Эксперименты с интерактивным специализатором подмножества языка Java, реализующим метод частичных вычислений

И.А. Адамович <i.a.adamovich@gmail.com>[1]

1 Институт программных систем им. А.К. Айламазяна РАН, 152021, Россия, Ярославская обл., с. Веськово, ул. Петра Первого, д. 4а.

Аннотация. Методы метавычислений развиваются больше 30 лет, но так и не получили широкого распространения. В работе обсуждаются возможные причины этого явления, на основе которых делается вывод о необходимости погружения специализатора в среду разработки с удобным графическим интерфейсом. Такой специализатор подходит для реализации человеко-машинного диалога, организованного таким образом, чтобы пользователь понимал, что происходит в специализаторе, получал ценную и интересную информацию о коде, был способен корректировать исходный код для лучшей специализации и управлял специализатором.

Далее в работе рассматривается реализация интерактивного специализатора на основе частичных вычислений для подмножества языка Java. Этот специализатор погружён в популярною среду разработки (IDE) Eclipse. Впервые на основе графического интерфейса были реализованы диалог программиста с подсистемой специализации, интерактивные средства для составления задания на специализацию и управление процессом специализации.

Приводится пример успешного применения разработанного специализатора к программе на языке Java, вычисляющей степенную функцию (так называемый «пример Ершова»). На основе этого примера демонстрируется преимущества интерактивного специализатора над классическим подходом на основе командной строки.

Во многих случаях, остаточные программы, полученные в результате специализации, работают в несколько раз быстрее, чем исходные.

Ключевые слова: анализ программ; преобразование программ; интерактивная специализация программ; частичные вычисления; объектно-ориентированный язык; среда разработки программ.

Experiments with the interactive specializer that implements the partial evaluation method for a subset of the Java language

I.A. Adamovich <i.a.adamovich@gmail.com>[1]

1 Ailamazyan Program Systems Institute of the Russian Academy of Sciences
4a Peter the First str., Veskovo, Yaroslavl region, 152021 Russia

Abstract. Metacomputation methods have been developing for more than 30 years but have not been widely used. The paper discusses the possible causes of this phenomenon. On that basis the conclusion about the need to embed the specializer in the development environment with a user-friendly graphical interface is made. This specializer should be suitable for implementation of human-machine dialogue, organized in such a way that the user understands what is happening in the specializer, received valuable and interesting information about the code, was able to correct the source code for better specialization and manage the specialization process.

The paper discusses the implementation of an interactive specializer based on partial evaluation for a subset of the Java language. This specializer is immersed in the popular Eclipse development environment (IDE). For the first time on the basis of the graphical interface, the programmer's dialogue with the specialization sub-system, interactive tools for compiling a task for specialization and managing the process of specialization were implemented.

An example of the successful application of the developed specializer to a program in the Java language that calculates the power function (the so-called "Ershov example") is given. Based on this example, the advantages of an interactive specializer over the classic command-line approach are demonstrated.

In many cases, the residual programs (programs that obtained as a result of specialization) work several times faster than the original ones.

Keywords: program analysis; program transformation; interactive program specialization; partial evaluation; object-oriented language; integrated development environment.

1. Введение

Специализация — это оптимизация программ на основе использования наперёд заданной информации о значении части переменных. Пусть дана программа f(x,y) от двух аргументов x и y и значение одного из ее аргументов x=a. Результатом специализации программы f(x,y) по известному аргументу x=a называется новая программа от одного аргумента g(y), обладающая следующим свойством: f(a,y)=g(y) для любого y.

Частичные вычисления (Partial Evaluation) — это метод специализации, который заключается в получении более эффективного кода на основе однократного выполнения той части кода, которая зависит только от известной

части аргументов (и не зависит от неизвестной части). В процессе частичных вычислений операции над известными данными исполняются, а над неизвестными — переносятся в остаточную программу. Остаточная программа зависит только от неизвестной (на стадии специализации) части аргументов. Цель частичных вычислений — генерация остаточной программы более эффективной, чем исходная. Источник оптимизации — использование значения a.

Частичные вычисления развиваются более 30 лет. Исследователями получены выдающиеся результаты [1], [2] по реализации самоприменимого специализатора, воплотившего в жизнь первую, вторую и третью проекции Футамуры [3]–[5]. Первый этап исследований был завершён в начале 1990-х когда была опубликована одна из главных книг по частичным вычислениям [2]. Много задач программирования были решены специализацией (наиболее известной из которых является задача генерации компилятора по интерпретатору в соответствии со второй проекцией Футамуры) и ожидалось появление новых средств разработки на основе специализации. Однако удивительно, что спустя три десятилетия эти многообещающие методы не получили широкого распространения. Возникает вопрос: в чём же причина?

Наша гипотеза в том, что главное ожидание, связанное с развитием специализаторов, было ошибочным. Разработчики этих методов надеялись, что специализаторы могут работать в полностью автоматическом режиме, и разработчикам просто нужно было придумать некоторое количество улучшений, которые решат эту задачу, после чего «великая цель» будет достигнута, и счастливые программисты начнут использовать новые инструменты. Однако этого не случилось. Временная и пространственная сложность программных преобразований, необходимых для специализации, оказались намного выше сложности оптимизации программ, которые могут использоваться как «черные ящики» с коротким и прогнозируемым временем выполнения и потребляемой памятью. По всей вероятности, автоматические методы оптимизации программ достигли пределов. Чтобы разработать и использовать более мощные инструменты, мы должны отказаться от ожиданий, что системы анализа и преобразования программ будут работать в автоматическом режиме без вмешательства человека.

Задача специализации программ обладают слишком многими степенями свободы, поэтому она не может быть решена с помощью автоматических алгоритмов и, следовательно, человеческая помощь является важным компонентом при специализации.

Исходя из этого наблюдения, мы поставили цель реализовать интерактивный специализатор, погружённый в привычную интегрированную среду разработки (IDE), например Eclipse [8]. Eclipse предоставляет разработчику специализатора набор инструментов с открытым исходным кодом, называемый Java development tools (JDT) [9]. Этот набор инструментов позволяет разработчику сфокусировать усилия на важных задачах анализа,

визуализации и трансформации Java-кода. Помимо самого специализатора, будут разработаны диалоговые инструменты для управления специализатором, взаимодействия с результатом специализации.

Мы хотели бы подчеркнуть, что существует строгое разделение задач между машиной и человеком: специализатор гарантирует функциональную эквивалентность преобразования программы, а пользователь несет ответственность за контроль специализатора таким образом, чтобы он выдавал код, удовлетворяющий целям и потребностям пользователя (которые машина не знает).

Мы считаем, что частичные вычисления лучше подходят, чем другие методы специализации (например, суперкомпиляция) для реализации человеко-машинного диалога, организованного таким образом, чтобы пользователь понимал, что происходит в специализаторе, получал ценную и интересную информацию о коде, был способен корректировать исходный код для лучшей специализации и управлял специализатором.

Причина в том, что метод частичных вычислений состоит из двух этапов:

- *анализ времени связывания* (BTA), который помечает части исходного кода, которые должны быть вычислены во время специализации, и
- *генерация остаточной программы* (RPG), которая, следуя информации, предоставленной BTA, выполняет специализацию и создает результирующий код (называемый *остаточным*).

Приятной особенностью BTA является то, что его результат (так называемая *BT аннотация*) может быть естественно показан в исходном коде при помощи подсветки, и благодаря такой визуализации остаточный код интуитивно предсказуем.

Мы надеемся, что это позволит рядовым программистам легко принять специализатор в качестве нового инструмента программирования.

Терминологическое примечание. В теории частичных вычислений части кода, подлежащие вычислению во время специализации, называются *статическими*. Остальной исходный код, который перемещается в остаточную программу, называется *динамическим*. Термин статический конфликтует с понятием статический в языке Java, а термин динамический может быть перепутан с понятием времени исполнения. Поэтому мы избегаем использования этих слов в смысле частичных вычислений и вместо этого будем использовать сокращения S и D, например S-аннотация, D-аннотация, S-код, D-код, S-часть и D-часть программы.

2. Пример специализации программы на языке Java

Рисунки 1 и 2 содержат скриншоты исходного и остаточного кода примера Ершова, сделанные в работающем в Eclipse IDE специализаторе. Метод **powXtoN** реализует возведение в степень n (второй аргумент этого метода) вещественного числа x (первый аргумент этого метода). Метод **run**

вызывает метод **powXtoN** со вторым константным аргументом равным 20. Java аннотация

```
public class ErshovExample {

    public final static double powXtoN(double x, int n) {
        double r = 1;

        while(n != 0)
            if (n%2 == 0) {
                x = x * x; n = n / 2;
            }
            else {
                r = r * x; n = n - 1;
            }

        return r;
    }

    @Specialize
    public static void run(double x, int n) {
        powXtoN(x, 20);
    }
}
```

Рис. 1. Размеченная программа, возводящая число в степень

@Specialize перед определением метода **run** означает, что он должен быть проспециализирован и его тело будет заменено остаточным кодом, также будут сгенерированы специализированные версии вызываемых из **run** методов.

Имена методов **run** и **powXtoN** в первых строке своих определений подкрашены оранжевым, чтобы показать, что они участвуют в BTA. Тела этих методов анализируются и аннотируются: зелёная подсветка отмечает S-части кода (при монохромной печати вы будете видеть серую подсветку).

2.1. Анализ времени связывания

Алгоритм BTA для переменных и операций над примитивными типами довольно прост.

Во-первых, все константы аннотируются как S. Затем итеративно: подвыражения, содержащие только S-части, становятся S; локальные объявления и присваивания переменных с правыми сторонами уже аннотированными S становятся S; параметры метода, соответствующие S аргументам во всех точках вызова, становятся S; в случае конфликта нескольких вызовов одного и того же метода конфликтный параметр становится D; конфликт с несколькими присваиваниями локальной переменной влечёт аннотирование этой переменной как D; if оператор с S условным выражением аннотируется как S независимо от аннотации его ветвей (это означает, что if-else исчезнет, тогда как одна из ветвей перейдёт в остаточный код); другие инструкции управления анализируются и аннотируются аналогичным образом.

```
public class ErshovExample {
    public static void run(double x, int n) {
        powXtoN_20(x);
    }

    public final static double powXtoN_20(double x) {
        double r = 1;
        {
            {
                x = x * x;
            }
            {
                x = x * x;
            }
            {
                r = r * x;
            }
            {
                x = x * x;
            }
            {
                x = x * x;
            }
            {
                r = r * x;
            }
        }
        return r;
    }
}
```

Рис. 2. Остаточная программа

Когда ни одно из только что описанных правил не может быть применено, остальные части кода аннотируются как D. D-части программы никак не выделены на рисунке 1.

Этот режим работы BTA, когда каждый фрагмент кода получает однозначную аннотацию S или D называется *моновариантным*. Более общий подход, когда возможны несколько версий аннотации для одного и того же участка кода, называется *поливариантным*. Текущая версия BTA моновариантна. В будущем мы планируем реализовать поливариантный BTA для классов и ссылочных типов в соответствии с теорией, разработанной в работах [8]–[16].

Моновариантный BTA на примитивных типах может быть формально определен как абстрактная интерпретация на решетке с тремя элементами: undefined < S < D.

2.2. Генерация остаточной программы

На стадии генерации специализатор, работающий в соответствии с принципами частичных вычислений, начинает работу с метода, помеченного аннотацией @Specialize, и рекурсивно посещает все вызванные методы. Обратите внимание, что поскольку все инструкции и методы с побочными эффектами помечены как D и, следовательно, должны быть перенесены в

остаточную программу, а не выполнены во время специализации, то порядок специализации методов не имеет значения.

Для каждого из специализированных методов может быть создано несколько остаточных версий — по одному для каждой комбинации значений S-аргументов. У специализированной версии метода останутся только те параметры, которые соответствуют D-параметрам в исходной программе.

Текущая версия специализатора может зациклиться если будет сгенерировано бесконечно много значений S-аргументов. Когда специализатор будет доведен до “промышленной” версии, то будут разработаны специальные средства отладки, чтобы избежать таких ситуаций. Это одна из задач для нашей будущей работы.

3. Обзор области и сравнение

Много работ посвящено частичным вычислениям для функциональных языков. Книга [2] подводит итог первой волны развития этого метода.

В начале 1990-х годов был разработан первый (насколько нам известно) специализатор языка C, названный C-MIX [17], [18]. В главе 11 книги [2] содержится подробное изложение. C-MIX специализирует программу в три этапа. Первый этап — анализ ссылок. Для каждой ссылочной переменной строится набор переменных, к которым она может обращаться. Если анализ обнаруживает, что несколько ссылок на переменные могут ссылаться на одну и ту же память, они помечаются одинаково. Второй этап - построение аннотации времени связывания для исходного кода. Ссылки на одну и ту же область памяти аннотируются одинаково. В случае конфликтов аннотация сводится к D, как обычно. Третий этап - генерация остаточной программы.

Специализация ссылочных типов в Java может быть аналогична анализу указателей в C-MIX. Однако более строгая типизация Java может быть использована для более глубокой специализации. Текущая версия нашего специализатора аннотирует все ссылочные переменные как D и, следовательно, они остаются неизменными.

В будущем мы планируем добавить анализ времени связывания для ссылочных типов. В отличие от C-MIX, мы ожидаем, что наш специализатор будет продолжать работать в два этапа — без этапа анализа ссылок.

Дальнейшая разработка идей C-MIX привела к созданию специализатора программ, написанных на C, называемый Tempo [19], [20]. Этот специализатор очень похож на C-MIX.

Важным шагом было создание первого специализатора для объектно-ориентированного языка — JSpec для Java [21]. JSpec использует компилятор Harissa [22] для трансляции программы с Java на C. Затем упомянутый выше спицализатор Tempo преобразует программу. Полученное C-представление специализированной Java-программы отображается обратно в Java с использованием транслятора Assirah [21]. И наконец, инструмент AspectJ

соединяет специализированную программу с исходной для получения исполняемого байт-кода Java.

Наиболее развитой (насколько нам известно) метод частичных вычислений для объектно-ориентированных языков, таких как C# и Java, был реализован в специализаторе CILPE [8]–[16] для языка Common Intermediate Language (CIL), байкода Microsoft .NET Framework. Этот специализатор поддерживает почти все базовые конструкции объектно-ориентированных языков, таких как C# и Java. В CILPE была введена новая концепция кучи времени связывания (BT куча). BT куча представляет собой абстрактное описание состояния кучи времени выполнения, которое позволяет разделить ссылочные данные на вычислимые во время специализации и те, которые следует переместить в остаточную программу. в результате специализации некоторые объекты больше не создаются в остаточной программе, и при необходимости вместо полей объектов используются локальные переменные. Реализуя классы и объекты, мы будем развивать наш специализатор в соответствии с исследованием, проведенным в этой работе.

Мы знаем только об одной работе по частичным вычислениям, выполненной в практической области — набор инструментов GraalVM в Oracle Labs [23], [24]. Инструментарий предназначен для создания предметно-ориентированных языков (DSL) с помощью интерпретаторов, получая при этом высокую производительность из-за использования специализации. Первая проекция Футамуры обеспечивает такую возможность [3], [4] и [глава 1.5.1 книги 2]: зная программу и интерпретатор, исполняющий эту программу, GraalVM специализирует этот интерпретатор по части данной программы и создает машинный код этой части. Полученный код выполняется намного быстрее, чем исходный в интерпретаторе. Основная цель GraalVM — предоставить технологию подобную JIT-компиляции для разработчика языка программирования, без необходимости реализации разработчиком сложного механизма JIT. Специализация интерпретатора в GraalVM не является автоматической и использует подсказки разработчика интерпретатора. Именно в этом направлении мы и развиваем наше исследование: разрабатываются механизмы предоставления программисту, во-первых, существенной информации о специализируемой программе и, во-вторых, рычагов управления специализацией.

4. Заключение

В данной статье был описан интерактивный специализатор, преобразующий программы на языке Java. Принципы работы и возможности интерактивного специализатора объясняются на основе примера исходной программы и результата её специализации. Показано, что использование интерактивной специализации позволяет преодолеть некоторые трудности, с которыми сталкиваются полностью автоматические специализаторы.

Мы видим следующие возможности для развития текущей реализации специализатора:

- развить процессы анализа и оптимизации программы в части классов и объектов;
- реализовать диалоговые средства составления задания на специализацию и управлением процессом анализа времени связывания (BTA), генерацией остаточной программы;
- добавить средства анализа и сопоставления остаточной программы с исходной;
- добавить средства преобразования программ, хорошо структурированных в понятиях высокого уровня, которые не поддаются автоматическому распараллеливанию, в код, который может быть распараллелен;
- выполнить подготовка демонстрационных задач, которые получают преимущества от специализации, например, построение компилятора программы по интерпретатору;
- совершить переход от моновариантной разметки к поливариантной;

Специализатор можно скачать по ссылке:
ftp://ftp.botik.ru/rented/iaadamovich/Specializer .

Мы благодарны нашим друзьям и коллегам Юрию и Андрею Климовым за ценные советы по методам специализации в целом и частичным вычислениям в частности, за конструктивную обратную связь по дизайну нашего специализатора.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 18-37-00454.

Литература

1. Jones N.D., Sestoft P. and Søndergaard H. An experiment in partial evaluation: the generation of a compiler generator // Rewriting Techniques and Applications, Lecture Notes in Computer Science, J.-P. Jouannaud, (Ed.), vol. 202. Springer-Verlag, 1985, pp. 124–140.
2. Jones N.D., Gomard C.K., and Sestoft P. Partial Evaluation and Automatic Program Generation // Prentice-Hall, 1993. URL: http://www.itu.dk/~sestoft/pebook/pebook.html, дата обращения: 20.06.2018.
3. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler // Systems, Computers, Controls, vol. 2, no. 5, pp. 45–50, 1971.
4. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler // Higher-Order and Symbolic Computation, vol. 12, no. 4, pp. 381–391, Dec 1999. Updated and revised version of [3].

5. Futamura Y. EL1 Partial Evaluator (Progress Report) // Center for Research in Computing Technology, Harvard University, Tech. Rep., 1973. URL: http://fi.ftmr.info/PE-Museum/EL1.PDF, дата обращения: 20.06.2018.
6. Eclipse Foundation. Eclipse integrated develoment environment (IDE). URL: https://www.eclipse.org.
7. Eclipse Foundation. Eclipse Java development tools (JDT). URL: https://www.eclipse.org/jdt.
8. Klimov Yu.A. An approach to polyvariant binding time analysis for a stack-based language // First International Workshop on Metacomputation in Russia, Proceedings. Pereslavl-Zalessky, Russia, July 2–5, 2008. Pereslavl-Zalessky: Ailamazyan University of Pereslavl, 2008, pp. 78–84. URL: http://meta2008.pereslavl.ru/accepted-papers/paper-info-6.html, дата обращения: 20.06.2018.
9. Климов Ю.А. Особенности применения метода частичных вычислений к специализации программ на объектно-ориентированных языках // Препринты ИПМ им. М.В. Келдыша. 2008. № 12. URL: http://library.keldysh.ru/preprint.asp?id=2008-12, дата обращения: 20.06.2018.
10. Климов Ю.А. Возможности специализатора CILPE и примеры его применения к программам на объектно-ориентированных языках // Препринты ИПМ им. М.В. Келдыша. 2008. № 30. URL: http://library.keldysh.ru/preprint.asp?id=2008-30, дата обращения: 20.06.2018.
11. Климов Ю.А. SOOL: объектно-ориентированный стековый язык для формального описания и реализации методов специализации программ // Препринты ИПМ им. М.В. Келдыша. 2008. № 44. URL: http://library.keldysh.ru/preprint.asp?id=2008-44, дата обращения: 20.06.2018.
12. Климов Ю.А. Специализатор CILPE: анализ времен связывания // Препринты ИПМ им. М.В. Келдыша. 2009. № 7. URL: http://library.keldysh.ru/preprint.asp?id=2009-07, дата обращения: 20.06.2018.
13. Климов Ю.А. Специализатор CILPE: генерация остаточной программы // Препринты ИПМ им. М.В. Келдыша. 2009. № 8. URL: http://library.keldysh.ru/preprint.asp?id=2009-08, дата обращения: 20.06.2018.
14. Климов Ю.А. Специализатор CILPE: доказательство корректности. // Препринты ИПМ им. М.В. Келдыша. 2009. № 33. 2009. URL: http://library.keldysh.ru/preprint.asp?id=2009-33, дата обращения: 20.06.2018.
15. Климов Ю.А. Специализация программ на объектно-ориентированных языках методом частичных вычислений // Дис. к.ф.-м.н., Институт прикладной математики им. М.В. Келдыша РАН, Москва, Россия, ноябрь

2009, 183 стр. URL: http://pat.keldysh.ru/~yura/publications/2009.10-Klimov-Disser-Specializacia_programm_na_ob'ektno-orientirovannyx_yazykah.pdf, дата обращения: 20.06.2018.
16. Климов Ю.А. Специализатор CILPE: частичные вычисления для объектно-ориентированных языков // Программные системы теория и приложения, № 3(3), стр. 13–36, 2010 URL: http://psta.psiras.ru/read/psta2010_3_13-36.pdf, дата обращения: 20.06.2018.
17. Andersen L.O. Program analysis and specialization for the C programming language // Ph.D. dissertation, DIKU, University of Copenhagen, May 1994, (DIKU report 94/19).
18. Andersen L.O. Binding-time analysis and the taming of C pointers // Proceedings of the 1993 ACM SIGPLAN symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '93). ACM, 1993, pp. 47-58. URL: http://dx.doi.org/10.1145/154630.154636, дата обращения: 20.06.2018.
19. Consel C., Lawall J.L., and Meur A.-F.L. A tour of Tempo: a program specializer for the C language // Sci. Comput. Program., vol. 52, no. 1-3, pp. 341–370, 2004.
20. Meur A.L., Lawall J.L. and Consel C. Towards bridging the gap between programming languages and partial evaluation // Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM ’02), Portland, Oregon, USA, January 14-15, 2002, P. Thiemann, (Ed.). ACM, 2002, pp. 9–18. URL: http://doi.acm.org/10.1145/503032.503033, дата обращения: 20.06.2018.
21. Schultz U.P., Lawall J.L. and Consel C. Automatic program specialization for Java // ACM Trans. Program. Lang. Syst., vol. 25, no. 4, pp. 452–499, 2003.
22. Muller G., Moura B., Bellard F. and Consel C. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code // Proceedings of the Third USENIX Conference on Object-Oriented Technologies (COOTS), June 16-20, 1997, Portland, Oregon, USA, S. Vinoski, (Ed.). USENIX, 1997, pp. 1–20.
23. Würthinger T., Wimmer C., Wöß A., Stadler L., Duboscq G., Humer C., Richards G., Simon D., and Wolczko M. One VM to rule them all // Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013. New York, NY, USA: ACM, 2013, pp. 187–204. URL: http://doi.acm.org/10.1145/2509578.2509581, дата обращения: 20.06.2018.
24. Würthinger T., Wimmer C., Humer C., Wöß A., Stadler L., Seaton C., Duboscq G., Simon D., and Grimmer M. Practical partial evaluation for high-performance dynamic language runtimes // SIGPLAN Not., vol. 52, no. 6, pp. 662–676, Jun. 2017. URL: http://doi.acm.org/10.1145/3140587.3062381, дата обращения: 20.06.2018.

References

1. Jones N.D., Sestoft P. and Søndergaard H. An experiment in partial evaluation: the generation of a compiler generator // Rewriting Techniques and Applications, Lecture Notes in Computer Science, J.-P. Jouannaud, (Ed.), vol. 202. Springer-Verlag, 1985, pp. 124–140.
2. Jones N.D., Gomard C.K., and Sestoft P. Partial Evaluation and Automatic Program Generation // Prentice-Hall, 1993. URL: http://www.itu.dk/~sestoft/pebook/pebook.html, data obrashcheniia: 20.06.2018.
3. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler // Systems, Computers, Controls, vol. 2, no. 5, pp. 45–50, 1971.
4. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler // Higher-Order and Symbolic Computation, vol. 12, no. 4, pp. 381–391, Dec 1999. Updated and revised version of [3].
5. Futamura Y. EL1 Partial Evaluator (Progress Report) // Center for Research in Computing Technology, Harvard University, Tech. Rep., 1973. URL: http://fi.ftmr.info/PE-Museum/EL1.PDF, дата обращения: 20.06.2018.
6. Eclipse Foundation. Eclipse integrated develoment environment (IDE). URL: https://www.eclipse.org.
7. Eclipse Foundation. Eclipse Java development tools (JDT). URL: https://www.eclipse.org/jdt.
8. Klimov Yu.A. An approach to polyvariant binding time analysis for a stack-based language // First International Workshop on Metacomputation in Russia, Proceedings. Pereslavl-Zalessky, Russia, July 2–5, 2008. Pereslavl-Zalessky: Ailamazyan University of Pereslavl, 2008, pp. 78–84. URL: http://meta2008.pereslavl.ru/accepted-papers/paper-info-6.html, data obrashcheniia: 20.06.2018.
9. Klimov Yu.A. Osobennosti primeneniia metoda chastichnykh vychislenii k spetsializatsii programm na obieektno-orientirovannykh iazykakh // Preprinty IPM im. M.V. Keldysha. 2008. № 12. URL: http://library.keldysh.ru/preprint.asp?id=2008-12, data obrashcheniia: 20.06.2018.
10. Klimov Yu.A. Vozmozhnosti spetsializatora CILPE i primery ego primeneniia k programmam na obieektno-orientirovannykh iazykakh // Preprinty IPM im. M.V. Keldysha. 2008. № 30. URL: http://library.keldysh.ru/preprint.asp?id=2008-30, data obrashcheniia: 20.06.2018.
11. Klimov Yu.A. SOOL: obieektno-orientirovannyi stekovyi iazyk dlia formalnogo opisaniia i realizatsii metodov spetsializatsii programm // Preprinty IPM im. M.V. Keldysha. 2008. № 44. URL: http://library.keldysh.ru/preprint.asp?id=2008-44, data obrashcheniia: 20.06.2018.

12. Klimov Yu.A. Spetsializator CILPE: analiz vremen sviazyvaniia // Preprinty IPM im. M.V. Keldysha. 2009. № 7. URL: http://library.keldysh.ru/preprint.asp?id=2009-07, data obrashcheniia: 20.06.2018.
13. Klimov Yu.A. Spetsializator CILPE: generatsiia ostatochnoi programmy // Preprinty IPM im. M.V. Keldysha. 2009. № 8. URL: http://library.keldysh.ru/preprint.asp?id=2009-08, data obrashcheniia: 20.06.2018.
14. Klimov Yu.A. Spetsializator CILPE: dokazatelstvo korrektnosti. // Preprinty IPM im. M.V. Keldysha. 2009. № 33. 2009. URL: http://library.keldysh.ru/preprint.asp?id=2009-33, data obrashcheniia: 20.06.2018.
15. Klimov Yu.A. Spetsializatsiia programm na obieektno-orientirovannykh iazykakh metodom chastichnykh vychislenii // Dis. k.f.-m.n., Institut prikladnoi matematiki im. M.V. Keldysha RAN, Moskva, Rossiia, noiabr 2009, 183 str. URL: http://pat.keldysh.ru/~yura/publications/2009.10-Klimov-Disser-Specializacia_programm_na_ob'ektno-orientirovannyx_yazykah.pdf, data obrashcheniia: 20.06.2018.
16. Klimov Yu.A. Spetsializator CILPE: chastichnye vychisleniia dlia obieektno-orientirovannykh iazykov // Programmnye sistemy teoriia i prilozheniia, № 3(3), str. 13–36, 2010 URL: http://psta.psiras.ru/read/psta2010_3_13-36.pdf, data obrashcheniia: 20.06.2018.
17. Andersen L.O. Program analysis and specialization for the C programming language // Ph.D. dissertation, DIKU, University of Copenhagen, May 1994, (DIKU report 94/19).
18. Andersen L.O. Binding-time analysis and the taming of C pointers // Proceedings of the 1993 ACM SIGPLAN symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '93). ACM, 1993, pp. 47-58. URL: http://dx.doi.org/10.1145/154630.154636, data obrashcheniia: 20.06.2018.
19. Consel C., Lawall J.L., and Meur A.-F.L. A tour of Tempo: a program specializer for the C language // Sci. Comput. Program., vol. 52, no. 1-3, pp. 341–370, 2004.
20. Meur A.L., Lawall J.L. and Consel C. Towards bridging the gap between programming languages and partial evaluation // Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM ’02), Portland, Oregon, USA, January 14-15, 2002, P. Thiemann, (Ed.). ACM, 2002, pp. 9–18. URL: http://doi.acm.org/10.1145/503032.503033, дата обращения: 20.06.2018.
21. Schultz U.P., Lawall J.L. and Consel C. Automatic program specialization for Java // ACM Trans. Program. Lang. Syst., vol. 25, no. 4, pp. 452–499, 2003.
22. Muller G., Moura B., Bellard F. and Consel C. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code // Proceedings of the

Third USENIX Conference on Object-Oriented Technologies (COOTS), June 16-20, 1997, Portland, Oregon, USA, S. Vinoski, (Ed.). USENIX, 1997, pp. 1–20.
23. Würthinger T., Wimmer C., Wöß A., Stadler L., Duboscq G., Humer C., Richards G., Simon D., and Wolczko M. One VM to rule them all // Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013. New York, NY, USA: ACM, 2013, pp. 187–204. URL: http://doi.acm.org/10.1145/2509578.2509581, дата обращения: 20.06.2018.
24. Würthinger T., Wimmer C., Humer C., Wöß A., Stadler L., Seaton C., Duboscq G., Simon D., and Grimmer M. Practical partial evaluation for high-performance dynamic language runtimes // SIGPLAN Not., vol. 52, no. 6, pp. 662–676, Jun. 2017. URL: http://doi.acm.org/10.1145/3140587.3062381, дата обращения: 20.06.2018.