Визуализация графа потоков управления

А. А. Михайлов, А. Е. Хмельнов
Институт динамики систем и теории управления имени В.М. Матросова СО РАН
e-mail: mikhailov@icc.ru

В работе предложен метод визуализации графа потоков управления, позволяющий анализировать сложные графовые представления программ, полученные после обработки исходного кода компилятором, либо в процессе декомпиляции исполняемого кода. Метод основан на выделении в управляющем графе регионов с одним входным и одним выходным узлом с последующей их заменой на абстрактные узлы. Таким образом, в результате выполнения семантически эквивалентных преобразований исходный граф сворачивается в один абстрактный узел, содержащий в себе иерархию выделенных регионов, каждому из которых ставится в соответствие один из предопределенных шаблонов отображения. В итоге задача визуализации управляющего графа сводится к описанию правил отображения шаблонов. Предложенный метод позволяет выделять в управляющем графе подграфы соответствующие высокоуровневым операторам языков программирования, что дает возможность использовать изобразительные соглашения, принятые при рисовании блок-схем.

Ключевые слова: визуализация, управляющий граф, структурный анализ

Введение

В программировании графы являются одной из основных структур данных. Управляющий граф – это естественное представление программы, которое может быть вычислено автоматически как по исходному, так и по бинарному коду. Граф используется в качестве промежуточного представления программы компилятором для проведения внутренних оптимизирующих преобразований.

Исходный код в текстовом представлении содержит в себе всю необходимую информацию о поведении программы. Однако его анализ часто является сложной задачей, даже при том, что современные интегрированные среды разработки поддерживают интеллектуальные механизмы, значительно её упрощающие.

Для анализа бинарного кода используются специализированные программы – дизассемблеры и декомпиляторы. Анализ ассемблерного кода – сложная и трудоемкая задача, требующая от специалиста обширных знаний. В то же время декомпиляция произвольного исполняемого файла не всегда возможна, и в некоторых случаях восстановленный код более труден для восприятия, чем ассемблерный.

Альтернативным способом является анализ визуального представления потока управления программы. В настоящее время существует большой выбор универсальных систем визуальной обработки графовых моделей, таких как uDraw (daVinci) [1], VCG [2], Graphlet [3], GraVis [4], Graph Drawing Server [5], graphViz [6], VisualGraph [7]. Несмотря на то, что таких систем достаточно много, все они обладают недостатками. Например, применительно к задаче визуализации графа потоков управления подобные системы не учитывают особенности и характерные черты таких графов.

Исходя из вышесказанного, визуализация атрибутивных графовых моделей является актуальной задачей и требует отдельного рассмотрения с учётом специфики природы их возникновения.

В работе рассматривается вопрос применения методов структурного анализа [8] графа потоков управления к задаче визуализации, представляющих его графовых моделей.

1. Критерии качества визуализации графа потоков управления

Определение 1. *Раскладка графа на плоскости (или в пространстве) — это отображение вершин и ребер графа в множество точек плоскости (или пространства).*

Один и тот же граф можно визуализировать разными способами (рис. 1), причем качество одного и того же изображения может оцениваться по разному в зависимости от характера использования и вида отображаемой информации. Основным критерием оценки качества визуализации является соответствие изображения природе возникновения информации. Помимо этого для определения качества визуализации выделяют такие понятия, как изобразительное соглашение, эстетичность и ограничения [9]:

- *Изобразительное соглашение* — это одно из основных правил, которому должно удовлетворять изображение графа, чтобы быть допустимым.
- *Эстетические критерии* определяют такие свойства изображений, которые желательно применять в наибольшей степени, насколько это возможно, чтобы повысить наглядность изображения.
- *Ограничения.* Если соглашения и эстетические критерии формулируются по отношению ко всему графу и его изображению, то ограничения относятся к отдельным подграфам и частям изображений.

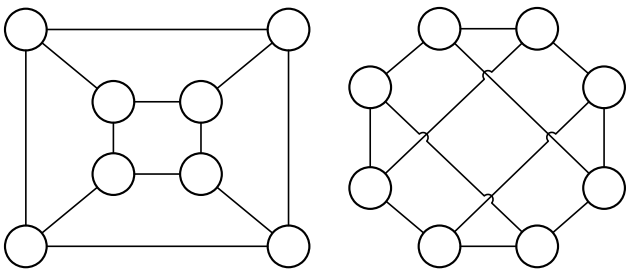


Рис. 1. Разные способы визуализации одного и того же графа

В управляющем графе, восстановленном из бинарного кода, сохраняется информация о разбиении на базовые блоки, начальной вершине (входе) и непустом множестве конечных вершин (выходах), а также информация о возможных путях передачи управления между базовыми блоками. Естественным визуальным представлением управляющего графа является его изображение в виде диаграммы потоков управления (блок схемы). Исходя из данного соображения определим следующие изобразительные соглашения для визуализации узлов такого графа (рис. 2):

а) **Блок действия.** Представляет собой узел, передача управления из которого осуществляется только в одном направлении.

б) **Логический блок (блок условия).** Соответствует условному оператору в высокоуровневых языках, а в графе узлу расхождения потока управления.

в) **Граница цикла.** Состоит из двух частей, обозначающих начало и конец операций, выполняемых внутри цикла.

г) **Блок начало-конец (пуск-остановка).** Отображает вход и выход в функцию (программу).

а) блок действия б) логический блок в) границы цикла г) начало-конец

Рис. 2. Типы элементов.

Основной задачей данной работы является разработка алгоритма визуализации управляющего графа с критериями визуализации, принятыми для изображения блок схем.

2. Обзор методов визуализации графов потоков управления

Алгоритмы визуализации графов разрабатываются с начала 60-х годов [10]. Классическими в данной области считаются работы Eades P. [11], Kamada T. и Kawai S. [12] В данных работах рассматриваются универсальные алгоритмы визуализации графов. Так как управляющий граф является направленным, рассмотрим способы визуализации такого класса графов.

Метод поуровнего изображения направленных графов предложенный в работе [13] является основным для изображения данного класса графов. Как правило этот метод состоит из 4 шагов:

1. **Распределение вершин по уровням.** Каждой вершине присваивается ее ранг. При этом все дуги могут следовать только от меньшего ранга к большему. Между вершинами одного ранга не может быть дуг. Для распределения рангов вершин могут использоваться различные методы, в простейшем случае в качестве ранга может быть использована длина пути при обходе в глубину.

2. **Определение порядка вершин на уровнях.** Вершины упорядочиваются внутри уровня таких образом, чтобы минимизировать количества пересечений дуг. Самым распространенным способом решения этой задачи является «метод медиан» [14].

3. **Определение координат вершин на уровне.** На каждой уровне каждой вершине присваиваются координаты таким образом, чтобы граф соответствовал определённым для него эстетическим критериям.

4. **Проведение дуг.** Обычно, эту задачу выделяют в отдельный этап, только в том случае, если дуги изображаются не в виде прямых.

3. Граф потоков управления

Определение 3.1. *Ориентированный граф $G(X, U)$ называется графом потоков управления, если выполняются следующие условия:*

1) *граф G не содержит параллельных дуг;*
2) *в множестве вершин графа выделена одна вершина start, которая является входом графа;*
3) *в множестве вершин графа выделена одна вершина end, которая является выходом графа;*
4) *каждая вершина $x \in X$ достижима из start;*
5) *из каждой вершины $x \in X$ достижима вершина end;*

Случай, когда в управляющем графе имеется более чем одна выходная вершина, легко сводится к случаю одного выхода добавлением дополнительной фиктивной вершины соединенной со всеми фактическими выходами.

Приведём основные определения используемых далее понятий, взятые из [15].

Определение 3.2. *Узел x является доминатором* y *(x dom* y*) в направленном графе, если любой путь от start до y включает узел x.*

Определение 3.3. *Узел x является постдоминатором y (x pdom y), если любой путь от y до end включает x.*

Определение 3.4. *Узел x является непосредственным доминатором y (x idom y), если x dom y, и не существует такого промежуточного узла P, что x dom P и P dom y.*

Определение 3.5. *Узел x является непосредственным постдоминатором y (x pidom y), если x pdom y, и не существует такого промежуточного узла P, что x pdom P и P pdom y.*

Определение 3.6. *Пара дуг (a, b) различных узлов a и b управляющего графа G образует SESE-регион (Single Enter Single Exit), если*

1) *a dom b;*
2) *b pdom a;*
3) *любой цикл в управляющем графе, содержащий a, содержит также и b, и наоборот;*

В работе [17] отмечено, что два любых SESE-региона в управляющем графе должны быть, либо вложенными друг в друга, либо непересекающимися. Структурный анализ на основе SESE-регионов и PST (Program Structure Tree) обычно используется для эффективного построения промежуточного представления в SSA (Static Single Assignment) форме, а также в анализе потоков данных. Для решения этих задач существенным является требование, что SESE-регион образует именно пара дуг, т. е. узел схождения имеет только одну входящую дугу, а узел расхождения – только одну исходящую дугу.

Определение 3.7. *TT-регион (Two Terminal Region) — это подграф в управляющем графе, который имеет один вход и один выход.*

Другими словами TT-регион образуют узлы, которые соответствуют схождению потока управления в графе и его последующему расхождению (рис. 3). Требования к TT-региону являются более слабыми, чем требования к SESE-региону. Таким образом, каждый SESE-регион является и TT-регионом, но не наоборот.

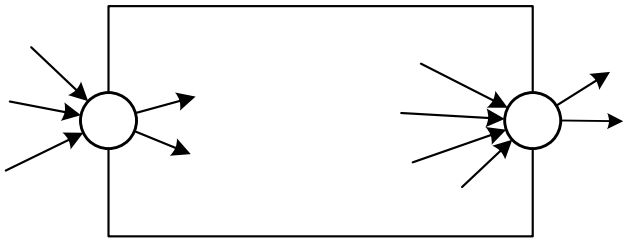


Рис. 3. TT-регион

В языках программирования, использующих идеологию структурного программирования, каждому входу TT региона в графе потоков управления соответствует оператор ветвления или цикла. Для визуализации управляющего графа предлагается построить иерархию вложенности TT регионов.

4. Визуализация графов потоков управления

Современные языки программирования в большинстве случаев поддерживаю стандартный набор высокоуровневых операторов (if-then, if-then-else, for, while и т. д.). Структурные конструкции в процессе компиляции порождают специфические только для них подграфы. Например, конструкция if-then породит фрагмент графа изображенного на рис. 4, где вместо блока then может находится другая управляющая конструкция.

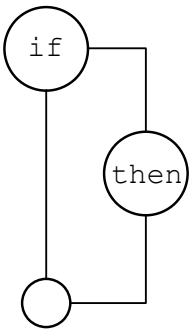


Рис. 4. Подграф для оператора if-then

Проанализировав все стандартные высокоуровневые операторы можно для каждой управляющей конструкции описать шаблон порождаемого ею подграфа. Для ограниченного количества шаблонов можно выполнить поиск в управляющем графе подграфов, соответствующих одному из шаблонов. В случае обнаружения подграфа он заменяется новым абстрактным узлом, при этом входящие и исходящие дуги перенаправляются соответствующим образом. После этого процесс поиска шаблонов повторяется

для полученного графа. Процесс поиска шаблонов может завершиться двумя способами. Во-первых, граф потоков управления может свернуться в одну абстрактную вершину, такие графы называются *сводимыми*. В противном случае может встретиться подграф, не соответствующий ни одному из заданных шаблонов. Такая область называется *неопределенной*, а весь граф — *несводимым*.

Синтаксис языков программирования, которые придерживаются идеологии структурного программирования, позволяет писать только такие программы, управляющий граф которых всегда сводим. Данное утверждение верно и для большинства других языков, до тех пор, пока явно или неявно не будет использован оператор безусловной передачи управления. Существует два подхода к решению этой проблемы:

1. Избавление от несводимых областей. Существенным недостатком такого метода является необходимость добавления или удаления вершин и дуг в графе, что является недопустимым при визуализации графовых моделей.

2. Выделение несводимой области в неопределенный регион и замена его на абстрактную вершину.

4.1. Описание алгоритма структурирования

Разработанный алгоритм основан на методе структурного анализа [16], в котором предлагается выделять в графе регионы заданного вида и заменять их абстрактным узлом, перенаправляя при этом входящие и исходящие дуги. Наложение шаблонов происходит в порядке обхода в глубину графа до тех пор, пока он не свернется в один абстрактный узел. Для графа строится дерево доминирующих вершин, которое используется для классификации дуг и выделения циклов. В данном методе качество структурирования зависит от порядка наложения шаблонов.

Ниже приведен разработанный алгоритм структурирования:

Исходные параметры: G, D, P
Результат: Абстрактный узел, содержащий в себе иерархию вложенных регионов
для каждого v *из* D *в обратном порядке* **выполнять**
  для каждого $p \in$ дети(v) **выполнять**
    если p *pidom* v **тогда**
      $S \leftarrow$ дети$(v) \setminus p$
      если КлассифицироватьРегион$(S) \neq$ *неопределенный* **тогда**
        НаложитьШаблон(S)
      конец условия
      иначе
        ИерархическийРаскладчик$(S \cup p)$
        ВыделитьНеопределенныйРегион(S)
      конец условия
      Модифицировать(G, D, P)
    конец условия
  конец цикла
конец цикла

Алгоритм 1: Алгоритм структурирования

Построим дерево доминиаторов D и постдоминаторов P для графа $G(E, V)$. Для программы, управляющий граф которых не имеет выделенной конечной вершины классифицируем дуги как обратные, прямые и косые. Без учета обратных дуг в графе обязательно найдется хотя бы одна вершина, которая не имеет выходных дуг. Если такая вершина одна, то она является конечной, в противном случае введем фиктивную вершину перенаправив дуги из всех вершин v графа G, для которых выполняется условие $out(v) = 0$. Получившийся управляющий граф является правильным, так как из любой вершины достижима конечная.

Обход дерева доминаторов совершается снизу вверх, в порядке обратном обходу в ширину. Для этого удобно использовать структуру данных «стек», помещая в неё узлы графа при обходе в ширину. При таком обходе ни один из узлов p не будет иметь собственных детей (рассматривается поддерево глубиной не больше 1). Таким образом, на каждой итерации алгоритма выделяется TT-регион, имеющий наибольший уровень вложенности. Для этого используется правило p *pidom* v. Постдоминатор вершины v из S является терминальной вершиной, на которой сходится поток управления, прошедший через v. Эту вершину нельзя включать в регион потому, что в управляющем графе она может является терминальной для другого региона.

После того, как выделено множество вершин TT-региона, он классифицируется. На выделенный подграф последовательно накладываются шаблоны, показанные на рис. 5. Если регион соответствует одному из этих шаблонов, то он считается *определённым*, иначе *неопределенным*.

Для определенных регионов создается новый абстрактный узел, включающий в себя структуру содержащегося в нём подграфа и ограничивающий прямоугольник этого подграфа, вычисленный по правилам, заданным в шаблоне. К неопределенным регионам применяется иерархический раскладчик, и только после этого по известным координатам узлов вычисляется его ограничивающий прямоугольник. Неопределенный регион также заменяется новым абстрактным узлом.

В конечном счёте останется один абстрактный регион, не имеющий входящих и исходящих дуг, который содержит в себе всю иерархию вложенности подпрограммы. Для этого региона вычисляется ограничивающая область, в которой гарантированно могут расположиться все узлы графа.

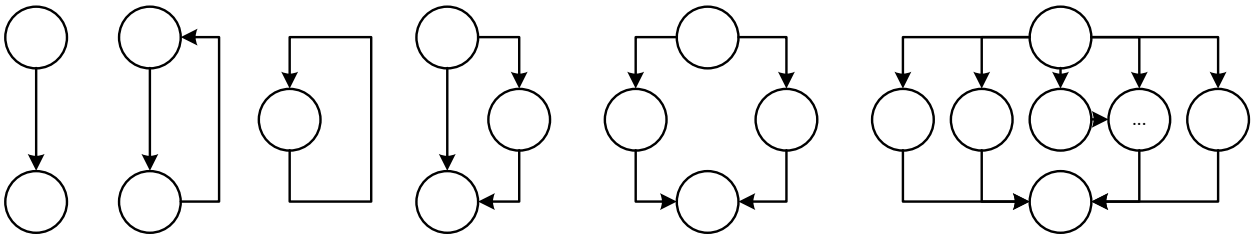


Рис. 5. Выделяемые шаблоны

4.2. Процесс раскладки

Процесс раскладки происходит рекурсивно сверху вниз, начиная с региона верхнего уровня. Для этого региона задаются начальные координаты. Далее, если у региона есть шаблон, применяются правила отображения, определенные в нем. Приведем пример для шаблона `if-then-else` (рис. 6).

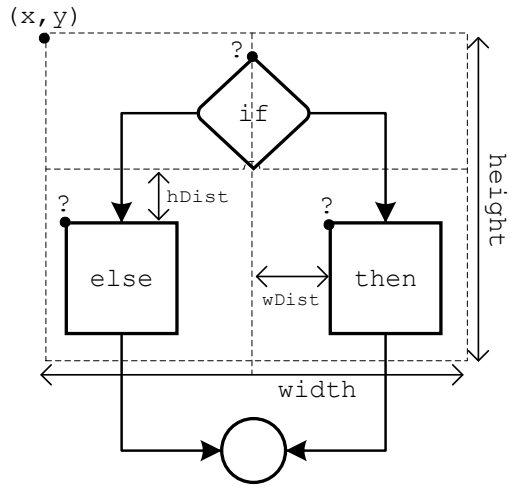


Рис. 6. Шаблон отображения if-then-else

Определим правила вычисления координат для узлов `if, then, else`:

- `if.x = x + abs(width - if.width) / 2;`
- `then.x = x  + wDist / 2, then.y = y + if.height + hDist;`
- `else.x = x - else.width - wDist / 2, else.y = y + if.height + hDist;`

Где wDist — вертикальное расстояние между узлами, hDist — горизонтальное. Данные параметры задаются вручную исходя из эстетических соображений.

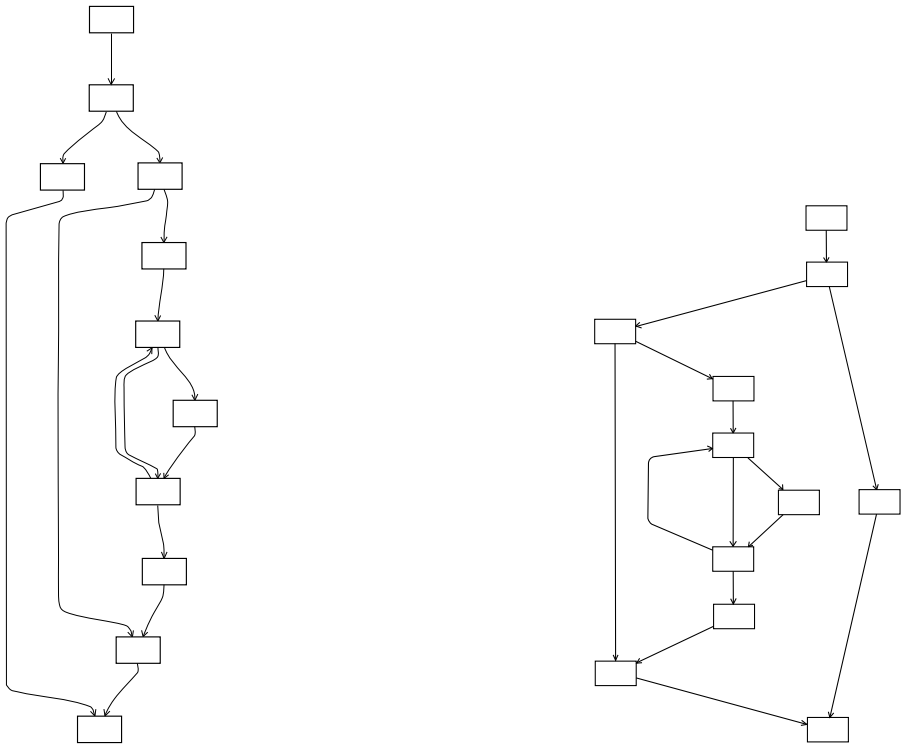
Для каждого шаблона аналогичным образом определены правила визуализации. Для неопределенных регионов используется иерархический раскладчик. Таким образом процесс раскладки сводится к последовательному применению правил отображения для вложенных регионов.

5. Реализация

Алгоритм реализован на объектно-ориентированном языке Java. На этом языке разработана библиотека JGraphX и система визуализации иерархических графовых моделей VisualGraph, которая её использует. В JGraphX поддержаны популярные алгоритмы визуализации графов. Библиотека предоставляет возможности визуализации узлов и дуг графа, позволяет задавать форму и цвет узла, а так же позволяет рисовать дуги с помощью задания узлов и угловых точек.

В текущей реализации частично поддерживаются изобразительные соглашения определенные для узлов в разделе 1. Для классификации узлов в регионах ветвления необходимо чтобы граф был атрибутивным и содержал в себе ассемблерный код базовых блоков (линейных участков кода). Также для этого необходимо проанализировать семантику программы, представленной в виде управляющего графа. С другой стороны анализ на основе дерева доминирующих вершин позволяет выделять циклы в графе, что позволяет специальным образом изображать обратные дуги.

Пример работы алгоритма и сравнение его с результатом раскладки, полученным при помощи иерархического раскладчика приведены на рис. 7: а) иерархический, б) структурный.



а) Иерархический раскладчик б) Структурный раскладчик

Рис. 7. Результат раскладки управляющего графа.

Заключение

Предложен новый подход к раскладке графов потоков управления на плоскости с использованием методов структурного анализа. На основе разработанных методов реализован структурный раскладчик атрибутивных графов потоков управления. Произведено тестирование структурного раскладчика на тестах SPEC CPU2000 [1]:

- 197.parser[2]
- 252.eon[3]

В результате около 70% графов удалось структурировать полностью (не содержат «неопределенных» регионов). Около 96% всех выделенных регионов являются структурными.

Основными преимуществами данного раскладчика являются:

- Простота изменения правил визуализации графа посредством задания правил отображения шаблонов.
- Однообразное отображение подграфов, соответствующих одним и тем же операторам в высокоуровневых языках программирования.

[1] https://www.spec.org/cpu2000/ — Standard Performance Evaluation Corporation. SPEC CPU2000 — тесты вычислительной производительности (ранее использовался пакет тестов CPU95) центрального процессора. В основном используются для тестирования компиялторов.

[2] Синтаксический разбор для естественного языка

[3] Трассировка лучей

- Возможность выделять специальным образом узлы и дуги графа, используя семантику, полученную в процессе структурирования графа.

Список литературы / References

[1] Frohlich M., Werner M. Demonstration of the Interactive Graph - Visualization Sytem daVinci // LNCS 894. 1995. P .266–269.

[2] Sander G. Graph loyout through the VCG tool // LNCS 894. 1995. P .194–205.

[3] Himsolt M. Graphlet system (system demonstration) // LNCS 1990. 1996. P .233–240.

[4] Lauer H., Ettrich M., Soukup K. GraVis system demonstration // LNCS 1353. 1997. P .344–349.

[5] Bridgeman S., Garg A., Tamassia R. A graph drawing and translation service on the WWW // LNCS 1190. 1996. P . 45–52.

[6] Gasner E. R., North S. C. An open graph visualization system and its applications to software engineering. // http://www.graphviz.org/Documentation/GN99.pdf

[7] Золотухин Т. А. Визуализация графов при помощи программного средства Visual Graph //Информатика в науке и образовании. – 2012. – С. 135-148.

[8] Михайлов А. А. Анализ графа потоков управления в задаче декомпиляции подпрограмм объектных файлов dcuil // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2014. Т. 12, вып. 2. С. 74–79.

[9] Касьянов В.Н. Визуализация информации на основе графовых моделей // Вычисл. технологии. 1999. Т. 11, № 11. С. 1123–1135.

[10] Tutte W. T. How to draw a graph // Proc. London Math. Society. 1963. 13(52). P. 743–768.

[11] Eades P. A heuristic for graph drawing // Congressus Numerantium. 1984. 42. P. 149–160.

[12] Kamada T., Kawai S. An algorithm for drawing general undirected graphs // Inform. Process. Lett. 1989. 31. P. 7–15.

[13] Eades P., Xuemin L. How to draw a directed graph // Visual Languages, 1989., IEEE Workshop on. – IEEE, 1989. – С. 13-17.

[14] P. Eades and N. C. Wormald The Median Heuristic for Drawing 2-Layered Networks // Technical Report 69, Dept. of Computer Science, University of Queensland. 1986.

[15] Aho A. V. Compilers: Principles, Techniques, And Tools Author: Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Publisher: Addison Wesle. – 1986. MLA

[16] Cifuentes C. Structuring decompiled graphs //Compiler Construction. – Springer Berlin Heidelberg, 1996. – С. 91-105.

[17] Johnson, R., Pearson, D., and Pingali, K. Finding regions fast: Single entry single exit and control regions in linear time. Tech. rep., Cornell University, Ithaca, NY, 1993