номер от 0 до 255, определяемый двоичной записью столбца значений в правой части таблицы переходов. Например, для функции $f(c_{i-1}, c_i, c_{i+1}) = c'_i$ с транспонированной правой частью $|0\ 0\ 0\ 1\ 0\ 1\ 1\ 1|^{\mathrm{T}}$ это число равно $0\cdot2^0+0\cdot2^1+0\cdot2^2+1\cdot2^3+0\cdot2^4+1\cdot2^5+1\cdot2^6+1\cdot2^7 = 232$. Приведенная функция называется "два и более из трех" и реализуется булевой формулой

$$c'_I = (c_{i-1} \vee c_i)\, c_{i+1} \vee c_{i-1}\, c_i. \tag{25}$$

Функция с номером $90 = 0\cdot2^0+1\cdot2^1+0\cdot2^2+1\cdot2^3+1\cdot2^4+0\cdot2^5+1\cdot2^6+0\cdot2^7$ реализуется формулой

$$c'_i = c_{i-1} \oplus c_{i+1}. \tag{26}$$

Обратим внимание, что эта функция описывает автомат с клетками без памяти, так как соответствующая ей формула не содержит переменную c_i в правой части.

Кроме автомата с номером 232, в качестве еще одного примера автомата из клеток с памятью, приведем автомат с номером $18 = 0\cdot2^0+1\cdot2^1+0\cdot2^2+0\cdot2^3+1\cdot2^4+0\cdot2^5+0\cdot2^6+0\cdot2^7$. Его функция переходов реализуется формулой

$$c'_i = !c_i\, (c_{i-1} \oplus c_{i+1}), \tag{27}$$

где ! – операция "инверсия".

Рассмотрим еще одну функцию переходов автомата из клеток с памятью, которая называется "один из трех" и имеет номер $22 = 0\cdot2^0+1\cdot2^1+1\cdot2^2+0\cdot2^3+1\cdot2^4+0\cdot2^5+0\cdot2^6+0\cdot2^7$. Она реализуется формулой

$$c'_i = !c_i\ (c_{i-1} \oplus c_{i+1}) \vee !c_{i-1}\ c_i\ !c_{i+1}. \qquad (28)$$

Функция "нечетность" имеет номер $150 = 0\cdot2^0+1\cdot2^1+1\cdot2^2+0\cdot2^3+1\cdot2^4+0\cdot2^5+0\cdot2^6+1\cdot2^7$ и реализуется формулой

$$c'_i = c_{i-1} \oplus c_i \oplus c_{i+1}. \qquad (29)$$

Функция с номером $60 = 0\cdot2^0+0\cdot2^1+1\cdot2^2+1\cdot2^3+1\cdot2^4+1\cdot2^5+0\cdot2^6+0\cdot2^7$ реализуется формулой

$$c'_i = c_{i-1} \oplus c_i. \qquad (30)$$

Обратим внимание, что эта функция, как и функция с номером 90, не зависит от одной из переменных, однако, описывает автомат из клеток с памятью, так как в этом случае переменная c_i входит в правую часть формулы.

Функция с номером $165 = 1\cdot2^0+0\cdot2^1+1\cdot2^2+0\cdot2^3+0\cdot2^4+1\cdot2^5+0\cdot2^6+1\cdot2^7$ реализуется формулой

$$c'_i = !\ (c_{i-1} \oplus c_{i+1})\ . \qquad (31)$$

Эта функция инверсна функции с номером 90, что подтверждается равенством 90 + 165 = 255.

4.4.2. Начальные условия

Структура, порождаемая клеточным автоматом при выбранной функции переходов, определяется его начальным состоянием. В настоящей главе каждый автомат на нулевом шаге содержат единственную клетку в состоянии "1".

4.4.3. Инвариантность относительно операции "равенство"

Одинаковое поведение клеточных автоматов с разными функциями переходов является основанием для их объединения в один класс. При этом используется, например, операция инвариантности, названная "равенство".

Один класс могут образовывать клеточные автоматы, которые имеют существенно различные функции переходов. Так тривиальное поведение (зародыш погибает на первом же шаге) демонстрируют автоматы с такими различными функциями, как, например, "тождественный ноль" и "два и более из трех". Всего в этот класс входит 16 функций: {0, 8, 32, 40, 64, 72, 96, 104, 128, 136, 160, 168, 192, 200, 224, 232}. Для каждой из этих функций характерно, что в правой части ее таблицы переходов нули находятся в тех строках, левые части которых не содержат единиц или содержат только одну единицу. Их столбцы значений имеют вид $|0\ 0\ 0\ ?\ 0\ ?\ ?\ ?|^{T}$, где "?" – "0" или "1".

Предполагается, что если два клеточных автомата рассматриваемого типа демонстрируют одинаковое поведение на первых пяти шагах, то они будут демонстрировать его и в дальнейшем.

4.4.3.1. Поведение типа "салфетка Серпинского"

Автомат с номером 90 порождает самовоспроизводящуюся структуру, называемую "салфетка Серпинского". Ниже приведена структура, порожденная этим автоматом, после первых восьми шагов:

```
0000000100000000
0000001010000000
0000010001000000
0000101010100000
0001000000010000
0010100000101000
0100010001000100
1010101010101010
```

На рис. 33 приведено изображение этой же структуры после 64 шагов.

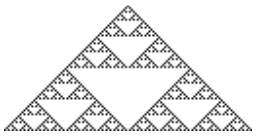
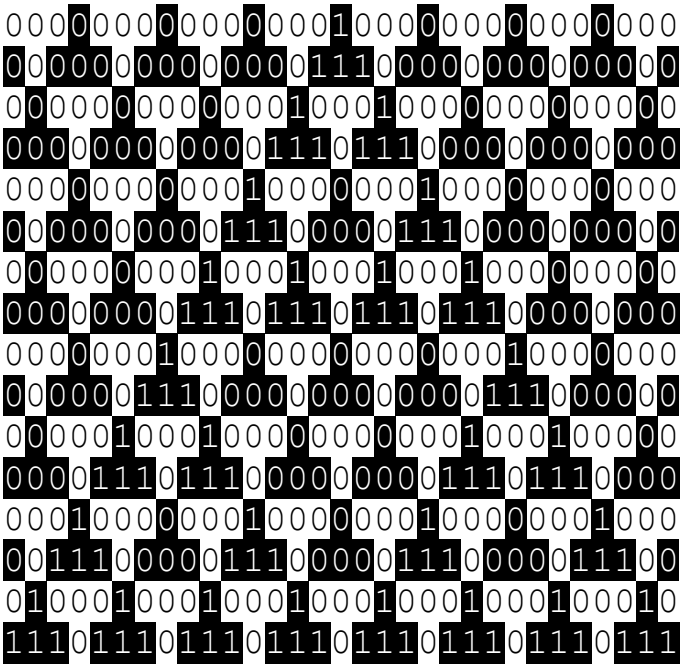


Рис. 33. Структура, порождаемая автоматом 90 после 64 шагов

Автомат с номером 18 порождает точно такую же структуру, как и автомат 90, так как правые части их функций переходов отличаются только в тех строках, левые части которых содержат комбинации с двумя единицами, которые не могут возникнуть при данных начальных условиях. Таким образом, в рассматриваемом случае автомат с клетками без памяти и с памятью имеют одинаковое поведение. При этом отметим, что формула, реализующая функцию переходов с номером 90, видимо, является простейшим математическим описанием "салфетки Серпинского" [68–70].

Отметим также, что класс "салфеток Серпинского" в целом содержит восемь автоматов с номерами {18, 26, 82, 90, 146, 154, 210, 218} = *C*.

Обратим внимание, что в этот класс не входит автомат с номером 22. Он один в своем классе и порождает разновидность "салфетки Серпинского", состоящую из нулевых и единичных "пьедесталов". В приводимой ниже, порождаемой им структуре, смежные пьедесталы выделены разным цветом:



Если нулевые пьедесталы заменить нулем, а единичные – единицей, то получится "салфетка Серпинского", рассмотренная выше.

На рис. 34 приведено изображение порождаемой автоматом с номером 22 структуры после 64 шагов.

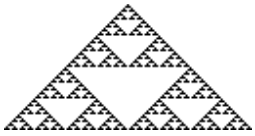


Рис. 34. Структура, порождаемая автоматом 22 после 64 шагов

Также уникален автомат с номером 150, порождающий еще одну самоподобную структуру:

```
0000000010000000
0000000111000000
0000001010100000
0000011010110000
0001000100010000
0011101110111000
0101000100010100
1101101110110110
```

На рис. 35 приведено изображение этой же структуры после 64 шагов.

Рис. 35. Структура, порождаемая автоматом 150 после 64 шагов

4.4.3.2. Поведение типа "двоичный треугольник Паскаля"

Двоичный треугольник Паскаля [69] имеет следующий вид:

```
      1
     1 1
    1 0 1
   1 1 1 1
  1 0 0 0 1
 1 1 0 0 1 1
```

Деформируем этот треугольник так, как показано ниже:

```
1
1 1
1 0 1
1 1 1 1
1 0 0 0 1
1 1 0 0 1 1
```

Автомат с номером 60 порождает деформированный двоичный треугольник Паскаля (рис. 36), который один в своем классе.

Рис. 36. Структура, порождаемая автоматом 60 после 64 шагов

В завершение раздела отметим, что число классов инвариантности относительно операции "равенство" – 143.

Отметим, что 115 из указанных выше классов содержат по одному представителю. Все остальные структуры распределяются между 28 классами, в каждом из которых по два и более элемента. Ниже приводятся эти 28 классов в виде нумерованного списка, каждый элемент которого содержит номера автоматов, порождающих структуры, входящие в один класс:

1) 0 8 32 40 64 72 96 104 128 136 160 168 192 200 224 232
2) 1 33
3) 2 10 34 42 66 74 98 106 130 138 162 170 194 202 226 234
4) 3 35
5) 4 12 36 44 68 76 100 108 132 140 164 172 196 204 228 236
6) 6 38 134 166
7) 7 19 21 23 31 55 63 87 95 119 127
8) 11 43 47
9) 14 46 142 174
10) 16 24 48 56 80 88 112 120 144 152 176 184 208 216 240 248
11) 17 49
12) 18 26 82 90 146 154 210 218
13) 20 52 148 180
14) 28 156
15) 50 58 114 122 178 186 242 250

16) 70 198
17) 81 113 117
18) 84 116 212 244
19) 129 161
20) 139 171
21) 151 159 183 191 215 223 233 235 237 239 247 249 251 253 255
22) 173 189
23) 203 217 219
24) 206 238
25) 209 241
26) 220 252
27) 222 254
28) 229 231

По аналогии с классификацией булевых функций [13], например, *PN*-классификацией (от английских слов "*Permutation*" – "перестановка" и "*Negation*" – "отрицание"), назовем проведенное выше объединение в классы *E*-классификацией (от "*Equality*" – "равенство").

4.4.4. Инвариантность относительно операций "равенство" и "инверсия"

Приведем в качестве примера структуру, инверсную структуре, порождаемой автоматом с номером 90. Она генерируется автоматом с номером 165 (рис. 37).

Рис. 37. Структура, порождаемая автоматом 165 после 64 шагов

Отметим, что в настоящей главе инверсность рассматривается с точностью до нулевого шага.

Указанная взаимосвязь "прямых" и "инверсных" структур может являться основанием для их объединения в один класс относительно операции инвариантности "инверсия". Однако, проведение классификации относительно только операций инвариантности "инверсия" нецелесообразно. Это связано с тем, что, во-первых, в каждом классе не могло бы содержаться больше двух экземпляров, а, во-вторых, многие экземпляры принадлежали бы более чем одному классу. Поэтому общее число классов могло бы существенно возрасти.

В связи с изложенным предлагается в один класс объединять структуры, инвариантные относительно двух операций: "равенство" и "инверсия".

При использовании этих операций в один класс будут входить, например, девять автоматов с номерами {18, 26, 82, 90, 146, 154, 165, 210, 218} = $C \cup \{165\}$.

Обратим внимание, что в рассмотренном в настоящем разделе примере инверсны не только поведения автоматов, но и их функции, что подтверждается тем, что сумма их номеров 90+165 равна 255.

При этом отметим, что инверсии функций 18, 26, 82, 146, 154, 210 и 218 не порождают инверсных структур для класса *C*. В частности, функция 37 (инверсия функции 218) порождает малоинтересное поведение (рис. 38).

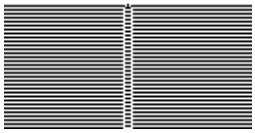


Рис. 38. Структура, порождаемая автоматом 37 после 64 шагов

В завершение раздела отметим, что число классов инвариантности относительно операций "равенство" и "инверсия" – 135. Назовем эту классификацию *EI*-классификацией (от "*Equality*" и "*Inverse*" – "инверсия").

4.4.5. Инвариантность относительно операций "равенство" и "зеркальное отображение"

Приведем в качестве примера структуру, зеркальную (относительно вертикальной оси) к структуре, порождаемой автоматом с номером 60. Эта структура порождается автоматом с номером 102 (рис. 39).

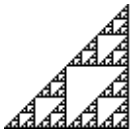


Рис. 39. Структура, порождаемая автоматом 102 после 64 шагов

Число классов инвариантности относительно операций "равенство" и "зеркальное отображение" – 89.

Назовем эту классификацию *EM*-классификацией (от "*Equality*" и "*Mirror*" – "зеркало").

В качестве примера приведем *EM*-класс с максимальным числом представителей: {2, 10, 16, 24, 34, 42, 48, 56, 66, 74, 80, 88, 98, 106, 112, 120, 130, 138, 144, 152, 162, 170, 176, 184, 194, 202, 208, 216, 226, 234, 240, 248}. В нем 32 элемента, так как этот класс является объединением двух *E*-классов:

- {2, 10, 34, 42, 66, 74, 98, 106, 130, 138, 162, 170, 194, 202, 226, 234} – "левая диагональ", для которой столбцы значений таблиц переходов имеют вид $|0\ 1\ 0\ ?\ 0\ ?\ ?\ ?|^T$;
- {16, 24, 48, 56, 80, 88, 112, 120, 144, 152, 176, 184, 208, 216, 240, 248} – "правая диагональ", для которой столбцы значений таблиц переходов имеют вид $|0\ 0\ 0\ ?\ 1\ ?\ ?\ ?|^T$.

4.4.6. Инвариантность относительно операций "равенство", "инверсия" и "зеркальное отображение"

Приведем в качестве примера четыре структуры с номерами {60, 102, 153, 195}, принадлежащие одному классу относительно рассматриваемого набора операций (рис. 40).

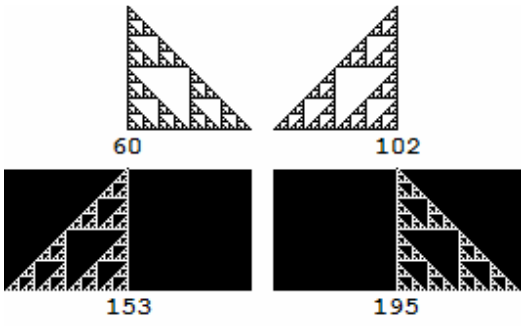


Рис. 40. Структуры, порождаемые автоматами 60, 102, 153 и 195 после 64 шагов

Этот рисунок показывает, что структуры 60 и 102 зеркальны, также как и структуры 153 и 195. При этом структуры 60 и 195 инверсны, как и структуры 102 и 153.

Число классов инвариантности относительно операций "равенство", "инверсия" и "зеркальное отображение" – 83.

Назовем эту классификацию *EIM*-классификацией (от "*Equality*", "*Inverse*" и "*Mirror*"). Этот вариант содержит наименьшее число классов из рассмотренных.

4.4.7. Инвариантность относительно операций "равенство" и "инверсно-зеркальное отображение"

Приведем в качестве примера те же четыре структуры с номерами {60, 102, 153, 195}. При классификации относительно указанного выше набора операций один класс образует пара автоматов {60, 153}, а другой – пара {102, 195}.

Назовем эту классификацию *E*(*I*+*M*)-классификацией (от "*Equality*", "*Inverse*" и "*Mirror*").

Число классов инвариантности относительно операций "равенство и "инверсно-зеркальное отображение" – 135, что совпадает с количеством *EI*-классов. В действительности, эти две классификации порождают похожие классы. При этом только 40 структур классифицируются по-разному. Они попадают в разные четыре класса при *EI*- и *E*(*I*+*M*)-классификациях. Различия между этими четырьмя классами показаны в таблице 10.

Таблица 10. Различия между *EI*- и *E*(*I*+*M*)-классификациями

***EI*-классификация**	***E*(*I*+*M*)-классификация**
2, 10, 34, 42, 66, 74, 98, 106, 130, 138, 162, 170, 173, 189, 194, 202, 226, 234	2, 10, 34, 42, 66, 74, 98, 106, 130, 138, 162, 170, 194, 202, 226, 229, 231, 234
16, 24, 48, 56, 80, 88, 112, 120, 144, 152, 176, 184, 208, 216, 229, 231, 240, 248	16, 24, 48, 56, 80, 88, 112, 120, 144, 152, 173, 176, 184, 189, 208, 216, 240, 248
60, 195	60, 153
102, 153	102, 195

В завершение раздела отметим, что все рассмотренные выше операции инвариантности могут быть отнесены к операциям с прямым наложением (без сдвигов). В столбце "*L*0" таблицы 11 приведены количества классов для рассмотренных вариантов классификаций.

Таблица 11. Число классов при разных вариантах классификации без сдвигов

	***L*0**	***L*1**	***L*2**	***L*3**	***L*4**	***L*5**	***L*6**
E	143	134	129	128	128	128	127
EM	89	83	81	80	80	80	79
EI	135	124	120	119	119	119	118
***E*(*I*+*M*)**	135	124	120	119	119	119	118
EIM	83	76	74	73	73	73	72

4.4.8. Классификации с учетом сдвигов на одну клетку

Введем новые операции инвариантности: "равенство со сдвигом", "инверсия со сдвигом", "зеркальное отображение со сдвигом" и "инверсно-зеркальное отображение со сдвигом". При выполнении этих операций взаимное соответствие двух структур проверяется не только прямым наложением, но и наложением со сдвигами на одну клетку по вертикали, горизонтали и диагоналям. При этом эквивалентными считаются структуры, которые соответствуют друг другу непосредственно или с учетом любого из этих смещений.

Классификации относительно операций со сдвигами будем называть так же, как классификации без сдвигов, но с добавлением суффикса "*O*" (от "*Offset*" – "сдвиг").

В качестве примера приведем структуры 20 и 155 (рис. 41), эквивалентные друг другу относительно операции "инверсно-зеркальное

отображение со сдвигом". Отметим, что, в отличие от рисунков, приведенных выше, структуры на рис. 41 приведены с увеличением.

Рис. 41. Структуры, порождаемые автоматами 20 и 155 после 16 шагов

Структура 155 получается из структуры 20 посредством зеркального отображения, инвертирования и сдвига на одну клетку вниз.

Рассматривать операции инвариантности со сдвигами больше, чем на одну клетку, нет смысла, так как цель настоящей работы состоит в объединении в классы автоматов со сходным поведением. При этом зародыш может за один шаг повлиять лишь на клетки, получаемые из него самого сдвигом на одну клетку и не более того, в силу размеров окрестности.

Классификации со сдвигом позволяют уменьшить число полученных классов, по сравнению с классификациями без сдвигов.

Так, например, если *EIM*-классификация выделяет 83 класса, то *EIMO*-классификация сокращает их количество до 56 классов. Эти классы приводятся ниже:

1) 0 8 32 40 64 72 96 104 128 136 151 159 160 168 183 191 192 200 215 223 224 232 235 237 239 247 249 251 253 255
2) 1 33 123
3) 2 10 16 24 34 42 48 56 66 74 80 88 98 106 112 120 130 138 144 152 162 170 173 175 176 184 187 189 194 202 208 216 226 229 231 234 240 243 245 248
4) 3 17 35 49 59 115

5) 4 12 36 44 68 76 100 108 132 140 164 172 196 203 204 207 217 219 221 228 236
6) 5
7) 6 20 38 52 134 148 155 166 180 211
8) 7 19 21 23 31 55 63 87 95 119 127
9) 9 65 111 125
10) 11 43 47 81 113 117
11) 13 69 79 93
12) 14 46 84 116 139 142 143 171 174 209 212 213 241 244
13) 15 85
14) 18 26 82 90 146 154 165 167 181 210 218
15) 22
16) 25 61 67 103
17) 27 39 53 83
18) 28 70 156 157 198 199
19) 29 71
20) 30 86 135 149
21) 37 91
22) 41 97
23) 45 101
24) 50 58 114 122 178 179 186 242 250
25) 51
26) 54 147
27) 57 99
28) 60 102 153 195
29) 62 118
30) 73

31) 75 89
32) 77
33) 78 92
34) 94
35) 105
36) 107 121
37) 109
38) 110 124
39) 126 129 161
40) 131 145
41) 133
42) 137 193
43) 141 197
44) 150
45) 158 214
46) 163 177
47) 169 225
48) 182
49) 185 227
50) 188 230
51) 190 246
52) 201
53) 205
54) 206 220 238 252
55) 222 254
56) 233

Количество классов, получаемых при использовании инвариантных операций со сдвигами, приведено в таблице 12 в столбце "*OL*0".

Таблица 12. Число классов при разных вариантах классификации со сдвигами

	***OL*0**	***OL*1**	***OL*2**	***OL*3**	***OL*4**	***OL*5**	***OL*6**
E	125	118	118	117	117	117	117
EM	78	75	75	74	74	74	74
EI	88	86	86	85	85	85	85
***E*(*I*+*M*)**	88	85	85	84	84	84	84
EIM	56	55	55	54	54	54	54

4.4.9. Классификации с учетом погрешностей

Соответствие между структурами относительно операций инвариантности имеет смысл проверять не только строго, но и допуская небольшое число несоответствий, так как отличия в нескольких клетках не является основанием для того, чтобы не считать порождаемые структуры родственными.

В качестве примера приведем структуры 233 и 235 (рис. 42), отличающиеся лишь пятью клетками.

Рис. 42. Структуры, порождаемые автоматами 20 и 155 после 16 шагов

Классификации относительно операций с учетом погрешностей будем называть так же, как классификации без них, но с добавлением суффикса "*Ln*"

(от "*Lapse*" – "небольшая ошибка"), где *n* – наибольшее допустимое количество несовпадений.

В таблице 11 и 12 в столбцах "*L*0" – "*L*6" и "*OL*0" – "*OL*6", соответственно, приводится количество классов для различных классификаций с числом несовпадений от нуля до шести.

Обратим внимание, что, если сдвиги не использовать, то значения в строках "*EI*" и "*E*(*I*+*M*)" таблице 11 совпадают в каждом столбце, в то время как в таблице 12 они различны, причем значение из первой строки не меньше значения из второй.

Таким образом, впервые проведена классификация всех различных типов поведения одномернух двоичных клеточных автоматов из точечного зародыша. Показано, что существуют автоматы с клетками без памяти и с памятью, демонстрирующие одинаковое поведение. Исследования этого класса структур является важным, так как это – простейшие структуры, многие их которых демонстрируют сложное поведение (в том числе, самовоспроизведение).

4.4.10. Дополнительные материалы

На сайтах [**Error! Bookmark not defined.**, 71] опубликованы следующие материалы:

- 256 изображений порождаемых структур после 64 шагов (в формате *Portable Network Graphic*);
- 256 изображений порождаемых структур после 16 шагов при четырехкратном увеличении (в формате *Portable Network Graphic*);
- 256 текстовых представлений порождаемых структур после 64 шагов;
- 70 вариантов классификации порождаемых структур;
- порождаемые структуры и некоторые классификации в таблице.

Кроме того, в приложении 1 приводится исходный код компонента правил для инструментального средства *CAME&L*, генерирующий структуры рассматриваемого типа, а в приложении 2 размещена таблица, содержащая все порождаемые структуры и некоторые варианты классификаций.

4.5. Автоматизация проектирования программного обеспечения *FPGA*-систем с помощью инструментального средства *CAME&L*

В ходе написания настоящей диссертации была выполнена работа по созданию программного обеспечения для организации вычислительных экспериментов на *FPGA*-системах [72–74] ("*Field Programmable Gate Array*" – "программируемая вентильная матрица") при помощи инструментального средства *CAME&L* [75]. Тем самым было расширено множество вычислительных архитектур, на которых можно проектировать и выполнять вычисления с помощью обсуждаемого инструментального средства.

FPGA-процессоры были изобретены в 1984 году Россом Фриманом (*Ross Freeman*), одним из основателей корпорации *Xilinx*. Как и многие другие переконфигурируемые процессоры, устройства этого типа представляют собой объединение *RISC*-процессора с двумерный массивом обрабатывающих элементов, каждый из которых, в свою очередь, состоит из процессора, памяти и высокоскоростных интерфейсов ввода-вывода, обеспечивающих преимущества тандемной обработки. При этом выделенный *RISC*-процессор занимается, например, переключением "основного" и "вспомогательного" контекстов, размещением контекстов в обрабатывающих элементах, а также другими "хозяйственными" вопросами [76].

Условное изображение структурной схемы вычислительного массива *FPGA*-процессора приведено на рис. 43.

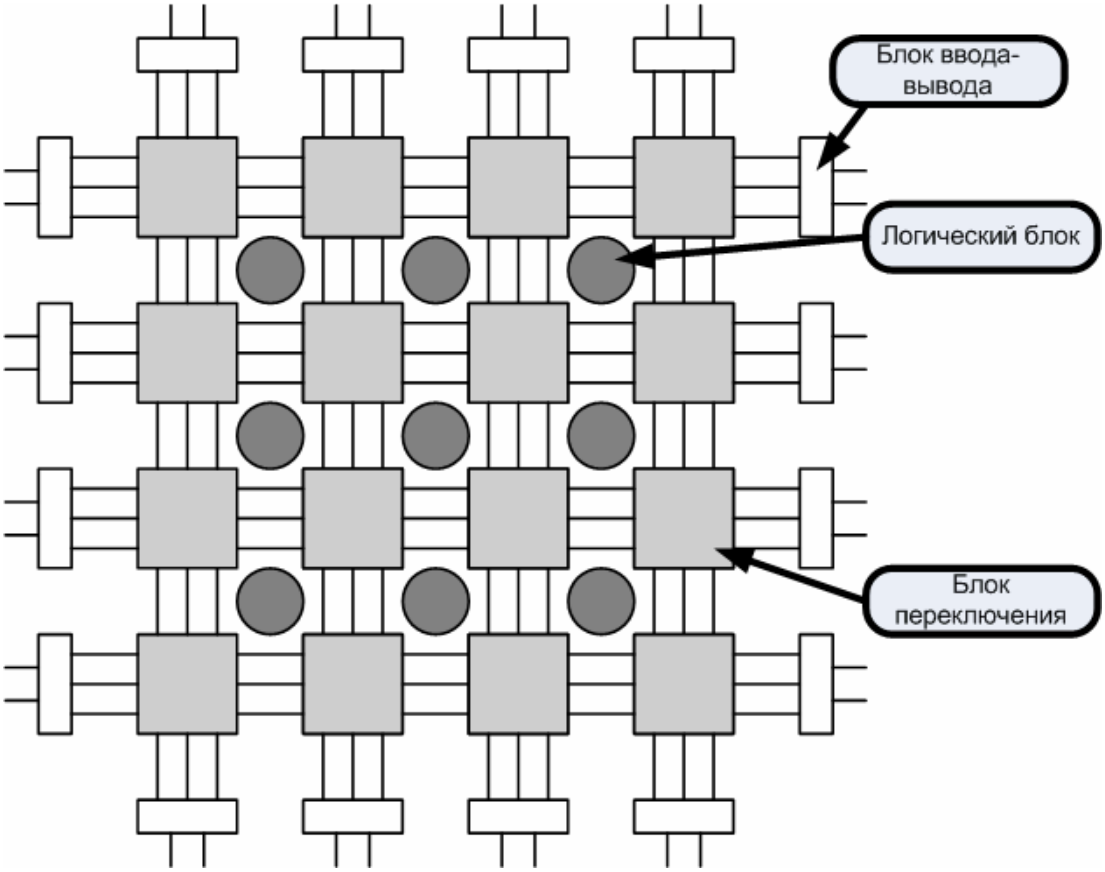


Рис. 43. Схематическое представление вычислительного массива *FPGA*-процессора

Идея использования *FPGA*-процессоров для реализации вычислительных экспериментов с помощью клеточных автоматов не нова [77, 78], так как применение этих систем позволяет существенно повысить эффективность вычислений.

В разд. 1.9 упоминалось инструментальное средство *CDL* [45, 78], которое позволяло использовать *FPGA*-системы для вычислений на основе клеточных автоматов. Однако оно не предоставляло богатый инструментарий для описания и проведения экспериментов. Кроме того, необходимо отметить, что и сами вентильные матрицы сильно развились и усовершенствовались с тех пор. Принимая во внимание их чрезвычайно низкую относительную стоимость (несколько сотен долларов США),

применение этих систем для научных вычислений выглядит весьма перспективным.

В настоящем разделе описывается набор средств преобразования исходного кода компонента-правил для инструментального средства *CAME&L* в выполнимый файл для *FPGA*-процессора, а также вспомогательный компонент правил, загружаемый в среду и позволяющий управлять экспериментом. Это позволило расширить число архитектур, на которых можно проводить вычислительные эксперименты с помощью предложенного инструментального средства.

Структурная схема процесса преобразования и примения разработанных средств приводится на рис. 44.

Необходимо отметить, что пунктирные прямоугольники на рис. 44 представляют собой обрабатываемые файлы, которыми обмениваются инструментальные средства, изображенные в виде прямоугольников со сплошными границами. При этом инструментальные средства пронумерованы для того, чтобы на них можно было ссылаться в следующем тексте.

На входе имеется исходный код класса, реализующего компонент правил для инструментального средства *CAME&L*. Для его приведения к виду, пригодному для дальнейшей обработки, был создан препроцессор ("1" на рис. 44). Эта программа осуществляет преобразование конструкций языка *C++* и специфических выражений библиотеки *CADLib* (таких как, обращение к клетке, к ее соседям и т.п.), а также вычислительного ядра правил в форму, пригодную для взаимодействия с *FPGA*-системой на основании имеющегося у нее шаблона. Большинство инструкций заменяется вызовами коммуникационных функций и функций преобразования данных к виду, пригодному для работы с *FPGA*-процессорами.

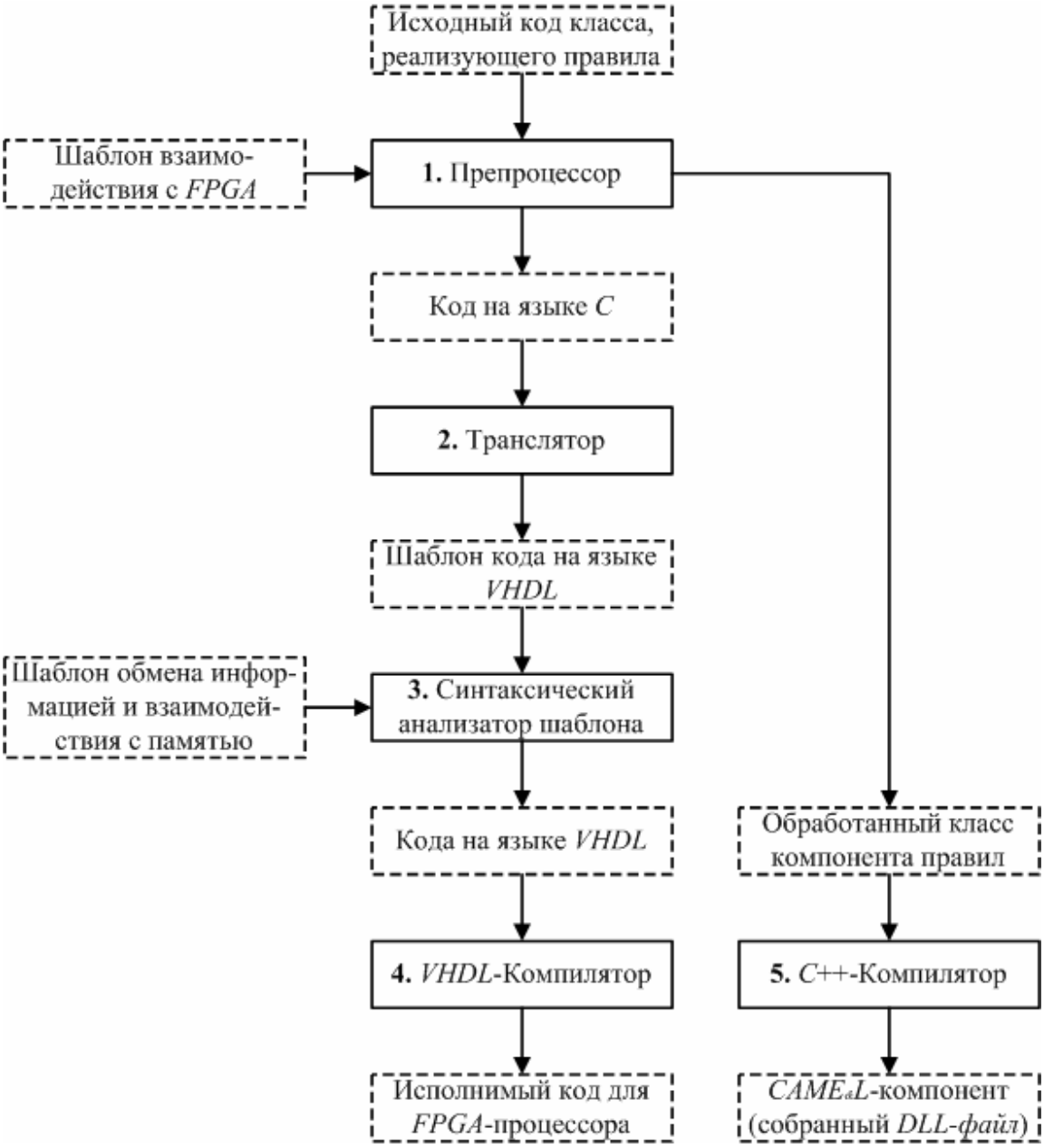


Рис. 44. Схема процесса преобразования компонента клеточного автомата для инструментального средства *CAMEaL* в выполнимый файл для *FPGA*-системы и управляющий компонент правил

При этом такие специфические параметры, как, например, тип окрестности, описываются директивами `#pragma` [62], анализируемыми препроцессором. Результат работы этой программы, с одной стороны, подается на вход

транслятору ("2" на рис. 44), а с другой собирается обычным компилятором ("5" на рис. 44) (как правило, используется средство *Microsoft Visual C++* 6) в библиотеку управляющего *CAME&L* компонента-правил, который будет взаимодействовать с *FPGA*-системой и координировать эксперимент.

Транслятор ("2" на рис. 44) является важнейшей программой в цепочке средств для приведения исходного кода компонента-правил для инструментального средства *CAME&L* к виду, пригодному для выполнения *FPGA*-процессором. Он состоит из двух частей. Первая, созданная с помощью инструментального средства *LEX* [79], переводит исходный код на языке программирования *C++*, порожденный препроцессором ("1" на рис. 44), в набор лексем языка *C*. Вторая, которая была разработана при помощи средства *YACC* [79], транслирует этот набор на язык *VHDL* [80] ("*VHSIC Hardware Description Language*", где "*VHSIC*" – "*Very-High-Speed Integrated Circuit*"), являющийся одним из самых распространенных языков высокого уровня для программирования, в частности, *FPGA*-систем. Необходимо отметить, что здесь имеет место не тривиальный перевод, а трансляция с элементами семантического анализа. Так, например, обращение к функции `rand()` [62] преобразуется в существенный фрагмент кода на *VHDL*, реализующий генерацию псевдослучайных чисел. Одной из самых существенных задач транслятора является приведение разрядности переменной для хранения состояния клетки, заявленной в исходном коде компонента правил, в соответствие с характеристиками разрядности клетки используемого аппаратного обеспечения.

Далее, полученный на выходе транслятора ("2" на рис. 44) шаблон будущего кода (набор инструкций на языке *VHDL*), обрабатывается синтаксическим анализатором шаблона ("3" на рис. 44). Имея в своем распоряжении подготовленный заранее типичный шаблон обмена

информацией и взаимодействия с памятью, а также, обладая значениями фундаментальных параметров эксперимента, как, например, размерность решетки, синтаксический анализатор шаблона собирает программу на языке *VHDL* окончательно, сопоставляя, в частности, номера портов ввода-вывода всем клеткам решетки *FPGA*-системы для дальнейшего взаимодействия со средой.

Итоговая программа поступает на вход компилятору с языка *VHDL* ("4" на рис. 44), а результат его работы загружается в *FPGA*-систему.

FPGA-система подключается к персональному компьютеру через *UART* (*Universal Asynchronous Receiver/Transmitter*), 16-байтовый *FIFO*-буфер и контроллер *PicoBlaze* [81], от компании *Xilinx*, того же производителя, что и вся *FPGA*-система.

В результате при загрузке в среду инструментального средства *CAME&L* компонента, собранного компилятором ("5" на рис. 44), и последующем запуске эксперимента, будут выполняться следующие действия:

1. Среда отправит на вход устройства *UART* информацию о состояниях клеток решетки на момент начала вычислений.
2. Устройство *UART*, в свою очередь, через входной порт *FPGA*-системы, направляет данные в тот узел (клетку), которому они предназначались.
3. Данные поступают до тех пор, пока все клетки не будут проинициализированы.
4. Среда передаёт устройству занение числа итераций, которое необходимо выполнить.
5. Устройство *UART* транслирует эту директиву головному *RISC*-процессору *FPGA*-системы.
6. Система выполняет необходимое число итераций.

7. Головной процессор *FPGA*-системы, через *UART*, информирует среду о том, что вычисления закончены.
8. Состояния всех клеток решетки, через *UART*, сгружаются в среду.
9. Данный набор действий может быть повторен.

Таким образом, разработан подход и набор инструментов для автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов на *FPGA*-системах, что расширяет область использования инструментального средства *CAME&L*.

4.6. Практическое использование результатов работы

Результаты, полученные в диссертации, используются на практике следующим образом:

1. В учебном процессе на кафедре "Компьютерные технологии" СПбГУ ИТМО при чтении лекций по курсу "Теория автоматов в программировании", а также при выполнении курсовых работ. Акт реализации прилагается.
2. В учебном процессе на кафедре "Вычислительная физика" Санкт-Петербургского государственного университета (СПбГУ) при чтении лекций по курсу "Современные технологии программирования для научных работников". Акт реализации прилагается.
3. В учебном процессе в Университете Амстердама (Нидерланды) при чтении лекций по курсу "Моделирование поведения сложных систем". Акт реализации прилагается.
4. В учебном процессе в Политехническом университете Милана (Италия) при чтении лекций по курсу "Математическое моделирование".

5. В Университете Амстердама при создании программного обеспечения для организации вычислительных экспериментов на программируемых вентильных матрицах. Акт реализации прилагается.
6. В Политехническом университете Бухареста (Румыния) при разработке инструментального средства для обработки изображений были использованы компоненты предложенного в диссертации программного обеспечения. Акт реализации прилагается.

Выводы по главе 4

1. Описан процесс проектирования вычислительных экспериментов на основе клеточных автоматов с помощью разработанного инструментального средства *CAME&L*.
2. Приведены примеры проектирования разнообразных вычислительных экспериментов на различных вычислительных платформах и показана их эффективность.
3. Впервые проведена классификация структур, порождаемых одномерными клеточными автоматами из точечного зародыша. Показано, что существуют автоматы с клетками без памяти и с памятью, демонстрирующие одинаковое поведение. Исследования этого класса структур является важным, так как это – простейшие структуры, многие их которых демонстрируют сложное поведение (в том числе, самовоспроизведение).
4. Разработан подход и набор инструментов для автоматизации проектирования программного обеспечения вычислительных экспериментов с помощью клеточных автоматов на *FPGA*-системах, что расширяет область использования инструментального средства *CAME&L*.

5. Показано, что инструментальное средство *CAME&L* успешно внедрено в учебный и научно-исследовательский процессы ряда университетов.

Заключение

В настоящей работе разработаны метод введения обобщенных координат для клеточных автоматов, обеспечивающий универсальность организации данных для разнообразных решеток произвольной размерности, а также инструментальное средство *CAME&L* для автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов, которое обеспечивает единообразие подхода к решению задачи вне зависимости от применяемой вычислительной платформы. При этом отметим, что разработанный метод вносит вклад в теорию клеточных автоматов, что подтверждается публикацией в материалах одной из крупнейших конференций по вычислительным наукам "*Cellular Automata for Research and Industry*" (Амстердам, 2004).

Проект *CAME&L* – первая попытка создания универсального расширяемого инструмента, пригодного для проведения широкого круга вычислительных экспериментов с помощью клеточных автоматов. Библиотека *CADLib*, входящая в пакет, обладает широким спектром функциональных возможностей. Например, она обеспечивает детальное описание клеточных автоматов, сбор статистики, анализ и вывод полученных данных в различных формах и т.д.

Среда, являющаяся частью инструментального средства *CAME&L*, предоставляет пользователю интуитивно ясный интерфейс с широкими возможностями для выполнения вычислительных экспериментов с помощью клеточных автоматов. В частности, она позволяет единообразно выполнять эксперименты, как на однопроцессорных, так и на многопроцессорных и кластерных вычислительных архитектурах.

Одним из ключевых преимуществ, которыми обладает предложенное средство, является поддержка проведения вычислительных экспериментов с помощью разнообразных метрик и, в частности, обобщенных координат.

В работе приводятся примеры использования разработанного инструментального средства для ряда предметных областей и вычислительных платформ. Так, в частности, описан набор средств для автоматизации проектирования программного обеспечения *FPGA*-систем с помощью инструментального средства *CAME&L*.

Помимо применения для решения научных задач, описываемая среда, имеет и образовательное значение. Она является удобным средством визуализации параллельных алгоритмов, так как для параллельных вычислений не существует более наглядной и универсальной модели, чем клеточные автоматы.

Инструментальное средство *CAME&L* позволяет не только проводить вычислительные эксперименты, но и документировать их. Так, например, большинство иллюстраций для настоящей работы, которые изображают состояния решеток клеточных автоматов, были созданы с его помощью.

Таким образом, результаты, полученные в работе, вносят вклад в теорию клеточных автоматов, а также обеспечивают поддержку вычислительных экспериментов на основе клеточных автоматов для достижения научных и образовательных целей. Полученные результаты важны при разработке и изучении сложных алгоритмов, используемых при создании систем автоматизации проектирования приборов.

В диссертации получены следующие результаты:

1. Введено понятие "обобщенных координат" для решеток клеточных автоматов. Разработан метод введения таких координат.

2. На основе метода введения обобщенных координат разработано универсальное и расширяемое инструментальное средство автоматизации проектирования программного обеспечения вычислительных экспериментов с использованием клеточных автоматов *CAME&L*.
3. Метод и инструментальное средство *CAME&L* апробированы при автоматизации проектирования программного обеспечения ряда вычислительных экспериментов с использованием клеточных автоматов на однопроцессорной, многопроцессорной, кластерной и *FPGA* вычислительных системах. Это было невозможно выполнить ранее с помощью одного и того же инструментального средства.
4. Проведена классификация всех структур, порождаемых одномерными клеточными автоматами из точечного зародыша. Построено множество разнообразных классов эквивалентности. Показано, что автоматы с клетками с памятью и автоматы с клетками без памяти могут демонстрировать одинаковое поведение. Таким образом, выполнен анализ простейших клеточных автоматов, порождающих сложное поведение (в том числе, самовоспроизведение).
5. Результаты были внедрены в учебном процессе и при проведении научных исследований в ряде университетов.

На основе выполненной работы создан информационный ресурс "*CAMEL Laboratory*" – http://camellab.spb.ru, который содержит инструментальное средство *CAME&L*, доступное для свободного использования вместе с множеством разработанных компонентов и примеров его применения.

Список терминов

Анализатор (*analyzer*) клеточного автомата – в терминах среды *CAME&L*, один из компонентов вычислительного эксперимента, позволяющий наблюдать за характеристиками моделируемой системы.

Асинхронный клеточный автомат (*asynchronous cellular automaton*) – клеточный автомат, состояния клеток которого обновляются не одновременно, после вычисления новых состояний для каждой из них, а в порядке обхода.

Клетка (*cell*) клеточного автомата – элементарная единица моделирования системы, характеризуемая своим состоянием и выполняющая переходы из одного состояния в другое на основании функции переходов.

Клеточный автомат (*cellular automaton*) – модель параллельных вычислений, представляющая собой дискретную динамическую систему, поведение которой может быть полностью описано в терминах локальных зависимостей.

Клеточный автомат с клетками без памяти (*cellular automaton with memoryless cells*) – клеточный автомат, правила которого написаны таким образом, что состояние данной клетки на следующем шаге не зависит от ее состояния на текущем шаге.

Клеточный автомат с клетками с памятью (*cellular automaton with cells with memory*) – клеточный автомат, правила которого написаны таким образом, что состояние данной клетки на следующем шаге зависит от ее состояния на текущем шаге.

Компонент (*component*) – элементарная составляющая, из которой строится вычислительный эксперимент в инструментальном средстве *CAME&L*. Можно выделить пять типов компонентов: решетки, метрики,

хранилища данных, правила и анализаторы. Первые четыре формируют клеточный автомат и могут участвовать в эксперименте в единственном экземпляре. Анализаторы могут быть вовлечены в вычисления в произвольном количестве. С точки зрения разработчика, компонент представляет собой библиотеку динамической линковки (*DLL*), содержащую класс-наследник базового класса библиотеки *CADLib*, а также три функции: для создания объекта содержащегося класса, его уничтожения и представления библиотеки системе.

Метрика (*metrics*) клеточного автомата – метрика, введенная на решетке клеточного автомата. В терминах среды *CAME&L*, метрика – один из компонентов клеточного автомата, обеспечивающий разрешение отношений соседства между клетками, а также ряд других метрических функций.

Многоагентная система (*multiagent system*) – система, состоящая из нескольких компонентов (агентов), способных к взаимодействию. Взаимодействие может осуществляться в форме, например, передачи сообщений или изменения их общей окружающей среды.

Обобщенная координата (*generalized coordinate*) клетки клеточного автомата – неотрицательное целое число, позволяющее однозначно идентифицировать клетку.

Окрестность клетки (*neighborhood*) – множество клеток, состояние которых передается в функцию переходов в качестве параметра, для определения нового состояния данной клетки.

Подзадача (*subtask*) – часть задачи, решение которой приближает к решению всей исходной задачи. В контексте клеточных автоматов, как правило, подзадача алгоритмически эквивалентна исходной, однако решается над неким подмножеством входных данных, над некой частью решетки, называемой "подрешеткой".

Подрешетка (*subgrid*) – часть решетки клеточного автомата, над которой выполняется подзадача.

Правила (*rules*) клеточного автомата – функция, определяющая закономерности перехода клетки из одного состояния в другое в зависимости от состояний соседних клеток, а также, возможно, ее самой. В терминах среды *CAME&L*, правила – один из компонентов клеточного автомата, причем они трактуются несколько шире, чем "функция переходов", так как помимо перехода, могут описывать схему параллелизации, методику оптимизации и другие вопросы выполнения вычислений.

Решетка (*grid*) клеточного автомата – не более чем счетное множество клеток автомата, геометрически организованное, как *n*-мерная регулярная структура, никакие две области которой в классической модели не могут быть отличены по ландшафту. В терминах среды *CAME&L*, решетка – один из компонентов клеточного автомата. В отличие от трактовки в теории клеточных автоматов, в обсуждаемой среде компонент не подразумевает реального хранения информации о состояниях клеток (этим занимается "хранилище данных"), но лишь визуализацию и навигацию.

Сеть Петри (*Petri net*) – модель распределенных вычислений, представляющая собой двудольный ориентированный граф (множество вершин которого разбивается на два подмножества и не существует дуги, соединяющей две вершины из одного подмножества).

Соседи, соседние клетки (*neighbors*) – элементы окрестности (см. "Окрестность клетки").

Стохастический клеточный автомат (*stochastic cellular automaton*) – клеточный автомат, правила которого имеют вероятностный характер – зависят он неких псевдослучайных параметров.

Функция переходов (*transitions function*) клеточного автомата – см. "Правила клеточного автомата".

Хранилище данных (*datum*) клеточного автомата – в терминах среды *CAME&L*, один из компонентов клеточного автомата, изоморфный решетке, обеспечивающий хранение информации о состояниях клеток.

Предметный указатель

- З -

- К -

Литература

1. *Тоффоли Т., Марголус Н.* Машины клеточных автоматов. М.: Мир. 1991.
2. *Wolfram S.* A New Kind of Science. Wolfram Media Inc. 2002.
3. *фон Нейман Дж.* Теория самовоспроизводящихся автоматов. М.: Мир. 1971.
4. *Cellular Automata*. Ed. P.M.A. Sloot, B. Chopard, A. Hoekstra / Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. Springer–Verlag. 2004.
5. *Sarkar P.* A Brief History of Cellular Automata // ACM Computing Surveys. Vol. 32. 2000. № 1.

Error! Bookmark not defined.. *Сайт* "CAMEL Laboratory" – http://camellab.spb.ru.

6. *Ulam S.* Random Processes and Transformations / Proceedings of the International Congress of Mathematicians. Rhode Island: American Mathematical Society. 1952. Vol. 2.
7. *Zuse K.* Calculating Space. Massachusetts Institute of Technology Technical Translation AZT-70-164-GEMIT (Project MAC). Cambridge. 1970.
8. *Hedlung G. A.* Endomorphism and Automorphism of the Shift Dynamic System // Mathematical Systems Theory. 1969. № 3.
9. *Holland J.* Universal Spaces: A Basis for Studies in Adaptation // Automata Theory. Academic Press. 1966.
10. *Richardson D.* Tessellation with Local Transformations // Journal of Computer and System Sciences. 1972. № 6.
11. *Варшавский В.И., Мараховский В.Б., Песчанский В.А., Розенблюм Л.Я.* Однородные структуры. М.: Энергия. 1973.

12. *Qi A., Zheng X., Du C., An B.* A cellular automation model of cancerous growth // Journal of Theoretical Biology. 1993. № 161.
13. *Mansury Y., Kimuraz M., Lobozy J., Deisboeck T. S.* Emerging Patterns in Tumour Systems: Simulating the Dynamics of Multicellular Clusters with an Agent-based Spatial Agglomeration Model // Journal of Theoretical Biology. 2002. № 219.
14. *Dormann S., Deutsch A.* Modeling of self-organized avascular tumour growth with a hybrid cellular automaton // In Silico Biology 2. 2002.
15. *Moore J., Hahn L.* A cellular automata approach to detecting interactions among single-nucleotide polymorphisms in complex multifactorial diseases / Pacific Symposium on Biocomputing. 2002.
16. *Ward D.P., Murray A.T., Phinn S.R.* An optimized cellular automata approach for sustainable urban development in rapidly urbanizing regions / Geocomutation - 1999. 1999.
17. *Chopard B., Luthi P., Masselot A.* Cellular automata and lattice boltzmann techniques: An approach to model and simulate complex systems // Advances in Physics, submitted, 1998. № 137.
18. *Sullivan A., Knight I.* A hybrid cellular automata/semi-physical model of fire growth / Proceedings of the 7th Asia-Pacific Conference on Complex Systems. 2004.
19. *Chavez F., Vicente L.* Cellular automata approach to dissociative adsorption of diatomic molecules on a substrate with reconstruction // Molecular Physics. Vol. 96. 1999. № 6.
20. *Swiecicka A., Seredynski F.* Cellular automata approach to scheduling problem // Parallel Computing in Electrical Engineering. 2000. № 1.

21. *Janowicz M., Orlowski A.* Cellular automaton approach to light propagation in dispersive periodic and disordered media // Transparent Optical Networks. Vol. 2. 2004.
22. *Lee et. al.* Adaptive Stochastic Cellular Automata: Theory // Physica D: Nonlinear Phenomena. Cellular Automata: Theory and Experiment, 1990.
23. *Janowicz M., Ashbourn J., Orłowski A., et al.* Cellular automaton approach to electromagnetic wave propagation in dispersive media // Royal Society of London. Vol. 462. 2006 № 2074.
24. *Biondini F., Bontempi F., Frangopol D., Malerba P.* Cellular Automata Approach to Durability Analysis of Concrete Structures in Aggressive Environments // J. Struct. Engrg. Vol. 130. 2004. № 11.
25. *Keedwell E., Soon-Thiam K.* Novel Cellular Automata Approach to Optimal Water Distribution Network Design // J. Comp. in Civ. Engrg. Vol. 20. 2006. № 1.
26. *Zoran A.* Computation in Inhomogenous Celluar Automata / Complex Systems: From Biology to Computation. 2003
27. *Gardner M.* The Fantastic Combinations of John Conway's New Solitaire Game "Life" // Scientific American. 1970. № 223.
28. *Гарднер М.* Математические досуги. М.: Мир. 1972.
29. *Гарднер М.* Крестики-нолики. М.: Мир. 1988.
30. *Berlekamp E., Conway J. Guy R.* What Is Life? // Winning Ways. Vol. 2. Academic Press. 1982.
31. *Корн Г., Корн Т.* Справочник по математике (для научных работников и инженеров). М.: Наука. 1978.
32. *Брауэр В.* Введение в теорию конечных автоматов. М.: Радио и связь. 1987.

33. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998.
34. *Margolus N.* CAM-8: computer architecture based on cellular automata / Pattern Formation and Lattice-Gas Automata. Addison-Wesley. 1996.
35. *Броуди Л.* Начальный курс программирования на языке FORTH. М.: Финансы и статистика. 1990.
36. *Toffoli T.* Cellular Automata Mechanics // Tech. Rep. 208. Comp. Comm. Sci. Dept., The University of Michigan. 1977.
37. *Сайт* "Sigis.net" – http://lamp.sigis.net/article/articleview/2/1/5/.
38. *Сайт* "UCLA Gateway" – ftp://ftp.lifesci.ucla.edu/pub/alife/public/cam.zip.
39. *Сайт* Доменико Талия – http://isi-cnr.deis.unical.it:1080/~talia/.
40. *MacDonald N., Minty E., Harding T., Brown S.* Writing Message Passing Parallel Programs with MPI – Edinburgh Parallel Computing Center. The University of Edinburgh. 1996.
41. *Spezzano G., Talia D.* Designing Parallel Models of Soil Contamination by the CARPET Language. 1998.
42. *Сайт* "Department of Computer Science San Jose State University" – http://www.mathcs.sjsu.edu/capow/capow4a.zip.
43. *Сайт* "Friedrich-Schiller-Universität" – http://www.uni-jena.de/~msk/bin/casim.tar.gz.
44. *Сайт* "Gesellschaft für Mathematik und Datenverarbeitung" – ftp://ftp.gmd.de:/GMD/cat/.
45. *Сайт* "Technische Universität Darmstadt" – ftp://ftp.informatik.th-darmstadt.de/pub/MP/CDL/.
46. *Сайт* "Essex.ac.uk" – ftp://ftp.essex.ac.uk/pub/robots/ArtificialLife/public/cdm/.

47. *Сайт* "Index Librorum Liberorum by John Walker" – http://www.fourmilab.ch/cellab/cellab.zip.
48. *Сайт* "Santa Fe Institute. Artificial Life" – ftp://alife.santafe.edu/pub/SOFTWARE/Cellsim/.
49. *Сайт* "Radford University" – ftp://rucs2.sunlab.cs.runet.edu/pub/ca/cellular.tar.gz.
50. *Сайт* "Santa Fe Institute" – ftp://ftp.santafe.edu/pub/wuensch/.
51. *Сайт* Томаса Уорша – http://liinwww.ira.uka.de/~worsch/.
52. *Сайт* "Departamento De Computacion. Cinvestav" – http://delta.cs.cinvestav.mx/~mcintosh/newweb/software.html.
53. *Сайт* "Mirek's Free Software Site" – http://www.mirekw.com.
54. *Сайт* "Mirek's Personal Homepage" – http://www.mirwoj.opus.chelm.pl/.
55. *Сайт* "Pea Soup" – http://psoup.math.wisc.edu/mcell/.
56. *Сайт* Мартина Кутриба – http://www.informatik.uni-giessen.de/staff/kutrib.html.
57. *Сайт* "BU Programmable Matter" – http://pm.bu.edu/step.
58. *Сайт* "University of Wisconsin-Madison. Mathematics Department" – ftp://cam8.math.wisc.edu/pub/winca_10.exe.
59. *Sagan H.* Space Filling Curves. Springer. 2006.
60. *Bungartz H-J., Mehl M., Weinzierl T.* A Parallel Adaptive Cartesian PDE Solver Using Space-Filling Curves / Proceedings of "Euro-Par 2006". 2006.
61. *Günther F., Krahnke A., Langlotz M., Mehl M., Pögl M., Zenger C.* On the Parallelization of a Cache-Optimal Iterative Solver for PDEs Based on Hierarchical Data Structures and Space-Filling Curves // Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer. 2004.
62. *Страуструп Б.* Язык программирования C++. СПб: Невский диалект. 1999.

63. *Weimar J.R.* Simulation with Cellular Automata. Braunschweig. 1996.
64. *Спэйнауэр С., Экштейн Р.* Справочник вебмастера. СПб.: Символ-Плюс. 2001.
65. *Мешков А., Тихомиров Ю.* Visual C++ и MFC. СПб.: БХВ–Санкт-Петербург. 2000.
66. *Сайт* "GNU + Cygnus + Windows = Cygwin" – http://www.cygwin.com.
67. *Гордеев А., Молчанов А.* Системное программное обеспечение. СПб.: Питер. 2001.
68. *Уэлстид С.* Фракталы и вейвлеты для сжатия изображений в действии. М.: Триумф. 2003.
69. *Газале М.* Гномон. От фараонов до фракталов. Москва-Ижевск: Институт компьютерных исследований. 2002.
70. *Вирт Н.* Алгоритмы и структуры данных. СПб.: Невский диалект. 2001.
71. *Сайт* кафедры "Технологии программирования" Санкт-Петербургского государственного университета информационных технологий, механики и оптики – http://is.ifmo.ru.
72. *Wolf W.* FPGA-Based System Design. Prentice Hall Ptr. 2004.
73. *Lee S.* Advanced Digital Logic Design Using VHDL, State Machines, and Synthesis for FPGA's. Thomson-Engineering. 2005.
74. *Coffman K.* Real World FPGA Design with Verilog. Prentice Hall Ptr. 1999.
75. *Woudenberg M.* Using FPGAs to Speed Up Cellular Automata Computations / Master's thesis for University of Amsterdam. 2006.
76. *Venkatachalam D.* Reconfigurable Processors: Changing the Systems Design Paradigm. http://www.techonline.com/community/ed_resource/featurearticle/38022.

77. *Shackleford B., Tanaka M., Carter R.J., Snider G.* FPGA Implementation of Neighborhood-of-Four Cellular Automata Random Number Generators / Proceedings of FPGA'2002. 2002.
78. *Halbach M., Hoffmann R.* Implementing Cellular Automata in FPGA Logic / Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04). 2004.
79. *Керниган Б., Пайк Р.* UNIX. Программное окружение. СПб.: Символ-Плюс. 2003.
80. *Pedroni V.* Circuit Design with VHDL. The MIT Press. 2004.
81. *Xilinx* APPnote 213. http://direct.xilinx.com/bvdocs/appnotes/xapp213.pdf.

Публикации автора по теме диссертации

Error! Bookmark not defined.. *Naumov L.* CAME&L – Cellular Automata Modeling Environment & Library / Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. Springer–Verlag. 2004. с. 735–744.

Error! Bookmark not defined.. *Наумов Л.* CAME&L – среда моделирования и библиотека разработчика клеточных автоматов / Труды XI Всероссийской научно-методической конференции Телематика'2004. Том 1. СПб.: СПбГУ ИТМО. 2004. с. 184–186.

Error! Bookmark not defined.. *Наумов Л.* CAME&L – средство для осуществления параллельных и распределенных вычисления на основе клеточных автоматов / Технологии распределенных вычислений. 2005. с. 284–286.

83. *Наумов Л., Шалыто А.* Клеточные автоматы. Реализация и эксперименты // Открытые системы. Мир ПК. 2003. №8. с. 71–78.

84. *Наумов Л., Шалыто А.* "Цветные" клеточные автоматы // Открытые системы. Мир ПК. 2004. №5. с. 64–71.
85. *Наумов Л.* Как увеличить скорость "Жизни", или Эффективная организация данных для повышения скорости поиска клеток и разрешения отношений соседства при реализации клеточного автомата Джона Хортона Конвея "Жизнь". Части I и II // Информатика. 2001. № 33 (322). с. 25–27; № 34 (323). с. 20–24.
86. *Наумов Л.* Как увеличить скорость "Жизни", или Эффективная организация данных для повышения скорости поиска клеток и разрешения отношений соседства при реализации клеточного автомата Джона Хортона Конвея "Жизнь" // Компьютеры + Программы. 2001. №10. с. 68–73.
87. *Наумов Л.* Обзор программного обеспечения для решения задач с использованием клеточных автоматов / Телекоммуникации и информатизация образования. 2006. №2. с. 114–125.
88. *Наумов Л.* CTP (Commands Transfer Protocol) – Сетевой протокол для высокопроизводительных вычислений / Труды XI Всероссийской научно-методической конференции Телематика’2004. Том 1. СПб.: СПбГУ ИТМО. 2004. с. 188–189.
89. *Наумов Л.* CTP (Commands Transfer Protocol) v. 1.2 – Новая версия сетевого протокола для высокопроизводительных вычислений / Труды XII Всероссийской научно-методической конференции Телематика’2005. Том 1. СПб.: СПбГУ ИТМО. 2005. с. 92–93.
90. *Наумов Л.* Сравнение специализированных сетевых протоколов CTP (Commands Transfer Protocol) и IL (Internet Link) / Труды XIII Всероссийской научно-методической конференции Телематика’2006. Том 1. СПб.: СПбГУ ИТМО. 2006. с. 266–267.

91. *Naumov L.* Generalized Coordinates for Cellular Automata Grids / Computational Science – ICCS 2003. Part 2. Springer-Verlag. 2003. с. 869–878.
92. *Наумов Л.* Решение задач с помощью клеточных автоматов посредством программного обеспечения CAMEL (Части I и II) // Информационно-управляющие системы. 2005. № 5. с. 22–30; № 6. с. 30–38.
93. *Наумов Л., Шалыто А.* Классификация структур, порождаемых одномерными двоичными клеточными автоматами из точечного зародыша // Известия РАН. Теория и системы управления. №5. 2005. с. 137–145. Журнал из списка ВАК.
Naumov L., Shalyto A. Classification of Structures Generated by One-Dimensional Binary Cellular Automata from a Point Embryo // Journal of Computer and Systems Sciences International. Vol. 44. 2005. №5. с. 800–807.

Приложения

Приложение 1. Исходный код компонента правил для генерации структур, порождаемых одномерными клеточными автоматами

Как обычно, компонент правил реализуется набором файлов, большинство из которых типичны и повторяются для каждого компонента, с точностью до названий и описаний, содержащихся в них. Приведем два ключевых файла библиотеки компонента, которые описывают всю функциональность, необходимую для генерации структур, порождаемых одномерными клеточными автоматами. Описываемый компонент доступен с сайта [**Error! Bookmark not defined.**].

Файл *Research1D.h*

```
class CAResearch1D:public CARules
{
public:
    // Конструктор класса
    CAResearch1D();

    // Деструктор класса
    virtual ~CAResearch1D();

    // Перегрузка методов, унаследованных от класса
    // CAComponent
    COMPONENT_NAME(Research 1D)
```

```
    COMPONENT_INFO(Research of structures generated by
        1D binary cellular automaton)
    COMPONENT_ICON(IDI_ICON)
    COMPONENT_REQUIRES(Data.bool.*&
        Metrics.2D.cartesian.*)

    // Перегрузка методов, унаследованных от класса
    // CARules
    virtual bool Initialize();
    virtual bool SubCompute(Zone& z);

// Параметры:
public:
    // Номер функции переходов, в обсуждаемых выше
    // терминах
    ParamInt p_iRule;

// Функции, обеспечивающие работу с параметрами:
public:
    // Количество параметров
    PARAMETERS_COUNT(1)

    // Карта параметров
    BEGIN_PARAMETERS
        PARAMETER(0,&p_iRule)
    END_PARAMETERS
    PARAMETER_CHANGED
```

```
// Внутренние переменные
protected:
    // Массив булевых переменных для разбиения номера
    // функции переходов на биты
    bool bits[8];
};
```

Файл *Research1D.cpp*

```
// Подключаем заголовочные файлы библиотеки CADLib
#include <AfxDpnds.h>
#include <CADLib.h>

// Подключаем прочие заголовочные файлы для данного
// компонента
#include "Resource.h"
#include "Research1D.h"
#include "CompsSrc\Data\CrtsBool2D\CrtsBool2D.h"

// Добавить в библиотеку компонента
// функцию аутентификации
COMPATIBLE_RULES(1.1)
// Добавить в библиотеку компонента
// функции создания и уничтожения компонента
RULES_COMPONENT(CAResearch1D)

CAResearch1D::CAResearch1D():
```

```
    p_iRule("Rule","Number of the rule to be
        used",0,0,255,this)
{
    // Установить текстовый комментарий
    sComment="Research 1D";
}

CAResearch1D::~CAResearch1D()
{
}

bool CAResearch1D::Initialize()
{
    // Сбросить счетчик шагов. Это действие очень важно
    // для рассматриваемой задачи, так как мы будем
    // моделировать одномерный клеточный автомат с
    // помощью двумерного, причем номер шага, взятый со
    // знаком минус, будет выполнять у нас роль
    // координаты абсцисс
    Step=0;

    // Это стандартное макроопределение обеспечивает
    // то, что теперь указатель "datum" указывает на
    // текущий компонент хранилище данных, причем он
    // приведен к типу "CACrtsBool2DDatum *"
    DATUM(CACrtsBool2DDatum);
    // Поместить "точечный зародыш" в клетку (0;0;0)
```

```
    datum->Set(
        GetUnion()->Metrics->ToCell(0,0,0),true);

    // Разделить номер функции переходов на биты
    bits[0]=(p_iRule.Get()&1)?true:false;
    bits[1]=(p_iRule.Get()&2)?true:false;
    bits[2]=(p_iRule.Get()&4)?true:false;
    bits[3]=(p_iRule.Get()&8)?true:false;
    bits[4]=(p_iRule.Get()&16)?true:false;
    bits[5]=(p_iRule.Get()&32)?true:false;
    bits[6]=(p_iRule.Get()&64)?true:false;
    bits[7]=(p_iRule.Get()&128)?true:false;

    return true;
}

bool CAResearch1D::SubCompute(Zone& z)
{
    // Это стандартное макроопределение обеспечивает
    // то, что теперь указатель "datum" указывает на
    // текущий компонент хранилище данных, причем он
    // приведен к типу "CACrtsBool2DDatum *"
    DATUM(CACrtsBool2DDatum);

    // Идентификатор рассматриваемой клетки на
    // предыдущем шаге
    CACell c;
```

```
// Идентификатор левого соседа рассматриваемой
// клетки на предыдущем шаге
CACell cleft;
// Идентификатор правого соседа рассматриваемой
// клетки на предыдущем шаге
CACell cright;
// Композитное состояние рассматриваемой клетки и
// ее соседей, вычисляемое, как сумма состояния
// левого соседа, умноженного на 4, самой клетки,
// умноженного на 2 и правого соседа
int composite;

for(CACell i=z.a1; i<=z.b1; i++)
    {
        // Получить идентификаторы клеток, которые
        // нас интересуют
        c=GetUnion()->Metrics->ToCell(
            i,-(CACell)Step,0);
        cleft=GetUnion()->Metrics->ToCell(
            i-1,-(CACell)Step,0);
        cright=GetUnion()->Metrics->ToCell(
            i+1,-(CACell)Step,0);

        // Вычислить композитное состояние
        composite=(datum->Get(cleft)?4:0)+
            (datum->Get(c)?2:0)+
            (datum->Get(cright)?1:0);
```

```
            // Присвоение нового значение состоянию
            // клетки
            datum->Set(GetUnion()->Metrics->ToCell(
                i,-(CACell)Step-1,0),bits[composite]);
        }

    return true;
}

void CAResearch1D::ParameterChanged(Parameter* pPar)
{
    if (pPar==&p_iRule) {
        // Если выбранный номер функции переходов был
        // изменен, то эксперимент следует
        // переинициализировать
        Initialize();
    }
}
```

Приложение 2. Структуры, порождаемые клеточными автоматами из точечного зародыша и некоторые варианты классификации

В следующей таблице перечисляются все структуры, порождаемые одномерными клеточными автоматами, рассмотренными в настоящей главе.

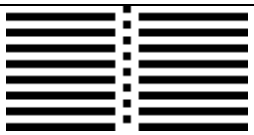
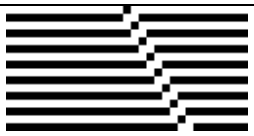
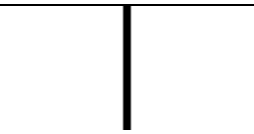
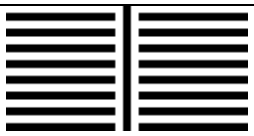
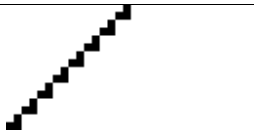
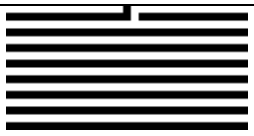
В первом столбце указан номер функции автомата. Во втором столбце приводится изображение порождаемой структуры после 16 шагов. Столбцы с третьего по девятый содержат *E*-, *EM*-, *EI*-, *E*(*I*+*M*)-, *EIM*-, *EIMO*- и *EIMOL*6-классификации. При этом в столбцах таблицы могут встретиться ячейки двух цветов. Каждая ячейка серого цвета соответствует первой встретившейся структуре, принадлежащей данному классу. Она содержит два числа (второе – в скобках). Первое из них – номер класса, которому принадлежит данная структура (они пронумерованы целыми числами, начиная с нуля), а второе – количество экземпляров в данном классе. Все остальные клетки – белые. Они содержат одно число, представляющее собой номер структуры, являющейся первым экземпляром того же класса, что и данная.

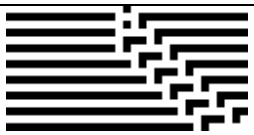
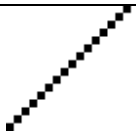
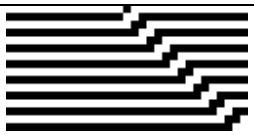
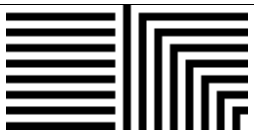
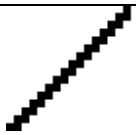
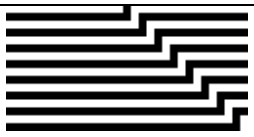
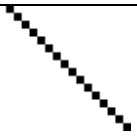
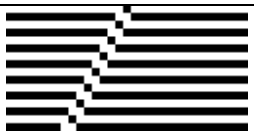
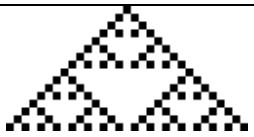
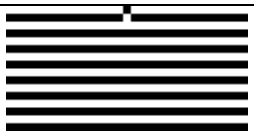
Таким образом, можно утверждать, что серые клетки являются хранилищами описаний классов (номера класса и количества экземпляров в нем), а белые – указателями на эти описания.

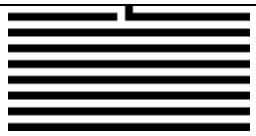
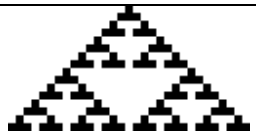
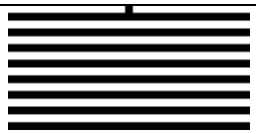
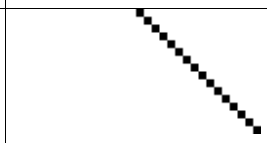
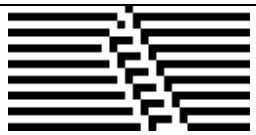
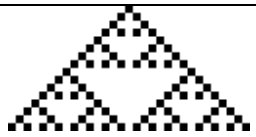
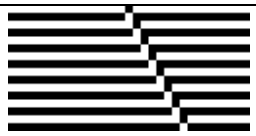
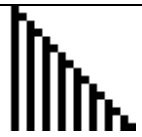
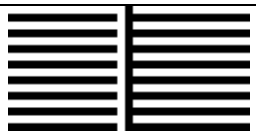
Приведем пример использования таблицы. Из ее рассмотрения следует, что структура номер 30 в *EIMO*-классификации не попадает ни в один из девятнадцати классов, сформированных первыми тридцатью структурами. Она создает новый двадцатый (номер девятнадцать, если считать с нулевого) класс, в который, попадают всего четыре структуры. Об этом информирует серая клетка, содержащая запись "19 (4)". Отсюда следует вывод, что в столбце, соответствующем *EIMO*-классификации, существуют еще три белые

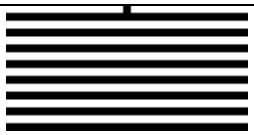
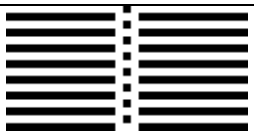
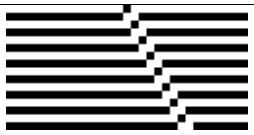
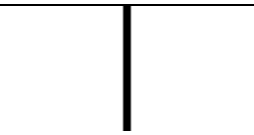
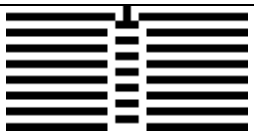
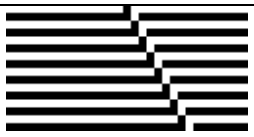
ячейки, которые содержат число тридцать. Действительно, в строках, соответствующих функциям с номерами 86, 135 и 149, указан номер "30".

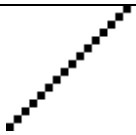
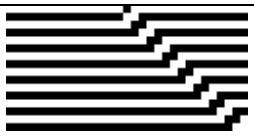
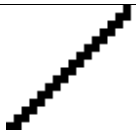
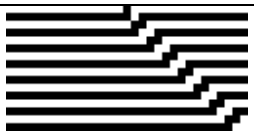
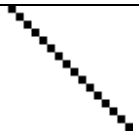
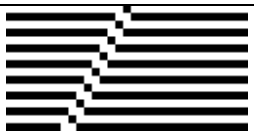
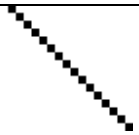
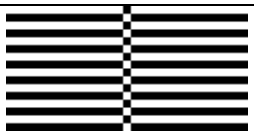
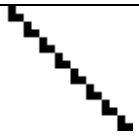
Таблица 13. Число классов при разных вариантах классификации со сдвигами

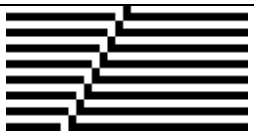
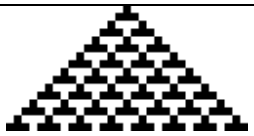
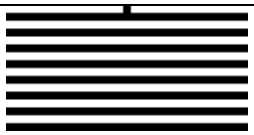
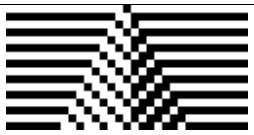
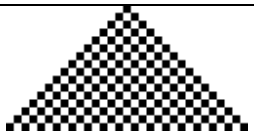
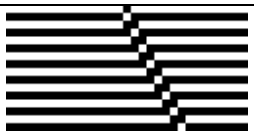
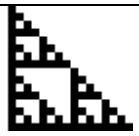
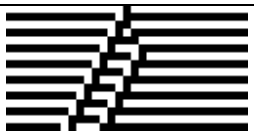
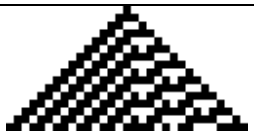
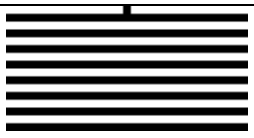
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
0		0 (16)	0 (16)	0 (24)	0 (24)	0 (24)	0 (30)	0 (31)
1		1 (2)	1 (2)	1 (2)	1 (2)	1 (2)	1 (3)	1 (3)
2		2 (16)	2 (32)	2 (17)	2 (17)	2 (34)	2 (40)	2 (40)
3		3 (2)	3 (4)	3 (2)	3 (2)	3 (4)	3 (6)	3 (6)
4		4 (16)	4 (16)	4 (17)	4 (17)	4 (17)	4 (21)	4 (21)
5		5 (1)	5 (1)	5 (1)	5 (1)	5 (1)	5 (1)	5 (1)
6		6 (4)	6 (8)	6 (4)	6 (4)	6 (8)	6 (10)	6 (10)
7		7 (1)	7 (2)	7 (1)	7 (1)	7 (2)	7 (11)	7 (11)
8		0	0	0	0	0	0	0

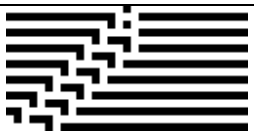
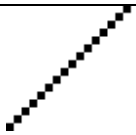
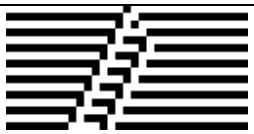
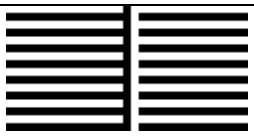
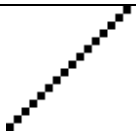
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
9		8 (1)	8 (2)	8 (1)	8 (1)	8 (2)	8 (4)	8 (4)
10		2	2	2	2	2	2	2
11		9 (2)	9 (4)	9 (2)	9 (2)	9 (4)	9 (6)	9 (6)
12		4	4	4	4	4	4	4
13		10 (1)	10 (2)	10 (1)	10 (1)	10 (2)	10 (4)	10 (4)
14		11 (4)	11 (8)	11 (4)	11 (4)	11 (8)	11 (14)	11 (14)
15		12 (1)	12 (2)	12 (1)	12 (1)	12 (2)	12 (2)	12 (2)
16		13 (16)	2	13 (17)	13 (17)	2	2	2
17		14 (2)	3	14 (2)	14 (2)	3	3	3
18		15 (8)	13 (8)	15 (9)	15 (9)	13 (9)	13 (11)	13 (11)
19		16 (1)	14 (1)	16 (1)	16 (1)	14 (1)	7	7

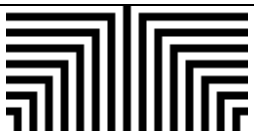
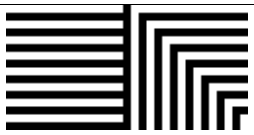
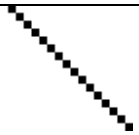
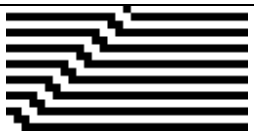
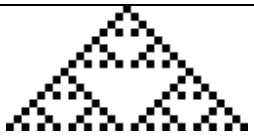
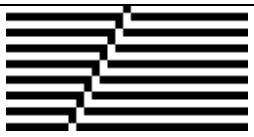
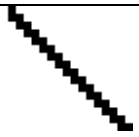
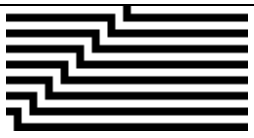
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
20		17 (4)	6	17 (4)	17 (4)	6	6	6
21		18 (1)	7	18 (1)	18 (1)	7	7	7
22		19 (1)	15 (1)	19 (1)	19 (1)	15 (1)	14 (1)	14 (1)
23		20 (8)	16 (8)	20 (8)	20 (8)	16 (8)	7	7
24		16	2	16	16	2	2	2
25		21 (1)	17 (2)	21 (1)	21 (1)	17 (2)	15 (4)	15 (4)
26		18	18	18	18	18	18	18
27		22 (1)	18 (2)	22 (1)	22 (1)	18 (2)	16 (4)	16 (4)
28		23 (2)	19 (4)	23 (2)	23 (2)	19 (4)	17 (6)	17 (6)
29		24 (1)	20 (2)	24 (1)	24 (1)	20 (2)	18 (2)	18 (2)
30		25 (1)	21 (2)	25 (1)	25 (1)	21 (2)	19 (4)	19 (4)

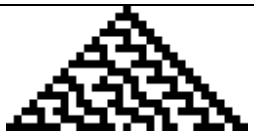
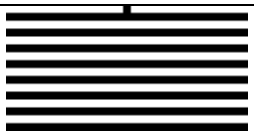
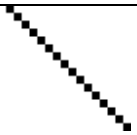
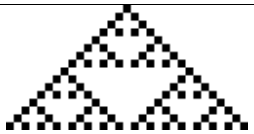
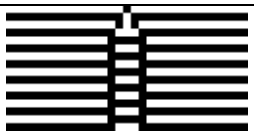
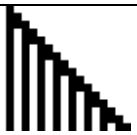
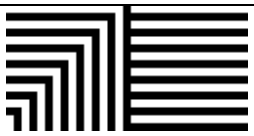
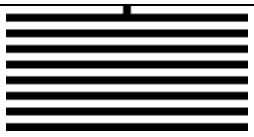
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
31		23	23	23	23	23	7	7
32		0	0	0	0	0	0	0
33		1	1	1	1	1	1	1
34		2	2	2	2	2	2	2
35		3	3	3	3	3	3	3
36		4	4	4	4	4	4	4
37		26 (1)	22 (1)	26 (1)	26 (1)	22 (1)	20 (2)	20 (2)
38		6	6	6	6	6	6	6
39		27 (1)	23 (2)	27 (1)	27 (1)	23 (2)	27	27
40		0	0	0	0	0	0	0
41		28 (1)	24 (2)	28 (1)	28 (1)	24 (2)	21 (2)	21 (2)

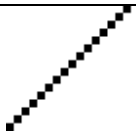
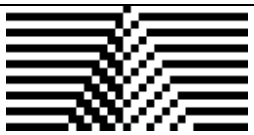
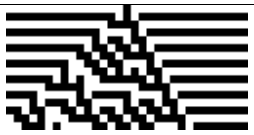
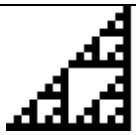
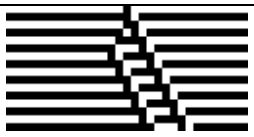
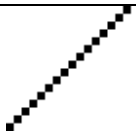
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
42		2	2	2	2	2	2	2
43		11	11	11	11	11	11	11
44		4	4	4	4	4	4	4
45		29 (1)	25 (2)	29 (1)	29 (1)	25 (2)	22 (2)	22 (2)
46		14	14	14	14	14	14	14
47		30 (1)	26 (2)	30 (1)	30 (1)	26 (2)	11	11
48		16	2	16	16	2	2	2
49		17	3	17	17	3	3	3
50		31 (8)	27 (8)	31 (8)	31 (8)	27 (8)	23 (9)	23 (9)
51		32 (1)	28 (1)	32 (1)	32 (1)	28 (1)	24 (1)	24 (1)
52		20	6	20	20	6	6	6

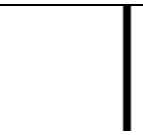
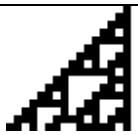
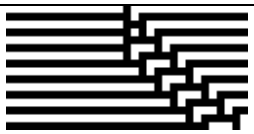
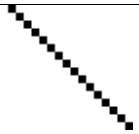
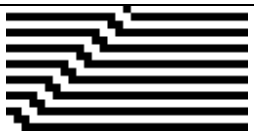
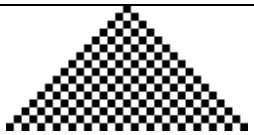
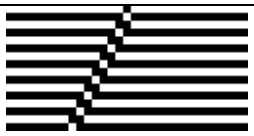
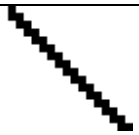
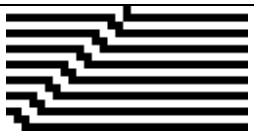
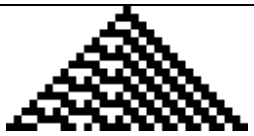
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
53		33 (1)	39	33 (1)	33 (1)	39	27	27
54		34 (1)	29 (1)	34 (1)	34 (1)	29 (1)	25 (2)	25 (2)
55		23	23	23	23	23	7	7
56		16	2	16	16	2	2	2
57		35 (1)	30 (2)	35 (1)	35 (1)	30 (2)	26 (2)	26 (2)
58		50	50	50	50	50	50	50
59		36 (1)	31 (2)	36 (1)	36 (1)	31 (2)	3	3
60		37 (1)	32 (2)	37 (2)	37 (2)	32 (4)	27 (4)	27 (4)
61		38 (1)	33 (2)	38 (1)	38 (1)	33 (2)	25	25
62		39 (1)	34 (2)	39 (1)	39 (1)	34 (2)	28 (2)	28 (2)
63		23	23	23	23	23	7	7

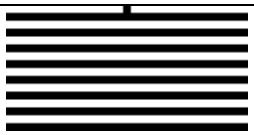
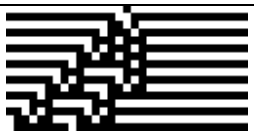
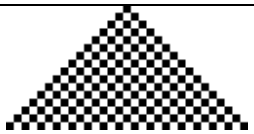
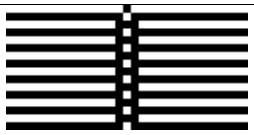
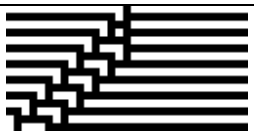
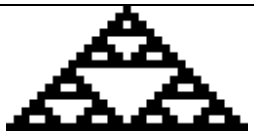
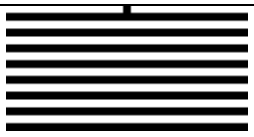
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
64		0	0	0	0	0	0	0
65		40 (1)	9	40 (1)	40 (1)	9	9	9
66		2	2	2	2	2	2	2
67		41 (1)	25	41 (1)	41 (1)	25	25	25
68		4	4	4	4	4	4	4
69		42 (1)	13	42 (1)	42 (1)	13	13	13
70		43 (2)	28	43 (2)	43 (2)	28	28	28
71		44 (1)	29	44 (1)	44 (1)	29	29	29
72		0	0	0	0	0	0	0
73		45 (1)	35 (1)	45 (1)	45 (1)	35 (1)	29 (1)	29 (1)
74		2	2	2	2	2	2	2

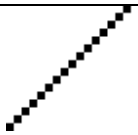
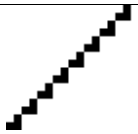
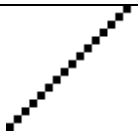
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
75		46 (1)	36 (2)	46 (1)	46 (1)	36 (2)	30 (2)	30 (2)
76		4	4	4	4	4	4	4
77		47 (1)	37 (1)	47 (1)	47 (1)	37 (1)	31 (1)	31 (1)
78		48 (1)	38 (2)	48 (1)	48 (1)	38 (2)	32 (2)	32 (2)
79		49 (1)	39 (2)	49 (1)	49 (1)	39 (2)	13	13
80		16	2	16	16	2	2	2
81		50 (2)	11	50 (2)	50 (2)	11	11	11
82		18	18	18	18	18	18	18
83		51 (1)	27	51 (1)	51 (1)	27	27	27
84		52 (4)	14	52 (4)	52 (4)	14	14	14
85		53 (1)	15	53 (1)	53 (1)	15	15	15

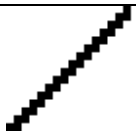
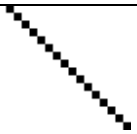
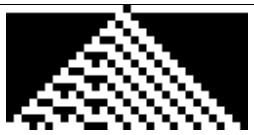
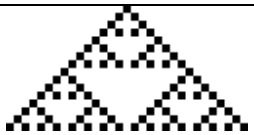
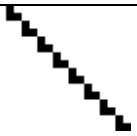
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
86		54 (1)	30	54 (1)	54 (1)	30	30	30
87		23	23	23	23	23	7	7
88		16	2	16	16	2	2	2
89		55 (1)	75	55 (1)	55 (1)	75	75	75
90		18	18	18	18	18	18	18
91		56 (1)	40 (1)	56 (1)	56 (1)	40 (1)	37	37
92		57 (1)	78	57 (1)	57 (1)	78	78	78
93		58 (1)	79	58 (1)	58 (1)	79	13	13
94		59 (1)	41 (1)	59 (1)	59 (1)	41 (1)	33 (1)	33 (2)
95		23	23	23	23	23	7	7
96		0	0	0	0	0	0	0

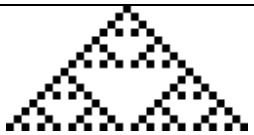
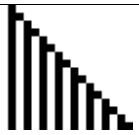
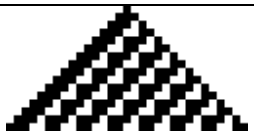
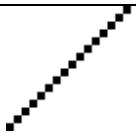
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
97		60 (1)	41	60 (1)	60 (1)	41	41	41
98		2	2	2	2	2	2	2
99		61 (1)	57	61 (1)	61 (1)	57	57	57
100		4	4	4	4	4	4	4
101		62 (1)	45	62 (1)	62 (1)	45	45	45
102		63 (1)	60	63 (2)	63 (2)	60	60	60
103		64 (1)	61	64 (1)	64 (1)	61	25	25
104		0	0	0	0	0	0	0
105		65 (1)	42 (1)	65 (1)	65 (1)	42 (1)	34 (1)	34 (1)
106		2	2	2	2	2	2	2
107		66 (1)	43 (2)	66 (1)	66 (1)	43 (2)	35 (2)	35 (2)

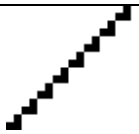
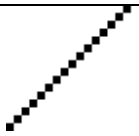
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
108		4	4	4	4	4	4	4
109		67 (1)	44 (1)	67 (1)	67 (1)	44 (1)	36 (1)	36 (1)
110		68 (1)	45 (2)	68 (1)	68 (1)	45 (2)	37 (2)	37 (2)
111		69 (1)	46 (2)	69 (1)	69 (1)	46 (2)	9	9
112		16	2	16	16	2	2	2
113		81	11	81	81	11	11	11
114		50	50	50	50	50	50	50
115		70 (1)	59	70 (1)	70 (1)	59	3	3
116		84	14	84	84	14	14	14
117		71 (1)	47	71 (1)	71 (1)	47	11	11
118		72 (1)	62	72 (1)	72 (1)	62	62	62

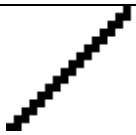
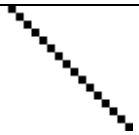
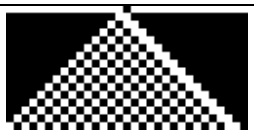
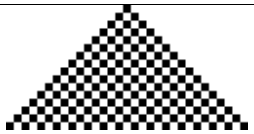
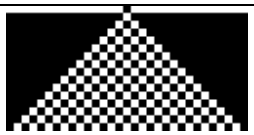
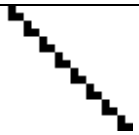
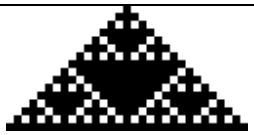
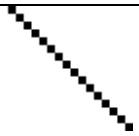
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
119		23	23	23	23	23	7	7
120		16	2	16	16	2	2	2
121		73 (1)	107	73 (1)	73 (1)	107	107	107
122		50	50	50	50	50	50	50
123		74 (1)	47 (1)	74 (1)	74 (1)	47 (1)	1	1
124		75 (1)	110	75 (1)	75 (1)	110	110	110
125		76 (1)	111	76 (1)	76 (1)	111	9	9
126		77 (1)	48 (1)	77 (3)	77 (3)	48 (3)	38 (3)	38 (3)
127		23	23	23	23	23	7	7
128		0	0	0	0	0	0	0
129		78 (2)	49 (2)	126	126	126	126	126

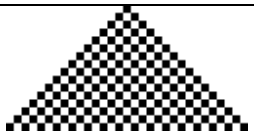
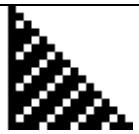
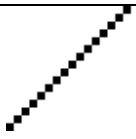
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
130		2	2	2	2	2	2	2
131		79 (1)	50 (2)	78 (1)	78 (1)	49 (2)	39 (2)	39 (2)
132		4	4	4	4	4	4	4
133		80 (1)	51 (1)	79 (1)	79 (1)	50 (1)	40 (1)	94
134		6	6	6	6	6	6	6
135		81 (1)	52 (2)	80 (1)	80 (1)	51 (2)	30	30
136		0	0	0	0	0	0	0
137		82 (1)	53 (2)	81 (1)	81 (1)	52 (2)	41 (2)	40 (2)
138		2	2	2	2	2	2	2
139		83 (2)	54 (4)	82 (2)	82 (2)	53 (4)	14	14
140		4	4	4	4	4	4	4

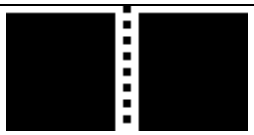
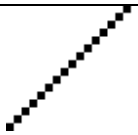
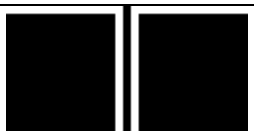
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
141		84 (1)	55 (2)	83 (1)	83 (1)	54 (2)	42 (2)	41 (2)
142		14	14	14	14	14	14	14
143		85 (1)	56 (2)	84 (1)	84 (1)	55 (2)	14	14
144		16	2	16	16	2	2	2
145		86 (1)	131	85 (1)	85 (1)	131	131	131
146		18	18	18	18	18	18	18
147		87 (1)	57 (1)	86 (1)	86 (1)	56 (1)	54	54
148		20	6	20	20	6	6	6
149		88 (1)	135	87 (1)	87 (1)	135	30	30
150		89 (1)	58 (1)	88 (1)	88 (1)	57 (1)	43 (1)	42 (1)
151		90 (8)	59 (8)	0	0	0	0	0

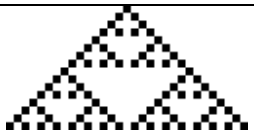
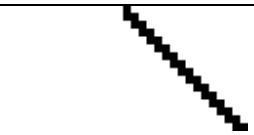
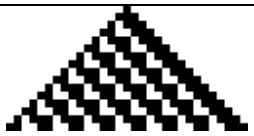
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
152		16	2	16	16	2	2	2
153		91 (1)	60 (2)	102	60	60	60	60
154		18	18	18	18	18	18	18
155		92 (1)	61 (2)	89 (1)	89 (1)	58 (2)	6	6
156		28	28	28	28	28	28	28
157		93 (1)	62 (2)	90 (1)	90 (1)	59 (2)	28	28
158		94 (1)	63 (2)	91 (1)	91 (1)	60 (2)	44 (2)	43 (2)
159		151	151	0	0	0	0	0
160		0	0	0	0	0	0	0
161		129	129	126	126	126	126	126
162		2	2	2	2	2	2	2

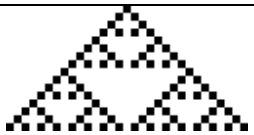
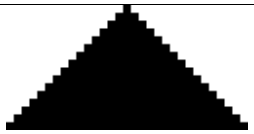
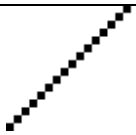
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
163		95 (1)	64 (2)	92 (1)	92 (1)	61 (2)	45 (2)	44 (2)
164		4	4	4	4	4	4	4
165		96 (1)	65 (1)	18	18	18	18	18
166		6	6	6	6	6	6	6
167		97 (1)	66 (2)	93 (1)	93 (1)	62 (2)	18	18
168		0	0	0	0	0	0	0
169		98 (1)	67 (2)	94 (1)	94 (1)	63 (2)	46 (2)	45 (2)
170		2	2	2	2	2	2	2
171		139	139	139	139	139	14	14
172		4	4	4	4	4	4	4
173		99 (1)	68 (2)	95 (1)	95 (1)	64 (2)	2	2

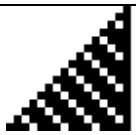
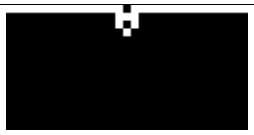
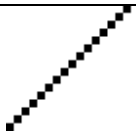
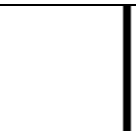
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
174		14	14	14	14	14	14	14
175		100 (1)	69 (2)	96 (1)	96 (1)	65 (2)	2	2
176		16	2	16	16	2	2	2
177		101 (1)	163	97 (1)	97 (1)	163	163	163
178		50	50	50	50	50	50	50
179		102 (1)	70 (1)	98 (1)	98 (1)	66 (1)	50	50
180		20	6	20	20	6	6	6
181		103 (1)	167	99 (1)	99 (1)	167	18	18
182		104 (1)	71 (1)	100 (1)	100 (1)	67 (1)	47 (1)	46 (1)
183		151	151	0	0	0	0	0
184		16	2	16	16	2	2	2

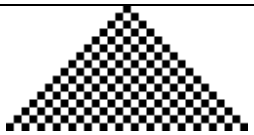
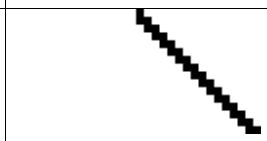
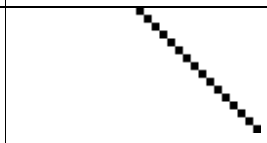
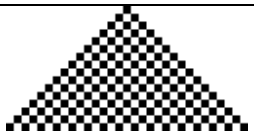
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
185		105 (1)	72 (2)	101 (1)	101 (1)	68 (2)	48 (2)	47 (2)
186		50	50	50	50	50	50	50
187		106 (1)	73 (2)	102 (1)	102 (1)	69 (2)	2	2
188		107 (1)	74 (2)	103 (1)	103 (1)	70 (2)	49 (2)	48 (2)
189		108 (1)	75 (2)	2	16	2	2	2
190		109 (1)	76 (2)	104 (1)	104 (1)	71 (2)	50 (2)	49 (2)
191		151	151	0	0	0	0	0
192		0	0	0	0	0	0	0
193		110 (1)	137	105 (1)	105 (1)	137	137	137
194		2	2	2	2	2	2	2
195		111 (1)	153	60	102	60	60	60

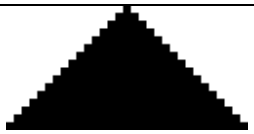
№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
196		4	4	4	4	4	4	4
197		112 (1)	141	106 (1)	106 (1)	141	141	141
198		70	28	70	70	28	28	28
199		113 (1)	157	107 (1)	107 (1)	157	28	28
200		0	0	0	0	0	0	0
201		114 (1)	77 (1)	108 (1)	108 (1)	72 (1)	51 (1)	50 (1)
202		2	2	2	2	2	2	2
203		115 (1)	78 (2)	109 (1)	109 (1)	73 (2)	4	4
204		4	4	4	4	4	4	4
205		116 (1)	79 (1)	110 (1)	110 (1)	74 (1)	52 (1)	51 (1)
206		117 (2)	80 (4)	111 (2)	111 (2)	75 (4)	53 (4)	52 (4)

№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
207		118 (1)	81 (2)	112 (1)	112 (1)	76 (2)	4	4
208		16	2	16	16	2	2	2
209		119 (2)	139	113 (2)	113 (2)	139	14	14
210		18	18	18	18	18	18	18
211		120 (1)	155	114 (1)	114 (1)	155	6	6
212		84	14	84	84	14	14	14
213		121 (1)	143	115 (1)	115 (1)	143	14	14
214		122 (1)	158	116 (1)	116 (1)	158	158	158
215		151	151	0	0	0	0	0
216		16	2	16	16	2	2	2
217		123 (1)	203	117 (1)	117 (1)	203	4	4

№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
218		18	18	18	18	18	18	18
219		124 (1)	82 (1)	4	4	4	4	4
220		125 (2)	206	118 (2)	118 (2)	206	206	206
221		126 (1)	207	119 (1)	119 (1)	207	4	4
222		127 (2)	83 (2)	120 (2)	120 (2)	77 (2)	54 (2)	53 (2)
223		151	151	0	0	0	0	0
224		0	0	0	0	0	0	0
225		128 (1)	169	121 (1)	121 (1)	169	169	169
226		2	2	2	2	2	2	2
227		129 (1)	185	122 (1)	122 (1)	185	185	185
228		4	4	4	4	4	4	4

№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
229		130 (1)	173	123 (1)	123 (1)	173	2	2
230		131 (1)	188	124 (1)	124 (1)	188	188	188
231		132 (1)	189	16	2	2	2	2
232		0	0	0	0	0	0	0
233		133 (1)	84 (1)	125 (1)	125 (1)	78 (1)	55 (1)	0
234		2	2	2	2	2	2	2
235		134 (1)	85 (2)	126 (1)	126 (1)	79 (2)	0	0
236		4	4	4	4	4	4	4
237		135 (1)	86 (1)	127 (1)	127 (1)	80 (1)	0	0
238		206	206	206	206	206	206	206
239		136 (1)	87 (2)	128 (1)	128 (1)	81 (2)	0	0

№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
240		16	2	16	16	2	2	2
241		209	139	209	209	139	14	14
242		50	50	50	50	50	50	50
243		137 (1)	187	129 (1)	129 (1)	187	2	2
244		84	14	84	84	14	14	14
245		138 (1)	175	130 (1)	130 (1)	175	2	2
246		139 (1)	190	131 (1)	131 (1)	190	190	190
247		151	151	0	0	0	0	0
248		16	2	16	16	2	2	2
249		140 (1)	235	132 (1)	132 (1)	235	0	0
250		50	50	50	50	50	50	50

№	Структура	*E*	*EM*	*EI*	*E*(*I*+*M*)	*EIM*	*EIMO*	*EIMOL6*
251		141 (1)	88 (1)	133 (1)	133 (1)	82 (1)	0	0
252		220	206	220	220	206	206	206
253		142 (1)	239	134 (1)	134 (1)	239	0	0
254		222	222	222	222	222	222	222
255		151	151	0	0	0	0	0