О детерминированной параллельной реализации метода ветвей и границ на монотонных объектах

А.И. Адамович[1], Анд.В. Климов[2]

1 Институт программных систем им. А.К. Айламазяна РАН
2 Институт прикладной математики им. М.В. Келдыша РАН

Аннотация. Статья продолжает работу авторов по разработке двухуровневой системы детерминированного параллельного программирования и создания библиотек так называемых *монотонных* классов, реализуемых на универсальном объектно-ориентированном языке, составляющем нижний уровень системы. Входной язык подсистемы верхнего уровня похож на функциональные языки с дополнительной возможностью создания и использования неизменяемых и монотонных объектов. Библиотеки монотонных классов гарантируют, что все программы на подъязыке верхнего уровня, использующие только монотонные классы, являются детерминированными и идемпотентными при их распараллеливании классическим способом — асинхронным вызовом всех функций. Существенная часть разработки такой системы — создание библиотек монотонных классов для различных предметно-ориентированных областей и образов решения прикладных задач с их помощью. Данная статья посвящена конкретной задаче реализации поиска минимального веса пути на графе с дугами с неотрицательными весами методом ветвей и границ. Приводится решение, названное *условно монотонным*, где монотонность выполняется для неотрицательных весов дуг. Ставится задача точной реализации метода ветвей и границ на монотонных объектах.

Ключевые слова: модели параллельных вычислений, детерминированные программы, метод ветвей и границ

On deterministic parallel implementation of the branch-and-bound method on monotonic objects

A.I. Adamovich[1], And.V. Klimov[2]

1 Ailamazyan Program Systems Institute of RAS
2 Keldysh Institute of Applied Mathematics of RAS

Abstract. This paper continues the authors' research into development of a two-level system of deterministic parallel programming and creation of libraries of so-called monotone classes implemented in a universal object-oriented language, which

Подана в сб. трудов XVIII Всероссийской научной конференции «Научный сервис в сети Интернет» (Новороссийск, 23–28 сентября 2019 г.). — http://agora.guru.ru/display.php?conf=abrau2019 https://keldysh.ru/abrau/2019/

constitutes the lower level of the system. The input language of the higher-level subsystem is like a functional language with the additional possibility of creating and using immutable and monotonic objects. The libraries of monotonic classes ensure that all programs in the higher-level sublanguage that use only monotonic classes are deterministic and idempotent when they are parallelized in the classical way by asynchronous calls of all functions. An essential part of the development of such a system is creation of the libraries of monotonic classes for various specific domains and examples of solving particular problems with their use. The paper is devoted to the specific task of implementing the search for the minimum weight of paths in a graph with edges labeled with nonnegative weights by the branch-and-bound method. A solution referred to as *conditionally monotonic* is given, where the monotonicity holds for nonnegative arc weights. The problem of the exact implementation of the branch-and-bound method on monotonic objects is stated.

Keywords: models of parallel computation, deterministic programs, monotonic objects, branch-and-bound method

1. Введение

Общая задача. Задача, обсуждаемая в этой статье, принадлежит следующему общему направлению: построение системы и библиотеки классов для *детерминированного* параллельного и конкурентного программирования на основе объектно-ориентированных языков.

Разрабатываемая авторами система предназначена для программирования тех (далеко не всех) задач, которые по своему смыслу дают детерминированный результат, и для той их части, которые допускают параллельную (и конкурентную, *англ*. concurrent) реализацию в виде детерминированной программы средствами данной системы. Речь об объектно-ориентированном программировании на языках типа Java с использованием потоков управления (*англ*. threads) включая «легкие» потоки, «файберы» (*англ*. fibers), и сопрограммы. В настоящее время принципы построения системы *не* ориентированы на численные задачи со специфическими средствами их программирования на базе MPI, OpenMP и прочих, и не пересекаются с областью применения и средствами, например, языка Норма [7] для декларативного и детерминированного программирования таких задач.

Ставится цель обеспечивать прикладного программиста объектно-ориентированными языковыми средствами и библиотеками, такими что при программировании с их использованием любая программа оказывается гарантированно детерминированной и хорошо масштабируемой на параллельной аппаратуре. Однако, не существует конечного и полного набора средств, которые можно раз и навсегда зафиксировать в системе и предоставить на языке верхнего уровня, чтобы все программы на нем были детерминированными и использовали всё многообразие средств параллельного программирования. Поэтому система должна быть открытой в смысле удобства создания новых проблемно-ориентированных библиотек.

Наблюдаемый по мировой литературе опыт разработки разнообразных средств детерминированного параллельного программирования показывает (см., например, наш обзор [5]), что они реализуются на низкоуровневых языках (вплоть до C и VHDL), предоставляя прикладному программисту высокоуровневые средства[1]. Получается, чтобы реализовать новые средства, необходимо каждый раз опускаться на низкий уровень с высокими умо- и трудозатратами.

Возникает задача поднять инструменты разработки системных библиотек на достаточно высокий уровень объектно-ориентированного языка типа Java, предоставляя прикладному разработчику подмножество средств (проверяемое компилятором и средой разработки), на котором ему будет гарантирована детерминированность его программ. Для продвижения в этом направлении в статьях [5,6] предложен проект двухуровневой системы программирования (который поясним в разделе 2).

Данный доклад продолжает серию работ авторов [2-6,11] по детерминированному параллельному программированию и основывается на более ранних заделах по Т-системе [1] и монотонным объектам [8,12]. Используемое нами понятие *монотонных объектов и классов* включает в себя два условия: к требованию *детерминированности* программ добавляется их *идемпотентность*, имеющее большое практическое значение для обеспечения отказоустойчивости параллельных и распределенных программ.

Мы разделяем цели авторов статьи с самоговорящим названием [14], но предлагаем для решения другие инструменты. Наш подход перекликается с работой [17,18] по детерминированности параллельных вычислений с общими данными на *полурешетках*[2], но ставит целью обобщить его на более универсальные средства в рамках объектно-ориентированных языков.

Наша работа также делает вклад в решение еще одной фундаментальной проблемы: построение модели вычислений, промежуточной между функциональной и объектно-ориентированной, такой что в ней есть явные ссылки на объекты, объекты могут изменяться (хотя и некоторым дисциплинированным образом), но при этом выполняются основные свойства функциональных языков: детерминированность, идемпотентность, наличие денотационной семантики (в том же стиле, каком дается классическая семантика функциональных языков) и некоторая адекватная замена референциальной прозрачности (*англ.* referential transparency), которая в классическом виде нарушена операторами создания объектов, порождающими новые ссылки, и побочными эффектами.

---

[1] Показательный пример: библиотеки параллельного и конкурентного программирования на чисто-функциональном (да еще ленивом, *англ.* lazy) языке Haskell [19] реализованы с использованием системных средств, не изобразимых на Haskell'е.

[2] https://en.wikipedia.org/wiki/Semilattice

Конкретная задача. Существенная часть системы — библиотека монотонных классов для различных предметно-ориентированных областей, использование которой гарантирует детерминированность и идемпотентность. Эта библиотека копится в процессе решения образцов прикладных задач.

Первый круг понятий, требуемый во многих задачах, — средства для эффективного представления графов. Они естественны и не вызывают затруднений при обычном объектно-ориентированном программировании, но нетривиальны при реализации через монотонные объекты. В работе [11] на примерах объясняются проблемы и демонстрируются способы построения циклических структур данных на монотонных объектах, так чтобы для представления отношений между вершинами и дугами графов использовались явные ссылки, как в объектно-ориентированном программирования, но что недоступно в чисто-функциональных языках (где основная структура данных — деревья[3]). Эти средства универсальны и нужны для многих программ.

В этом докладе мы рассмотрим специфическую задачу — поиск минимального веса простых[4] путей в графе из заданной вершины. На дугах графа заданы неотрицательные веса, а вес пути определяется как сумма весов составляющих его дуг. Эта задача может иметь много вариаций. Распространенный вариант — рассматриваются пути, достигающие второй заданной вершины. В экспериментальном исследовании для удобства изучения масштабируемости мы взяли другую версию — рассматриваются все простые пути фиксированной длины из заданной вершины, не фиксируя их конца.

Эта задача хорошо распараллеливается и является детерминированной по смыслу. Процессы, перебирающие различные пути, общаются через общую переменную, хранящую минимальный вес, найденный к настоящему моменту. Процесс, нашедший более короткий путь, присваивает ей новое значение. Поскольку минимум на конечном множестве чисел однозначен, чтобы доказать корректность и детерминированность его реализации конкретной программной, нужно убедиться, что посещаются все пути или достаточное их число.

Классический способ решения этой задачи — метод ветвей и границ[5]. Его «изюминка» — операция *отсева* (*англ*. pruning), прекращающая перебор путей, когда вес пройденного пути превзойдет или станет равен текущему минимуму. Здесь основная тонкость в том, что операция сравнения с текущим минимумом

---

[3] Показательна работа [13], решающая проблему представления и обхода графов наиболее эффективным и адекватным для функционального языка Haskell способом. В ней дается эталонное представление графов в виде списков вершин и дуг (то есть в виде деревьев), а потом сообщается, что представление с тем же набором операций, видимом из языка, реализовано средствами нижнего уровня *для эффективности*.

[4] В теории графов *простым* называется путь без повторных посещений вершин.

[5] https://en.wikipedia.org/wiki/Branch_and_bound

немонотонна, и потому не может явно вызываться из подъязыка верхнего уровня. Поэтому она должна быть «спрятана» внутрь монотонного объекта.

Основные результаты. Мы предлагаем реализацию метода ветвей и границ, удовлетворяющую следующим требованиям:

- алгоритм верхнего уровня написан на функциональном подъязыке и его распараллеливание осуществляется асинхронными вызовами всех (или хотя бы части) функций;
- для взаимодействия процессов используются монотонные объекты (в данном решении — один монотонный объект с текущим минимумом);
- детерминированность алгоритма верхнего уровня гарантируется реализацией монотонных классов (в данному случае одного) при написании любого кода на «верхнем» подъязыке;
- различие между версиями программы с полным перебором путей и с отсевом методом ветвей и границ реализуются в одном месте внутри монотонного объекта; это, в частности, упрощает доказательство их эквивалентности и, тем самым, корректности программы с отсевами.

Решение демонстрирует применение монотонного объекта высшего порядка, то есть с операцией, принимающей лямбда-выражение в качестве аргумента. Мы полагаем, что без высшего порядка столь же адекватную схему реализации метода ветвей и границ на монотонных объектах построить невозможно.

План оставшейся части статьи. В разделе 2 объясняются цели и принципы построения двухуровневой системы детерминированного параллельного программирования. В разделе 3 дается определение монотонных объектов и классов. Раздел 4 — основной: в нем приводится программа поиска минимального веса путей из заданной вершины графа, реализованная согласно нашему подходу. В разделе 5 мы выясняем, что это решение не удовлетворяет в точности нашим условиям монотонности, а является *условно* монотонным, завися от неотрицательности весов в графе, и объясняем, почему следует искать точное решение. Близкие предшествующие работы обсуждаются в разделе 6. Заключительный раздел 7 содержит основные выводы.

2. Двухуровневая система детерминированного параллельного программирования

Принципы разрабатываемой системы детерминированного параллельного программирования применимы для любого развитого объектно-ориентированного языка, удовлетворяющего следующий требованием:

- У объектно-ориентированного языка должно быть функциональное подмножество. Наличие соответствующего синтаксического сахара (например, лямбда-выражений) желательно, но не обязательно. Должна быть возможность реализовать средства проверки, что программа

принадлежит функциональному подмножеству (без этого потеряются гарантии детерминизма), или следует разработать специализированный язык, отображаемый в функциональное подмножество.

- Семантика функциональных языков требует наличие автоматического управления памяти и сборки мусора. Без этого функциональный стиль программирования невозможен. Например, подходят языки на платформах JVM и MS.NET.
- Должны присутствовать средства параллельного и конкурентного программирования: создания процессов, позволяющие реализовать асинхронный (параллельный) вызов функций, и средства синхронизации при доступе к общим данным.

Мы проводим прототипирование системы, используя платформу JVM[6], языки Java и Kotlin[7], язык собственной разработки Ajl [3], библиотеку Quasar[8], реализующую легких потоки («файберы») на JVM, отслеживаем развитие проекта Loom[9], идущего на смену Quasar'у с целью более тесной интеграции с языком Java и последующего включения стандартную Java-библиотеку. Языковые средства разрабатываем в IDE[10] Eclipse[11].

Поскольку цель проекта двоякая — предоставить и гарантированное детерминированное программирование разработчику приложений, и открытый нижний уровень реализации, на котором эксперты по параллельному программированию могут использовать все средства базового объектно-ориентированного языка, — входной язык должен четко подразделяться на две части — гарантированно детерминированную и недетерминированную:

- Верхний уровень — детерминированная часть: код на подмножестве языка, который пишет прикладной программист. Ему гарантируется детерминированность любой программы, использующей заготовленные библиотеки нижнего уровня.
- Нижний уровень — потенциально не детерминированная часть: библиотеки классов, создаваемые квалифицированными программистами для определенных областей применения на универсальном объектно-ориентированном языке с возможностью использования всех средств недетерминированного параллельного программирования. Авторы библиотек гарантируют детерминированность параллельных программ пользователям, кодирующим на функциональном подъязыке верхнего уровня.

---

[6] JVM = Java Virtual Machine.
[7] http://kotlin.jetbrains.org/
[8] https://github.com/puniverse/quasar
[9] https://wiki.openjdk.java.net/display/loom/Main
[10] IDE = Integrated Development Environment.
[11] https://www.eclipse.org/

3. Понятие монотонного объекта и класса

Объекты и классы, определяемые на языке нижнего уровня и используемые на функциональном подъязыке верхнего уровня, мы называем *монотонными*, поскольку, как оказывается, их состояние меняется монотонно на некоторой полурешетке, которую, как правило, можно построить по коду класса. Однако формальное определение, которое удобно использовать в рассуждениях о программах и доказательствах их свойств, мы даем операционно следующим образом [6,8,12].

Монотонные объекты и классы таковы, что при их создании и использовании в любой программе на функциональном подъязыке все выражения удовлетворяют следующим двум свойствам:

- *Детерминированность*: результаты, полученные в любом порядке параллельных вычислений копий выражения от одного и того же начального состояния, эквивалентны, а побочные эффекты не отличимы программно на подъязыке верхнего уровня.
- *Идемпотентность*: повторное вычисление копии выражения от состояния, полученного при первом вычислении, или параллельно с ним, дает эквивалентный результат и побочный эффект, не отличимый программно на подъязыке верхнего уровня от побочного эффекта однократного вычисления.

Эти свойства должны выполняться одновременно для всех монотонных классов, когда они используются вместе в любой функциональной программе.

В настоящее время мы конструируем монотонные классы, рассуждая об этих свойствах неформально и наивно. В будущем мы рассчитываем, что удастся разработать средства автоматизации доказательств монотонности в подавляющем числе случаев, хотя, конечно, полностью автоматических компьютерных инструментов на все случаи жизни никогда построено не будет, как и в общем случае средств верификации программ.

4. Детерминированная параллельная реализация метода ветвей и границ с монотонным объектом

Рассмотрим детерминированную параллельную реализацию задачи поиска минимального веса пути в графе из заданной вершины. На рис. 1 изображена версия, которая еще не удовлетворяет всем требованиям монотонности. Оставшиеся проблемы обсудим в следующем разделе 5.

Мы имеем следующие классы (перечислены только те операции[12], которые используются в алгоритме на рис. 1):

---

[12] Мы избегаем использования объектно-ориентированного термина «метод» для операций классов и называем их *функциям* (со значением), *процедурами* (без значения) и *операциями в* классах и объектах, чтобы не путаться с понятием метода «в высоком смысле», например, в словах «метод

- Представление графа *Graph* с вершинами *Vertex* и дугами *Edge*. Единственный экземпляр графа хранится в глобальной переменной *graph*. Объекты классов *Vertex* и *Edge* обращаются к ней для выполнения своих операций.
- Класс *Vertex* для представления вершин с операцией:
  - функция *vertex.outgoingEdges* — множество (*enumeration* в терминах Java) дуг, исходящих из вершины *vertex*.
- Класс *Edge* для представления дуг с операцией:
  - функция *edge.endVertex* — конечная вершина дуги *edge* (то есть вершина, для которой эта дуга входящая).
- Класс *Path* для представления путей в графе с операциями:
  - предикат *path.isComplete* — является ли путь *path* завершенным (например, имеет заданную длину или достиг заданной вершины — в зависимости от решаемой задачи);
  - предикат *path.contains*(*vertex*) — содержит ли путь *path* вершину *vertex*;
  - функция *path.weight* — вес пути *path* (сумма весов входящих в него дуг);
  - функция *path.lastVertex* — последняя вершина пути *path*;
  - функция *path.extendedWith*(*edge*) — объект-путь, полученный добавлением дуги *edge* в конце пути *path*.
- Монотонный класс *Minimizer* с единственным экземпляром объекта в глобальной переменной *minimizer* с операциями:
  - функция *minimizer.minimum* — выдать минимальное значение, *после того как* процедура *findMinimalPathWeight* завершится (см. обсуждение ниже);
  - процедура *minimizer.addValue*(*value*) — добавить число *value* к множеству, из которого берется минимум;
  - процедура *minimizer.branch*(*pathWeight*, *step*) — продолжение вычислений (*англ*. continuation) *step*. Здесь *step* — это лямбда-выражение без аргументов и значения. В варианте с отсевом *step* не вычисляется, если *pathWeight* больше уже накопленного минимума.

На рис. 1 изображен в Java-подобном синтаксисе (опуская фигурные скобки и точки с запятой) алгоритм поиска минимального веса пути *findMinimalPathWeight* от заданной вершины *root* среди завершенных путей согласно предикату *path.isComplete*. Процедура *findMinimalPathWeight* не имеет

ветвей и границ». Для вызова функций и процедур используем объектно-ориентированный синтаксис: *x.f* вместо *f*(*x*) и *x.f*(*y*) вместо *f*(*x*, *y*).

Входная информация и начальное состояние

- *Graph graph* — граф — глобальная переменная, используемая в коде классов *Vertex* и *Edge*
- *Vertex root* — начальная вершина графа
- *Minimizer minimizer* — глобальная переменная с монотонным объектом — вычислителем минимального значения в начальном состоянии «отрицательная бесконечность»

Начальное обращение к рекурсивной процедуре *findMinimalPathWeight*

- *findMinimalPathWeight*(*root*)

Рекурсивная процедура *findMinimalPathWeight*

```
findMinimalPathWeight(Path path)
    if (path.isComplete)
        minimizer.addValue(path.weight)
    else
        minimizer.branch(path.weight, () ->
            for (Edge edge : path.lastVertex.outgoingEdges)
                if (!path.contains(edge.endVertex))
                        findMinimalPathWeight(path.extendedWith(edge))
        )
```

Выдача результата после завершения процедуры *findMinimalPathWeight*

- *minimizer.minimum* — выдача минимального значения

Рис. 1. Алгоритм поиска минимального веса пути в графе с монотонным объектом *minimizer*

результата и накапливает ответ побочными эффектами — записью *path.weight* в монотонный объект *minimizer* вызовами *minimizer.addValue*(*path.weight*). Этот алгоритм может выполняться последовательно, но он легко распараллеливается, запуская все вызовы процедуры *findMinimalPathWeight* асинхронно и ожидая завершения всех процессов *findMinimalPathWeight* перед выдачей результата функцией *minimizer.minimum*.

Для знатоков метода ветвей и границ алгоритм на рис. 1 выглядит естественно и как бы не содержит ничего хитрого. Действительно, он близок к приведенному в Википедии[13] с точностью до перестановок некоторых фрагментов кода. Однако в нем есть процедура высшего порядка *minimizer.branch* со вторым аргументом *step* — лямбда выражением. Оно не имеет явных аргументов, принимая значения *path* и *minimizer* из объемлющего контекста. Обратите внимание на синтаксис «**() ->**», выделенный полужирным.

[13] https://en.wikipedia.org/wiki/Branch_and_bound

Приведенный код годится как для полного перебора всех путей, так и с отсевом методом ветвей и границ. Выбор версии делается в коде операции *minimizer.branch*: при полном переборе *step* всегда выполняется, при отсеве — не выполняется, если первый аргумент — вес текущего пути *path.weigh*t больше или равен текущему минимальному весу пройденных путей.

В объекте *minimizer* есть семантическая тонкость, требуемая понятием монотонности: результат *minimizer.minimum* должен выдаваться только после завершения всех процессов *findMinimalPathWeight*. В практических реализациях метода ветвей и границ здесь ставят барьер, что не соответствует чистой теории монотонных объектов, хотя, конечно, локальные отклонения от «высокой теории» — нормальная вещь на практике. К миру монотонных объектов лучше подойдут такие варианты решений:

1. Вариант 1, прагматический. Асинхронный вызов процедуры возвращает спец-объект-handler, операция *waitAll* которого обеспечивает ожидание завершения всех процессов, созданных данным вызовом, — его самого и всех рекурсивно им порожденных.
2. Вариант 2, соответствующий теории. Различаются два вида ссылок на монотонные объекты: одни — по которым возможна «запись» в объект, т.е. побочный эффект любого вида, другие — по которым можно только читать (извлекать информацию без видимого изменения объектов). Тогда процедура *findMinimalPathWeight* будет использовать ссылку «на запись» в объект *minimizer*, а функция *minimizer.minimum* — ссылку «только на чтение». Операции по ссылке «на чтение» завершаются выдачей результата, когда все ссылки на запись исчезнут (или по каким-либо семантическим соображениям станет ясно, что результат данной операции уже не может измениться). Это можно реализовать либо счетчиками ссылок, либо «слабыми ссылками» (*англ*. weak references) и сборкой мусора. Вариант со счетчиками ссылок годится только в предположении, что не образуются циклические структуры (что имеет место в нашей задаче, так как в монотонном объекте *minimizer* хранится число, а не ссылки на другие объекты). Вариант со сборкой мусора хуже тем, что надо дожидаться, когда она произойдет, или принудительно вызывать ее, осуществляя лишнюю работу, собирая весь мусор, хотя нужно лишь выяснить вопрос наличия ссылок на часть объектов. Оба решения не нарушают монотонности.

5. Немонотонность приведенной версии алгоритма

Теперь мы должны сделать основное утверждение, что класс *Minimizer* с приведенной семантикой является монотонным, то есть любая функциональная программа, его использующая, будет детерминированной и идемпотентной. Однако, мы можем сделать это применительно только первому из упомянутых

выше варианту операции *minimizer.branch*, когда *step* всегда выполняется и, тем самым, производится полный перебор.

В версии с отсевом согласно методу ветвей и границ «кто-то» должен гарантировать, что рекурсивные вызовы *findMinimalPathWeight* в коде лямбда-выражения для *step* заведомо увеличивают вес пути. Сейчас этот «кто-то» — это либо предусловие, что веса дуг неотрицательны, или функция *path.extendedWith*(*edge*), проверяющая, что вес дуги *edge* неотрицателен и, тем самым, вес пути не уменьшается. В обоих случаях гарантии даются за пределами класса *Minimizer*, а наше определение монотонности требует детерминированности и идемпотентности *любой* программы верхнего уровня. Поэтому это решение можно назвать лишь *условно монотонным*, то есть монотонным при гарантии выполнения некоторого внешнего условия.

В данной статье мы оставляем вопрос построения действительно монотонной реализации метода ветвей и границ открытым. Это задача на будущее.

6. Близкие работы

Обзор известных нам работ по детерминированному параллельному программированию приведен в статье [5]. Здесь остановимся только на двух показательных направлениях.

Первый подход имеет близкую к нашей цель — детерминированное параллельное программирование на объектно-ориентированном языке, но принципиально отличается по методам ее достижения. В серии работ, из которых выделим одну с ярким названием-лозунгом [14], их авторы разрабатывают методы статического анализа программ на языке Java с некоторыми расширениями, названном Deterministic Parallel Java, DPJ. Этот анализ выявляет случаи, когда побочный эффект не приводит к недетерминированности. Расширения по сравнению с языком Java дают возможность программисту указывать дополнительную информацию для анализа. Эти методы являются развитием с целью внедрения в практику долгой линии исследований по системам типизации, учитывающих побочные эффекты (effect systems[14]). Мы на них останавливаться не будем, так как в нашем подходе никакой статический анализ не используется (если не считать таковым выделение функционального подмножества объектно-ориентированного языка в качестве языка верхнего уровня в нашем двухуровневом подходе).

Второй подход близок к нашему и по целям, и по методам. Он основан на монотонном изменении общих переменных. Эта идея достаточно стара и реализовывалась в разных частных случаях. Общее в них то, что для взаимодействия параллельных процессов предоставляются специальные структуры данных с так определенными операциями, чтобы не нарушалась детерминированность. Приведем наиболее известные из них:

[14] https://en.wikipedia.org/wiki/Effect_system

– I-структуры (*англ*. I-structures) [10];
– сети Кана (*англ*. Kahn networks) [15];
– TStreams, Concurrent Collections [16];
– структуры данных на решетках (*англ*. lattice-based data structures) [17,18].

У этих (и нашего) подходов есть общая черта: переменные, объекты, через которые осуществляется взаимодействие параллельных процессов, меняют свое состояние монотонно, только вверх на некоторой полурешетке от неопределенного состояния ($\bot$) к «всё более определенному». При этом верхний элемент решетки ($\top$) обозначает «переопределено»; в программе это соответствует ошибке, выработке исключения. Например, множество значений I-структур с целыми числами описывается решеткой, называемой «плоской», состоящей из нижнего элемента «не определено» ($\bot$), не сравнимых между собой целых чисел и верхнего элемента «переопределено» ($\top$). При выполнении операции присваивания значения y в переменную со значением x в нее записывается наименьшая верхняя грань значений x и y. Если полученный результат оказывается верхним элементом $\top$, то вырабатывается исключение.

Эта идея была проработана в диссертации Lindsey Kuper [17] и в публикациях вместе с ее коллегами [18]. Они доказали детерминированность параллельных вычислений для процессов, взаимодействующих через переменные, принимающие значения из произвольной полурешетки. В нашей работе мы используем эти идеи, обобщая их на объекты, определяемые пользователем, с монотонно изменяющимся состоянием.

Отметим также в качестве нашей будущей работы, что следует сравнить технику верификации программ, реализующих метод ветвей и границ по предложенной схеме, и существующие подходы к доказательству ее корректности, например, описанные в статье [9].

7. Заключение

В статье обсуждается общая задача разработки библиотек так называемых *монотонных* классов в рамках двухуровневой объектно-ориентированной системы параллельного программирования. В этой системе язык верхнего уровня ограничен до близкого к функциональному языку, позволяющего распараллеливание асинхронными вызовами всех функций. Процессы взаимодействуют через монотонные объекты, классы которых определяются на языке нижнего уровня — полном объектно-ориентированном языке со средствами параллельного и конкурентного программирования. Реализация монотонных классов гарантирует *детерминированность* и *идемпотентность* всем программам на подъязыке верхнего уровня.

В рамках такой общей постановки данная статья посвящена решению частной задачи — реализации поиска минимального пути в графе среди путей исходящих из заданной вершины с помощью монотонного объекта, вычисляющего минимум среди переданных ему чисел. Приведено решение,

являющееся лишь *условно монотонным*, так как оно требует внешней проверки исходных данных, что веса на дугах неотрицательны. Поставлена (пока открытая) задача точной реализации метода ветвей и границ с помощью монотонных объектов, гарантируя детерминированность любой программы верхнего уровня, так чтобы все необходимые проверки делались внутри монотонных классов.

Работа выполнялась при финансовой поддержке Российской Федерации в лице Минобрнауки России (идентификатор № RFMEFI61319X0092).

Литература

1. Абрамов С.М., Адамович А.И., Коваленко М.Р. Т-система — среда программирования с поддержкой автоматического динамического распараллеливания программ. Пример реализации алгоритма построения изображений методом трассировки лучей // Программирование, 25:2, 1999. — С. 100–107.
2. Адамович А.И. Струи как основа реализации понятия Т-процесса для платформы JVM // Программные системы: теория и приложения, 6:4 (27), 2015. — С. 177–195. — doi:10.25209/2079-3316-2017-8-4-221-244 .
3. Адамович А.И. Язык программирования Ajl: автоматическое динамическое распараллеливание для платформы JVM // Программные системы: теория и приложения, 7:4 (31), 2016. — С. 83–117. — doi:10.25209/2079-3316-2016-7-4-83-117 .
4. Адамович А.И., Климов Анд.В. Об опыте использования среды метапрограммирования Eclipse/TMF для конструирования специализированных языков // Научный сервис в сети Интернет : Труды XVIII Всероссийской научной конференции. — М. : ИПМ им. М.В. Келдыша РАН, 2016. — С. 3–8. — doi:10.20948/abrau-2016-45 .
5. Адамович А.И., Климов Анд.В. Как создавать параллельные программы, детерминированные по построению? Постановка проблемы и обзор работ // Программные системы: теория и приложения, 2017, 8:4(35). — С. 221–244. — doi:10.25209/2079-3316-2017-8-4-221-244 .
6. Адамович А.И., Климов Анд.В. Подход к построению системы детерминированного параллельного программирования на основе монотонных объектов // Вестник СибГУТИ, 2019, № 3. — С. 14–26. — URL: http://vestnik.sibsutis.ru/showpapper.php?act=showpapper&id=870 .
7. Андрианов А.Н., Баранова Т.П., Бугеря А.Б., Ефимкин К.Н. Трансляция непроцедурного языка Норма для графических процессоров // Препринты ИПМ им. М.В. Келдыша, 2016, № 73. — 24 с. — doi:10.20948/prepr-2016-73 .
8. Климов Анд.В. Детерминированные параллельные вычисления с монотонными объектами // Научный сервис в сети Интернет: многоядерный компьютерный мир : Труды Всероссийской научной конференции. — М. : Изд-во Московского университета, 2007. — С. 212–217.

9. Шилов Н.В. Верификация шаблонов алгоритмов для метода отката и метода ветвей и границ // Моделирование и анализ информационных систем. 2011, 18(4). — С. 168–180. — URL: https://www.mais-journal.ru/jour/article/view/1107/820 .
10. Arvind, Nikhil R.S., Pingali K.K. I-structures: Data Structures for Parallel Computing // ACM Trans. Program. Lang. Syst., 11:4, 1989. — P. 598–632. — doi:10.1145/69558.69562 .
11. Adamovich A.I., Klimov And.V. Building Cyclic Data in a Functional-Like Language Extended with Monotonic Objects // X Workshop PSSV: Program Semantics, Specification and Verification: Theory and Applications (Novosibirsk, Akademgorodok, Russia, July 1–2, 2019) : Abstracts / V. Zakharov, N. Shilov, I. Anureev (eds.). — Novosibirsk : A.P. Ershov Institute of Informatics Systems, 2019. — P. 11-19. — ISBN 978-5-4437-0918-5.
12. Klimov And.V. Dynamic Specialization in Extended Functional Language with Monotone Objects // SIGPLAN Not., 26:9, 1991. — P. 199–210. — doi:10.1145/115865.376287 .
13. Erwig M. Inductive graphs and functional graph algorithms // J. Funct. Program. 11, 5, 2001, P. 467-492. — doi:10.1017/S0956796801004075 .
14. Bocchino R.L. (Jr.), Adve V.S., Adve S.V., Snir M. Parallel Programming Must Be Deterministic by Default // Fifth USENIX Conference on Hot Topics in Parallelism, HotPar'09. — USENIX Association, 2009. — P. 4–4 (6 pages).
15. Kahn G. The Semantics of Simple Language for Parallel Programming // IFIP Congress, 1974. — P. 471–475.
16. Burke M.G., Knobe K., Newton R., Sarkar V. Concurrent Collections Programming Model // Encyclopedia of Parallel Computing. — Springer US, 2011. — P. 364–371. — doi:10.1007/978-0-387-09766-4_238.
17. Kuper L. Lattice-based Data Structures for Deterministic Parallel and Distributed Programming. Ph.D. Thesis. — IN, USA : Indiana University, 2015. — 253 p.
18. Kuper L., Todd A., Tobin-Hochstadt S., Newton R.R. Taming the Parallel Effect Zoo: Extensible Deterministic Parallelism with LVish // ACM SIGPLAN Not., 49:6, 2014. — P. 2–14. — doi:10.1145/2666356.2594312 .
19. Marlow S. Parallel and Concurrent Programming in Haskell. — O'Reilly, 2013.

References

1. Abramov S.M., Adamovich A.I., Kovalenko M.R. T-sistema — sreda programmirovaniya s podderzhkoj avtomaticheskogo dinamicheskogo rasparallelivaniya programm. Primer realizacii algoritma postroeniya izobrazhenij metodom trassirovki luchej // Programmirovanie, 25:2, 1999. — P. 100–107.
2. Adamovich A.I. Strui kak osnova realizacii ponyatiya T-processa dlya platformy` JVM // Programmny`e sistemy`: teoriya i prilozheniya, 6:4 (27), 2015. — P. 177–195. — doi:10.25209/2079-3316-2017-8-4-221-244 .
3. Adamovich A.I. Yazy`k programmirovaniya Ajl: avtomaticheskoe dinamicheskoe rasparallelivanie dlya platformy` JVM // Programmny`e sistemy`: teoriya i

prilozheniya, 7:4 (31), 2016. — P. 83–117. — doi:10.25209/2079-3316-2016-7-4-83-117 .
4. Adamovich A.I., Klimov And.V. Ob opy`te ispol`zovaniya sredy` metaprogrammirovaniya Eclipse/TMF dlya konstruirovaniya specializirovanny`x yazy`kov // Nauchny`j servis v seti Internet : Trudy` XVIII Vserossijskoj nauchnoj konferencii. — M. : IPM im. M.V. Keldy`sha RAN, 2016. — P. 3–8. — doi:10.20948/abrau-2016-45 .
5. Adamovich A.I., Klimov And.V. Kak sozdavat` parallel`ny`e programmy`, determinirovanny`e po postroeniyu? Postanovka problemy` i obzor rabot // Programmny`e sistemy`: teoriya i prilozheniya, 2017, 8:4(35). — P. 221–244. — doi:10.25209/2079-3316-2017-8-4-221-244 .
6. Adamovich A.I., Klimov And.V. Podxod k postroeniyu sistemy` determinirovannogo parallel`nogo programmirovaniya na osnove monotonny`x ob``ektov // Vestnik SibGUTI, 2019, № 3. — P. 14–26. — URL: http://vestnik.sibsutis.ru/showpapper.php?act=showpapper&id=870 .
7. Andrianov A.N., Baranova T.P., Bugerya A.B., Yefimkin K.N. Translyaciya neprocedurnogo yazy`ka Norma dlya graficheskix processorov // Preprinty` IPM im. M.V. Keldy`sha, 2016, № 73. — 24 p. — doi:10.20948/prepr-2016-73 .
8. Klimov And.V. Determinirovanny`e parallel`ny`e vy`chisleniya s monotonny`mi ob``ektami // Nauchny`j servis v seti Internet: mnogoyaderny`j komp`yuterny`j mir : Trudy` Vserossijskoj nauchnoj konferencii. — M. : Izd-vo Moskovskogo universiteta, 2007. — P. 212–217.
9. Shilov N.V. Verifikaciya shablonov algoritmov dlya metoda otkata i metoda vetvej i granicz // Modelirovanie i analiz informacionny`x sistem, 18(4), 2011. — P. 168–180. — URL: https://www.mais-journal.ru/jour/article/view/1107/820 .
10. Arvind, Nikhil R.S., Pingali K.K. I-structures: Data Structures for Parallel Computing // ACM Trans. Program. Lang. Syst., 11:4, 1989. — P. 598–632. — doi:10.1145/69558.69562 .
11. Adamovich A.I., Klimov And.V. Building Cyclic Data in a Functional-Like Language Extended with Monotonic Objects // X Workshop PSSV: Program Semantics, Specification and Verification: Theory and Applications (Novosibirsk, Akademgorodok, Russia, July 1–2, 2019) : Abstracts / V. Zakharov, N. Shilov, I. Anureev (eds.). — Novosibirsk : A.P. Ershov Institute of Informatics Systems, 2019. — P. 11-19. — ISBN 978-5-4437-0918-5.
12. And.V. Klimov. Dynamic Specialization in Extended Functional Language with Monotone Objects // SIGPLAN Not., 26:9, 1991. — P. 199–210. — doi:10.1145/115865.376287 .
13. Erwig M. Inductive graphs and functional graph algorithms // J. Funct. Program. 11, 5, 2001, P. 467-492. — doi:10.1017/S0956796801004075 .
14. Bocchino R.L. (Jr.), Adve V.S., Adve S.V., Snir M. Parallel Programming Must Be Deterministic by Default // Fifth USENIX Conference on Hot Topics in Parallelism, HotPar'09. — USENIX Association, 2009. — P. 4–4 (6 pages).

15. Kahn G. The Semantics of Simple Language for Parallel Programming // IFIP Congress, 1974. — P. 471–475.
16. Burke M.G., Knobe K., Newton R., Sarkar V. Concurrent Collections Programming Model // Encyclopedia of Parallel Computing. — Springer US, 2011. — P. 364–371. — doi:10.1007/978-0-387-09766-4_238.
17. Kuper L. Lattice-based Data Structures for Deterministic Parallel and Distributed Programming. Ph.D. Thesis. — IN, USA : Indiana University, 2015. — 253 p.
18. Kuper L., Todd A., Tobin-Hochstadt S., Newton R.R. Taming the Parallel Effect Zoo: Extensible Deterministic Parallelism with LVish // ACM SIGPLAN Not., 49:6, 2014. — P. 2–14. — doi:10.1145/2666356.2594312 .
19. Marlow S. Parallel and Concurrent Programming in Haskell. — O'Reilly, 2013.