О суперкомпиляции
(к 80-тилетию со дня рождения В. Ф. Турчина)

Андрей П. Немытых

Институт программных систем РАН
nemytykh@math.botik.ru

Аннотация В России организационные основы направления исследований, которое теперь принято называть информатикой, заложил академик Мстислав Всеволодович Келдыш. Его выдающиеся научные результаты дополнялись незаурядными организаторскими способностями; в частности, и в деле привлечения талантливых ученых к работам в новых зарождающихся областях знаний [63]. Алексей Андреевич Ляпунов и Валентин Федорович Турчин оба начинали свой путь в кибернетике под руководством М. В. Келдыша. Для всех троих 2011 год является юбилейным: столетие со дня рождения М. В. Келдыша и А. А. Ляпунова и восьмидесятилетие со дня рождения В. Ф. Турчина.
В статье описывается история и обзор современного состояния развития идей В. Ф. Турчина в области автоматического преобразования программ, известных как суперкомпиляция, и дается анализ этих идей в общем контексте автоматической специализации программ.

1 Введение

Исследования в области построения систематических методов специализации программ по отношению к фиксированным свойствам их аргументов, композиционной структуре этих программ и заданным инвариантам были начаты в 1970-х годах российскими учёными А. П. Ершовым («смешанные вычисления»), В. Ф. Турчиным («суперкомиляция») и японским учёным Ё. Футамурой («generalized partial computation»). К настоящему времени накоплен большой материал, в основном относящийся к предметной области функциональных языков программирования.

Идеи суперкомиляции в основном изучались в предметной области функционального языка РЕФАЛ [57], хотя ряд результатов отшлифовывался на экспериментальной базе функционального языка LISP и его потомков. В настоящее время, наряду с несколькими примитивными моделями суперкомпиляторов [49], [35], [26], [28], построенными для простейших чисто теоретических языков, существует единственный экспериментальный суперкомпилятор SCP4 для реального языка программирования РЕФАЛ-5 [57], [61]. Название SCP4 было предложено В. Ф. Турчиным как отражающее историю идей суперкомпиляции.

В данной статье рассматриваются разные подходы к постановке задачи специализации как таковой, проводится обзор основных достижений полученных к данному моменту в области специализации функциональных программ, анализируются принципиальные отличия суперкомпиляции от других существующих методов, делается исторический обзор попыток построения суперкомпиляторов.

Ниже нам понадобится следующее определение:

Definition 1. *Реализацией функционального языка программирования* $\Re$ *назовём четвёрку* $\langle \mathtt{P,D,U,T} \rangle$*, где множество* $\mathtt{P}$ *называется множеством* $\Re$*-программ, множество* $\mathtt{D}$ *называется множеством* $\Re$*-данных; частично рекурсивные функции* $\mathtt{U}$*:* $\mathtt{P} \times \mathtt{D} \mapsto \mathtt{D}$ *и* $\mathtt{T} : \mathtt{P} \times \mathtt{D} \mapsto \mathbb{N}$ *называются соответственно универсальной функцией (или семантикой) и сигнализирующей функцией времени языка* $\Re$*. Здесь* $\mathbb{N}$ *– множество натуральных чисел.*

Мы также будем использовать обозначение $\mathtt{p(x)}$ как сокращение для $\mathtt{U(p,x)}$.

2 О двух постановках задачи специализации

В литературе рассматриваются две различные постановки задачи специализации как таковой. Мы сформулируем их в естественных точных терминах. Обратим внимание на то, что в постановках существенно различие между понятиями функции и частичной функции.

Пусть дана реализация функционального языка программирования $\langle \mathtt{P,D,U,T} \rangle$, где $\mathtt{D} = \bigcup_{n \in \mathbb{N}} M^n$[1] для некоторого множества M.

Задача №1. Пусть программа $\mathtt{p(x,y)} \in \mathtt{P}$ реализует *частично рекурсивную функцию* $F(x, y) : \mathtt{D} \times \mathtt{D} \mapsto \mathtt{D}$. Зафиксируем значение первого аргумента этой функции $x_0 \in \mathtt{D}$. В задаче специализации требуется построить другую программу $\mathtt{q(y)} \in \mathtt{P}$ такую, что

$$\forall \mathtt{y} \in \mathtt{D}.(\mathtt{q(y)} = \mathtt{p}(\mathtt{x}_0, \mathtt{y})) \wedge (\mathtt{T(q,y)} \leq \mathtt{T(p,x}_0, \mathtt{y})),$$

где значение $\mathtt{q(y)}$ определено тогда и только тогда, когда определено $\mathtt{p(x}_0\mathtt{,y)}$. В случае неопределённости $\mathtt{p(x}_0\mathtt{,y}_0\mathtt{)}$ и $\mathtt{q(y}_0\mathtt{)}$ их типы неопределённости (аварийная остановка или бесконечное время вычисления) должны совпадать. Таким образом, в этой задаче программа $\mathtt{q(y)}$ представляет ту же самую *частично рекурсивную функцию*, что и программа $\mathtt{p(x}_0\mathtt{,y)}$, а именно – $F(\mathtt{x}_0\mathtt{,y})$.

[1] То есть, $\mathtt{D}$ замкнуто относительно декартового произведения.

Задача №2. Пусть программа `p(x,y)` $\in$ `P` реализует *общерекурсивную функцию* $F(x,y) : X \times Y \mapsto$ `D`, где $X \subset$ `D`, $Y \subset$ `D`. Зафиксируем значение первого аргумента этой функции $x_0 \in X$. В задаче специализации требуется построить другую программу `q(y)` $\in$ `P` такую, что

$$\forall \texttt{y} \in Y.(\texttt{q(y)} = \texttt{p}(\texttt{x}_0, \texttt{y})) \wedge (\texttt{T(q,y)} \leq \texttt{T}(\texttt{p},\texttt{x}_0,\texttt{y})).$$

Таким образом, во второй задаче программа `q` представляет некоторое *продолжение общерекурсивной функции* $F(x_0, y) : Y \mapsto$ `D` по второй компоненте.

Программу `q` называют *остаточной* программой.

Содержательная сторона задачи состоит в построении оптимальной `q` *(по времени исполнения).*

Разные уточнения понятия оптимальности (сигнализирующей функциии времени `T`) определяют конкретные аппроксимации задачи специализации как таковой. Первая задача, грубо говоря, требует от остаточной программы `q` сохранения операционной семантики исходной программы `p`. Вторая задача более естественна, с точки зрения приложений: пользователя обычно не интересует каким будет операционное поведение остаточной программы на входных данных, не принадлежащих к области определения его предметной области. В то же время, условия второй задачи оставляют бóльшую свободу конкретным методам специализации, что часто позволяет строить более оптимальные остаточные программы по сравнению с методами действующими в рамках первой задачи.

Методы суперкомпиляции ориентированы на решение второй задачи.

3 Обзор результатов в области специализации программ

На пути реализации основополагающих идей, сформулированных А. П. Ершовым, В. Ф. Турчиным и Ё. Футамурой, встретились серьёзные трудности. Позже Н. Д. Джонс (Дания) предложил ослабить первоначально поставленные цели за счёт упрощения методов специализации («partial evaluation») [23], [20]. Эта упрощённая техника «частичных вычислений» наиболее разработана к данному моменту. Она решает первую задачу специализации. Пытаясь решить задачи самоприменения специализатора, также сформулированные в 70-х годах независимо вышеуказанными тремя исследователями, Н. Д. Джонс с сотрудниками сделали ещё один принципиальный шаг в сторону упрощения, но теперь уже не постановки задачи, а ограничения подходов к её решению. В университете Копенгагена (Н. Д. Джонс, П. Сестофт, Х. Сондергаад) в 1983 году удалось решить аппроксимирующие задачи самоприменения Копенгагенского частичного вычислителя `mix`. Здесь следует отметить, что всегда существует сигнализирующая функция времени `T` (см. постановки задач), делающая любую задачу специализации

программы p(x_0,y) тривиальной и бессодержательной. А именно, функция T, которая позволяет построить следующую остаточную программу:

q(y) { = p(x_0,y); }

то есть, просто повторив код исходной программы, зафиксировав в её входной точке данное значение аргумента. Первые результаты самоприменения mix, содержательно, незначительно отличались от приведённой тривиальной остаточной программы. Как пишет Н. Д. Джонс в статье 1993 года [20], длина остаточной программы, полученной в результате решения простейшей задачи самоприменения mix(mix(p_0,x,y))[2] mix по данной трёхстрочной программе p_0(x,y), была пятьсот страниц текста. Здесь значения y неизвестны обеим копиям mix; значение аргумента x известно специализируемому mix, но неизвестно специализирующему mix. Анализируя остаточную программу, копенгагенская группа предложила понятия «online» и «offline» методов специализации [23]. Ниже мы рассмотрим эти понятия. Выбор более простых «offline» методов позволил в 1984 году решить более приемлемые аппроксимирующие задачи самоприменения частичного вычислителя mix [23], [47]. Введя инструменты повышения местности («арности») специализируемых программ и их подпрограмм в рамках «offline» подхода, С. А. Романенко [43], [45], [46] удалось в 1987 году существенно улучшить временны́е и структурные характеристики остаточных программ задач самоприменения Московского частичного вычислителя unmix. Название его статьи, описывающей unmix, говорит само за себя «Генератор компиляторов, порожденный самоприменением специализатора, может иметь ясную и естественную структуру» [43].

Offline специализация разделяет в отдельные стадии-проходы анализ исходной программы p и метаинтерпретацию локальных шагов этой программы, которые могут быть выполнены без знания *конкретных* значений неизвестной части аргументов этих шагов. На вход первой стадии, которая называется связыванием по времени (BTA[3]), подаётся p и указание – какие из её аргументов будут известны на второй стадии преобразований (метаинтерпретации), а не сами значения этих аргументов, и какие неизвестны. Первые называются статическими, вторые – динамическими. На выходе у BTA размеченная программа p^{ann}, в которой каждое элементарное действие помечено как статическое, если его можно однозначно проинтерпретировать без знания *конкретных* значений динамической части входов этих действий-шагов. Аргументы каждого шага также размечаются на статические и динамические; анализ «движения» статической информации по программе p производится BTA. Задача, поставленная перед BTA как таковая, очевидно, алгоритмически неразрешима. Повсюду здесь, по умолчанию, имеется в виду некоторая аппроксимация сформулированной BTA-задачи. На вход

[2] Здесь подчёркивание означает кодировку.
[3] Binding time analysis.

второй стадии, которая собственно и называется при этом подходе «*специализацией*», подаётся результат BTA – $\mathtt{p}^{\mathtt{ann}}$ и значения её статических аргументов. «Специализация» (вторая стадия) логически проста и алгоритмизируема: все содержательные проблемы перенесены в BTA. Группа Н. Д. Джонса, как и С. А. Романенко, решила не оригинальную классическую задачу самоприменения, а задачу $\mathtt{mix}(\underline{\mathtt{mix}^{\mathtt{ann}}(\mathtt{p}_0^{\mathtt{ann}},\mathtt{x},\mathtt{y})})$[4], которая существенно проще классической: обе копии `mix` производят лишь вторую стадию преобразований, не занимаясь связыванием по времени. Позже эксперименты offline самоприменения Джонса-Романенко повторялись и уточнялись в разных направлениях рядом авторов. Здесь существенно, что входные данные (как статические, так и динамические, представленные параметрами) каждого шага программы просматриваются только один раз (одним проходом) на этапе «специализации» (второй стадии).

Online специализация производит метавычисления шагов преобразуемой программы `p` по ходу дела анализа тех или иных свойств этой программы; никак не ограничивая, вообще говоря, *a priori* себя ни в средствах, ни в количестве проходов по программе `p` (или по частям этой программы; например, – по входным данным каждого шага программы). Каждый проход даёт цикл, который в общем случае алгоритмически нераспознаваем, даже если он работает вхолостую, не производя преобразований, а только занимаясь поиском какого-то свойства и не находя его. Следовательно, этот цикл в общем случае будет присутствовать в остаточной программе, ухудшая её временну́ю сложность. При попытке решения задач самоприменения, проходы по данным, анализирующие являются ли эти данные статическими или динамическими, будут наблюдаться преобразующей копией специализатора, что существенно осложняет его логику (по сравнению с offline подходом). Алгоритмически неразрешимая и алгоритмически разрешимая части этой логики никак не разделены. Резюме: разработка методов online специализации сложнее разработки методов offline специализации. По определению, online специализация менее ограничена в методах, чем offline специализация в используемых методах, и, потому, потенциально значительно более сильная. Самым важным здесь является то, что при offline специализации происходят преобразования только исходной программы `p`; а при online специализации могут происходить (и происходят в случае суперкомпиляции) также преобразования подпрограмм, построенных самим специализатором (и потому могущие естественно содержать простейшие неэффективности), а не только подпрограмм программы `p`, написанных человеком.

Суперкомпиляция [56], [40] и «generalized partial computation» [9], [11], как наборы методов online специализации, дают намного более сильные механизмы автоматического анализа и преобразования программ, чем частичные вычисления. И, как следствие, ставят много более трудные задачи, - – чисто познавательные, алгоритмические и технологические (реализация конкретных специализаторов). Методы суперкомпиляции [54], [56], [41], [41],

[4] Здесь в задаче решённой С. А. Романенко нужно заменить оба `mix` на `unmix`.

[40], [37] и generalized partial computation [9], [10], в отличии от методов частичных вычислений: позволяют иногда понижать порядок временно́й сложности специализируемых программ; строят остаточные программы полностью на основании метаинтерпретации специализируемой программы, а не на её пошаговой чистке. Обзору идей суперкомпиляции посвящен следующий раздел 4.

По-видимому, наименее разработанными на данный момент нужно считать идеи generalized partial computation. В отличие от частичных вычислений и суперкомпиляции, подход generalized partial computation к специализации незамкнут. Например, анонсированный Ё. Футамурой в 2002 году [11] экспериментальный полуавтоматический специализатор WSDFU обращается к внешней системе доказательств теорем TPU [4] и внешней базе знаний для доказательств свойств преобразуемых программ. Особенный интерес представляют примеры специализации программ с числовыми аргументами, в которых в результате generalized partial computation понижается порядок временно́й сложности [9], [11]. В методах частичных вычислений и суперкомпиляции работа с числовыми данными развита крайне слабо; здесь основное внимание уделяется программам, преобразующим бинарные деревья (частичные вычисления) и конечные последовательности произвольных деревьев (суперкомпиляция).

Методы частичных вычислений, как наиболее простые, проработаны наиболее тщательно. Основной вклад здесь внесли Н. Д. Джонс и его школа. Как уже отмечалось выше, содержательной частью этих методов является BTA-анализ, аппроксимирующий алгоритмически неразрешимую часть задачи специализации как таковой. Проблема решаемая BTA состоит в том, чтобы как можно точнее распознать управляющие операторы специализируемой программы, которые можно вычислить во время специализации, не зная динамической части данных, и при этом как можно реже уходить в бесконечный цикл[5]. Наибольшие успехи в разработке BTA достигнуты анализом ожидаемого изменения размера программы в процессе её специализации (А. М. Бен-Амрам, Н. Д. Джонс, С. С. Ли [33]). Аналогом BTA в суперкомпиляции и generalized partial computation являются алгоритмы обобщения параметризованных конфигураций[6]; любая предварительная разметка исходной программы `p`, помогающая улучшить алгоритмы обобщения, была бы весьма полезна. Автору не известны попытки использования методов BTA в контексте суперкомпиляции. С другой стороны, разделение частичных вычислений на BTA и собственно специализацию, как и ориентация на пошаговую чистку исходной программы `p`, приводят к прямому, часто

[5] Требование обязательной остановки специализатора на всех его входных данных, обычно, немедленно ограничивает аппроксимирующую сигнализирующую функцию времени `T` до простейшей; интересных преобразований ожидать не приходится.

[6] Под конфигурацией понимается состояние машины в конкретной точке вычисления программы. Обобщение параметризованной конфигурации можно понимать как замену переменных (параметров) - в школьном смысле.

нежелательному, наследованию свойств p остаточной программой (см., например, статью Т. Могенсена [36]), которое и обходил С. А. Романенко, разрабатывая алгоритм повышения входной местности подпрограмм (см. выше в данном разделе). Исходное ограничение частичных вычислений на построение остаточных программ со свойствами, сформулированными в *Задаче №1* (см. 2), ставит непреодолимые препятствия на пути крайне желательных оптимизаций. Например, задача специализации типов, поставленная Н. Д. Джонсом в [20], может быть решена только в рамках постановки *Задачи №2*; на что указал Дж. Хюджс, предлагая методы специализации типов [19].

Говоря о частичных вычислениях, мы не может не отметить замечательную книгу Н. Д. Джонса «Computability and Complexity from a Programming Perspective» (1997, [22]), в которой он попытался осмыслить и обобщить с чисто теоретических позиций опыт, накопленный в области частичных вычислений.

4 Исторический обзор развития методов суперкомпиляции

В. Ф. Турчин рассматривал суперкомпиляцию с точки зрения приложения его *«философии метасистемных переходов»*. Данная статья не рассматривает философских построений Турчина.

Мы именует основные этапы истории развития идей суперкомпиляции, следуя терминологии В. Ф. Турчина, предложенной им в статьях [55], [56].

Первая публикация В. Ф. Турчина «Эквивалентные преобразования рекурсивных функций, описанных на языке РЕФАЛ» датируется 1972 годом [50]. Язык РЕФАЛ изначально проектировался Турчиным как метаязык, ориентированный на преобразование программ (в частности, программ на языке программирования РЕФАЛ). В этой статье описывается подмножество РЕФАЛа, названное ограниченным РЕФАЛом, в котором время отождествления входных данных функции с образцом равномерно ограничено по размеру входных данных. Ограничение накладывается на синтаксис образцов. Соответствующие ограниченные образцы названы L-выражениями. Во всех моделях супекомпиляторов[7], построенных до суперкомпилятора SCP4, объектными языками использовались только эти ограниченные образцы или их подмножества. Вводится естественное для любых метавычислений понятие прогонки (хотя, не называется) L-выражений. В. Ф. Турчин формулирует исчисление эквивалентных преобразований ограниченных РЕФАЛ программ. Идеи этого исчисления заложили основу методов суперкомпиляции.

Второй важной работой В. Ф. Турчина стал Courant Computer Science отчёт №20 (1980 г., [52]), где изложено много идей, относящихся к преобразованию программ. Многие из этих идей излагаются очень расплывчато и часто неубедительно. Работа изобилует примерами неалгоритмизированных преобразований программ, постановками интересных задач, многие из

[7] В том числе и для модельных диалектов LISPa.

которых до сих пор не решены. В примерах по существу используется ассоциативность конструктора конкатенации в РЕФАЛе. Этот отчёт ставит важные вопросы, но не отвечает на них.

SCP1. Первая простейшая модель суперкомпилятора была реализована в Нью-Йорке в 1981 году В. Ф. Турчиным, Б. Ниренбергом и Д. В. Турчиным [60] на одном из диалектов РЕФАЛа. SCP1 работал в диалоговом режиме, запрашивая у человека результат обобщения конфигураций. Таким образом, основная задача аппроксимации алгоритмически неразрешимой части логики суперкомпиляции осталась полностью за скобками рассмотрения. SCP1 представлял важный шаг в шлифовке алгоритма прогонки, которая производила метавычисления вызовов по необходимости («call by need») на этапе суперкомпиляции. Авторам удалось проспециализировать несколько простых примеров. Один из которых стал классическим: двухпроходная программа, заменяющая в строке символ `'a'` на `'b'`, – и, затем, `'b'` на `'c'`, была преобразована в однопроходную по контексту вызова этих двух проходов, представленному явной синтаксической композицией `f(g(x))`. Таким образом, SCP1 решал *Задачу №2* (см. 2) специализации. Позже соответствующую семантику стали называть «ленивой». В 1990 году Ф. Водлер назвал преобразование на основе такой прогонки «deforestation» [62] и описал *алгоритмически неполный* язык, допускающий только конечное число параметризованных конфигураций для фиксированной программы при повторении шагов ленивой прогонки в цикле.

SCP2 был разработан В. Ф. Турчиным в 1984 году. Статья В. Ф. Турчина «The concept of a supercompiler», опубликованная в 1986 г. [54] и описывающая некоторые идеи реализации суперкомпилятора SCP2, стала основным классическим трудом по методам суперкомпиляции. В языке описания параметризованных конфигураций программы появился связка-конструктор отрицания, что позволило решить методами суперкомпиляции классическую задачу преобразования наивного алгоритма `p(s,x)` поиска подстроки `s` в строке `x` в алгоритм КМР [29]; посредством специализации исходной программы по первому аргументу $\texttt{p(s}_0\texttt{,x)}$. Показана возможность использования суперкомпилятора SCP2 для автоматического доказательства простых теорем существования. Алгоритм обобщения, реализованный в SCP2, работает *ad hoc*, и для получения более-менее интересных преобразований требуется человеческое вмешательство в работу этого алгоритма.

Во второй половине 80-х годов на одном из московских РЕФАЛ семинаров А. Веденов объявил о скором завершении им реализации суперкомпилятора РЕФАЛа. Публикаций и сообщений о фактической реализации не последовало.

В 1987 году выходят два препринта Института прикладной математики им. Келдыша, опубликованные учениками Турчина, рассматривающие проблемы суперкомпиляции: «РЕФАЛ-4 – расширение РЕФАЛа-2, обеспечивающее выразимость прогонки» (С. А. Романенко, [44]) и «Метавычислитель

для языка РЕФАЛ, основные понятия и примеры» (Анд. В. Климов, С. А. Романенко, [27]).

Второй по значимости работой В. Ф. Турчина стала статья «The algorithm of generalization in the supercompiler» (1988 г., [53]), в которой описывается алгоритм обобщения стека вызовов функций и доказывается теорема об остановке этого алгоритма. Алгоритм получил название «Обнинский алгоритм разрезания стека», – в честь города, где В. Ф. Турчин впервые доложил этот алгоритм в России. Обнинский алгоритм является одним из самых важных алгоритмов суперкомпиляции. Суперкомпилятор SCP2 был расширен этим алгоритмом (см. [58]).

В 1989 году была сделана первая реальная попытка самоприменения суперкомпилятора. Базисными конфигурациями называются параметризованные обобщённые конфигурации специализируемой программы `p`, в терминах которых можно описать соответствующую `p` остаточную программу. В результате изучения трассировки циклящегося процесса самоприменения SCP2 в ручную были построены конечные множества базисных конфигураций для нескольких конкретных простых задач самоприменения. Эти множества базисных конфигураций обеспечивали конечность процесса суперкомпиляции на данных задачах без использования алгоритма обобщения. Удалив из SCP2 алгоритм обобщения и сообщая на входе суперкомпилятору SCP2 для каждой задачи множество базисных конфигураций, соответствующее этой задаче, удалось провести удачные эксперименты с несколькими простыми задачами самоприменения. Публикация об этих экспериментах появилась в 1990 году [15]. Таким образом, было достигнуто самоприменение алгоритмически разрешимой части задачи специализации как таковой, – без участия аппроксимирующего алгоритма обобщения.

В 1990 году ученик В. Ф. Турчина Н. В. Кондратьев (ИПС АН СССР) предпринял попытку [30] реализации суперкомпилятора РЕФАЛа, которая осталась незавершённой. Внутренним языком преобразований являлся вариант языка РЕФАЛ графов.

Аналогичную незавершённую попытку в 1992 году сделали другие ученики В. Ф. Турчина: С. М. Абрамов и Р. Ф. Гурин. Упростив задачу, они разрабатывали суперкомпилятор для модельного языка, работающего с данными LISPа. Основная трудность, которую они не смогли преодолеть, состояла в разработке алгоритма построения выходных форматов функций [1].

В 1993 году Анд. В. Климов и Р. Глюк (Австрия-Дания) опубликовали статью «Occam's razor in metacomputation: the notion of a perfect process tree» [14], в которой алгоритм прогонки излагался в терминах данных языка LISP (бинарных деревьев). Основной целью этой публикации было ознакомление зарубежных исследователей с некоторыми простыми идеями суперкомпиляции, демонстрируя их на более простых (по сравнению с РЕФАЛом) данных. (По разным причинам, важность ассоциативности конструктора конкатенации РЕФАЛ данных до сих пор по достоинству не оценена за пределами России.) Эта работа позволила уточнить некоторые понятия алгоритма прогон-

ки. В статье представлен также суперкомпилятор простейшего модельного LISP-подобного языка. Этот простой суперкомпилятор также *a priory* предполагает конечность процесса специализации, не используя критический алгоритм обобщения, аппроксимирующий алгоритмическую неразрешимость задачи специализации.

SCP3. Опыт ручного построения множества базисных конфигураций в экспериментах самоприменения SCP2 показал, что на пути удовлетворительного решения задачи полностью автоматического самоприменения суперкомпилятора стоит ещё много трудных проблем. В 1993 году В. Ф. Турчин решил ограничить объектный язык суперкомпилятора «плоским» (flat) алгоритмически полным подмножеством базисного РЕФАЛа-5, в котором не допускаются явные синтаксические конструкции композиции вызовов функций и синтаксис образцов гарантирует, что время отождествления равномерно ограничено по размеру входных данных. Основной целью разработки суперкомпилятора SCP3, преобразующим «плоские» РЕФАЛ программы, было достижение его полностью автоматического самоприменения. При этом SCP3 разрабатывался в терминах полного РЕФАЛа-5. Перед самоприменением предполагалось транслировать исходные тексты SCP3 в «плоский» РЕФАЛ. При построении суперкомпилятора SCP3 критическим шагом было расширение языка параметров, описывающего конфигурации специализируемой программы: введение дополнительной типизации параметров, позволяющей более точно формулировать задачи на самоприменение. В 1994 году удалось достигнуть *полностью автоматического самоприменения* суперкомпилятора SCP3 на нескольких простых задачах самоприменения. *Этим был решен долгое время стоявший открытым вопрос о принципиальной возможности самоприменения специализатора, построенного на основе методов суперкомпиляции.* В 1996 году появилась статья «A Self-Applicable Supercompiler» (В. Ф. Турчин, А. П. Немытых, В. А. Пинчук [41]) с изложением основных идей, позволивших провести эти удачные эксперименты, и описанием самих экспериментов. Алгоритм обобщения плоских конфигураций всё еще не имел под собой прочной теоретической основы, хотя и работал *полностью автоматически.*

В 1995 году М. Соренсен (Дания) [48] предложил использовать отношение Хигмана-Краскала [18], [32] для принятия важного аппроксимирующего решения алгоритма обобщения: «Обобщать или не обобщать две данные конфигурации?». Это предложение поставило на прочную теоретическую основу алгоритм обобщения «положительной» части конфигураций, описанной *без использования логического отрицания.* В 1996 году вышла статья М. Соренсена, Р. Глюка и Н. Д. Джонса, описывающая модельный суперкомпилятор для простейшего подмножества языка LISP. Язык описания параметризованных конфигураций в этом суперкомпиляторе не использует связки отрицания.

В 1995 году выходит монография С. М. Абрамова «Метавычисления и их применение» [2] (ученик В. Ф. Турчина, ИПС РАН), в которой описывается алгоритм обобщения части параметризованной конфигурации, описанной с

использованием связки отрицания, но заданной только в терминах данных *единичного размера* («символов»).

SCP4. Результатом продолжительных (под руководством В. Ф. Турчина) исследований автора данной статьи была разработка и реализация экспериментального суперкомпилятора SCP4 (1999-2003) для реального языка программирования РЕФАЛ-5 (без какого бы ни было ограничения на этот язык). В процессе этой работы были разработаны и реализованы принципиально новые алгоритмы преобразований. Самым ключевым (из разработанного набора алгоритмов глобального анализа) из которых стал алгоритм автоматического построения выходного формата функции[8] F в режиме online (по ходу дела суперкомпиляции); что позволяет сразу использовать построенный формат для специализации по нему как других функций, вызывающих функцию F, так и самой функции F. До настоящего времени суперкомпилятор SCP4 является единственным экспериментальным свободно распространяемым суперкомпилятором, который доступен также в режиме online. В 2007 году в издательстве УРСС вышла монография А. П. Немытых «Суперкомпилятор SCP4: Общая структура» [40].

Задача суперкомпиляции является по существу трудной задачей и по своей природе задачей аппроксимирующей. Практически любая содержательная проблема на оптимизацию является алгоритмически неразрешимой. Задача состоит, с одной стороны, в последовательном движении к расширению существующих методов и алгоритмов и разработке новых, расширяющих возможности оптимизации; с другой стороны, в компактном описании этих алгоритмов, позволяющем контролировать программный код самого суперкомпилятора. Существующий в суперкомпиляторе SCP4 набор базисных методов преобразований позволяет получать достаточно интересные преобразования за счет разнообразия композиции этих методов. Здесь уместно сравнение с классической машиной Тьюринга, которая, обладая тривиальным набором базисных действий, позволяет описывать произвольный алгоритм посредством разнообразия композиции этих элементарных действий.

В 2008-2010 гг. [35], [26], [28] были разработаны и реализованы простые суперкомпиляторы для модельных языков высшего порядка[9]. В них не реализован ключевой Обнинский (см. выше по тексту) метод обобщения параметризованных стеков; с другой стороны, некоторые понятия суперкомпиляции адоптированы к функциям высших порядков.

[8] Автор пришёл к выводу о необходимости разработки и использования такого алгоритма после анализа результата самоприменения одной из рабочих версий суперкомпилятора SCP4, – на начальных стадиях его разработки.

[9] Фрагменты языка Haskell и аппликативный язык высшего порядка.

5 Структура суперкомпилятора SCP4

Для того чтобы дать читателю более полное представление о современном состоянии процесса суперкомпиляции, в этом разделе мы даём общую структуру суперкомпилятора SCP4. Чтобы сделать описание алгоритма обозримым, мы вынуждены допустить некоторые неточности (обратное увело бы нас в технические детали, не соответствующие формату данной статьи).

Пусть дан язык программирования L. Операционная семантика L определяет вычислительную модель этого языка. Вычислительная модель описывается в конечных терминах; то есть, она определяет один шаг (`step`) абстрактной L-машины в самом общем состоянии этой машины и далее предписывает повторять такие шаги до полного завершения всех вычислений (в нормальном или аварийном состоянии). `step` преобразует текущее состояние в непосредственно следующее за ним состояние. Назовем эти состояния конфигурациями L-машины. Обобщим это понятие: наряду с конкретными конфигурациями будем рассматривать параметризованные конфигурации (в некотором языке параметров) – часть данных может быть неизвестна (но, тем не менее, фиксирована).

Входными данными любого суперкомпилятора является пара: программа `P` и её *параметризованная* входная точка $\mathtt{p}(\overline{\mathtt{x}_0}, \overline{\mathtt{y}})$ (*т.е. стартовая конфигурация*). Для данной параметризованной конфигурации $C(p_1, \ldots, p_n)$, алгоритм, который В. Ф. Турчин назвал *прогонкой*, строит развертку *одного* шага `step`. Другими словами, результат прогонки есть конечное ориентированное дерево T с корнем $C(p_1, \ldots, p_n)$ такое, что для любой подстановки параметров θ в $C(p_1, \ldots, p_n)$ (обозначим результат этой подстановки как $C(p_1, \ldots, p_n)\theta$) существует путь из корня в некоторый лист, – если вычисление `step` на $C(p_1, \ldots, p_n)\theta$ не приводит к аварийной остановке. Ребра в дереве T помечены предикатами, сужающими множества значений параметров и выбирающими конкретное вычисление. Ребра в T, исходящие из данной вершины, также упорядочены – согласно операционной семантике шага `step`, который проверяет (в этом порядке) логические ветвления (`case, if`). Прогонка вычисляет (параметризованные) конфигурации, соответствующие узлам дерева T; узлы помечаются этими конфигурациями. (Обозначим узел `n`, помеченный конфигурацией `c` как $\mathtt{n}_c$.) Она обрезает недостижимые ветви, если способна распознать их. Параметризованная конфигурация (как и узел, помеченной этой конфигурацией) называется *пассивной*, если ее описание не содержит терминов стека функций (то есть, ничего вычислять не нужно), иначе конфигурация называется *активной*.

Поясним на содержательном уровне бинарное отношение Хигмана-Краскала [18], [32] (обозначим его $\succ$)[10] на множестве параметризованных конфигураций. Это отношение можно понимать как отношение «*сложнее чем*». Пусть даны две параметризованные конфигурации $C_1(p_1, \ldots, p_n)$ и $C_2(q_1, \ldots, q_m)$. Скажем, что $C_1(p_1, \ldots, p_n) \succ C_2(q_1, \ldots, q_m)$, если

[10] О котором мы уже упоминали в разделе 4.

$C_1(t,\ldots,t) \neq C_2(t,\ldots,t)$ и $C_2(t,\ldots,t)$ можно получить из $C_1(t,\ldots,t)$ стиранием синтаксических конструкций. Где t – некоторый формальный параметр и под синтаксическими конструкторами мы понимаем термы[11] и конструкторы термов вместе с их скобками (если они есть). Например, `(a b c` p_1 `f(d e)) ≻ b c` p_2 `(e)`, `((()()) ≻ (())`. Но `(a b c d e)` и `(a c b d e)` несравнимы.

Узел ориентированного графа G назовем листом, если из него не выходит ни одного ребра. Пусть в G выделена вершина (без входящих рёбер), которая называется корнем. Пусть для каждой вершины n графа G, ребра исходящие из n упорядочены. Тогда множество всех путей, начинающихся в корне (ниже мы будем рассматривать только такие пути, если не оговаривается обратное) естественно лексикографически упорядочено, согласно последовательностям ребер, образующим эти пути. Ниже слова *наименьший* и знак $\leq$ относятся к этому лексикографическому порядку.

Теперь мы готовы дать грубый набросок алгоритма суперкомпиляции.

```
graph := стартовая параметризованная конфигурация ;
basics := nil;        /* Список базисных конфигураций/узлов. */
while ( в graph-е есть активный лист )
{ c := конфигурация из активного узла n_c, являющегося концом наименьшего
       пути p;
  d := driving(c);
  if( d не удовлетворяет условиям некоторой стратегии ) then continue;
  if(graph содержит конфигурацию c_1 на некотором пути q ≤ p,
      такую, что n_{c_1} ≠ n_c и c = c_1ζ есть результат подстановки параметров в c_1
    ) then                                        /* Свёртка. */
      { создать ребро (помеченное подстановкой ζ) из n_c в n_{c_1} ;
        if( любой путь, начинающийся в n_{c_1}, свёрнут или заканчивается в
            пассивном узле )
          then
           { properties := global-analyze( подграф с корнем (входом) n_{c_1});
             проспециализировать graph по глобальным свойствам
                  properties ;
           };
         basics := n_{c_1} ++ basics;
       }
      else if( p содержит конфигурацию c_2 такую, что n_{c_2} ≠ n_c и c ≻ c_2, и
                эти две конфигурации "похожи" (в некотором смысле)
             ) then { создать обобщенную конфигурацию[12] c_g такую,
                          что c и c_2 являются примерами
                          (результатами подстановок) конфигурации c_g;
                      стереть поддерево с корнем n_{c_2}, исключая саму n_{c_2} ;
                      выкинуть узлы этого поддерева из списка basics;
                      заменить узел n_{c_2} узлом n_{c_g} ;
                    }
```

[11] Можно считать, что это выражения языка РЕФАЛ или языка ЛИСП.

[12] Грубо можно понимать как наиболее точную/узкую обобщенную конфигурацию.

```
                else { заменить узел n_c деревом d; }
}
```

Остаточная программа описывается в терминах базисных конфигураций: они являются входными форматами функций (подпрограмм) остаточной программы.

6 Суперкомпиляция и метод частичных вычислений

Для сравнения суперкомпиляции с методом частичных вычислений мы снова рассмотрим машину Тьюринга (ТМ).

В чём сущность подхода Н. Д. Джонса, упростившего оригинальные идеи *online* преобразования программ и сформулировавшего основные понятия метода частичных вычислений? Ответ на этот вопрос состоит в следующем. Пусть дано некоторое конечное множество элементарных преобразований программ $\{q_1, q_2, \ldots, q_n\}$ (т.е. исчисление). Суперкомпилятор жонглирует ими с целью оптимизации данной входной программы P. Одна часть этих элементарных преобразований (пусть это будут $\{q_1, q_2, \ldots, q_m\}$) ответственна за обобщение параметризованных конфигураций программы P, другая же часть напрямую используется для метаинтерпретации шагов P (т.е. непосредственно для специализации). Как уже отмечено выше (см. раздел 3) BTA-алгоритм является аналогом алгоритма обобщения. Сущность идеи Н. Д. Джонса заключается в том, чтобы манипулировать преобразованиями $\{q_1, q_2, \ldots, q_m\}$ только посредством BTA-анализа; и результат этих манипуляций должен быть подан на вход второй стадии преобразований, которая использует только вторую часть элементарных преобразований. Такое разделение немедленно приводит к катастрофическим последствиям. Чтобы прочувствовать эти последствия, рассмотрим пример ТМ. Применим идею Н. Д. Джонса к базисным операциям ТМ[13]

$$\{t_1, \ldots, \mathtt{move}_{\mathtt{to\_left}}, \mathtt{move}_{\mathtt{to\_right}}, \ldots, t_k\}$$

и разобьём этот набор операций на два подмножества:

$$\{t_1, \ldots, \mathtt{move}_{\mathtt{to\_left}}\} \text{ и } \{\mathtt{move}_{\mathtt{to\_right}}, \ldots, t_k\}.$$

Теперь согласно методу частичных вычислений мы должны сначала манипулировать только операциями $\{t_1, \ldots, \mathtt{move}_{\mathtt{to\_left}}\}$ и только после этого нам разрешается использовать операции из второго подмножества. Манипулируя всем набором, мы можем построить любой алгоритм. Но что можно запрограммировать, используя подход Н. Д. Джонса? Ответ очевиден!

[13] Здесь $\mathtt{move}_{\mathtt{to\_left}}$ и $\mathtt{move}_{\mathtt{to\_right}}$ означают операции сдвига головки машины Тьюринга.

7 Заключение

В заключение необходимо сказать об одной фундаментальной проблеме, которая относится к любым методам оптимизации программ, а не только к суперкомпиляции. О методах оптимизации можно говорить только в контексте конкретной модели вычислений или, в более технических терминах, – в контексте конкретной реализации языка программирования. Критические свойства конкретных моделей вычислений, связанные с оптимизацией, как правило, вообще неизвестны исследователям, которые работают в области разработки методов преобразований программ (в рамках одного языка программирования). Операционная семантика языка, известная пользователю, оставляет большую свободу разработчикам интерпретаторов и компиляторов. Например, реализации следующих диалектов языка РЕФАЛ: РЕФАЛ-5 ([57], [61]), РЕФАЛ+ ([17], [16]), FLAC ([13], [12], [5]) различаются критическими с точки зрения оптимизации свойствами их конкретных моделей вычислений. Это замечание относится и к тому фрагменту языка РЕФАЛ, на котором, с точки зрения пользователя, все эти диалекты полностью совпадают. Следовательно, бессмысленно говорить о суперкомпиляторе языка РЕФАЛ; можно лишь разрабатывать суперкомпилятор конкретной реализации РЕФАЛа.

На решение этой проблемы еще в начале 1980-х годов указал А. П. Ершов [6]: «Автору представляется весьма важным создание концепции трансформационной машины как некоторого целостного устройства, системой команд которой являются базовые трансформации, а программами - программные процессоры. Предметной областью трансформационной машины являются программные тексты и их данные. Мы снова, как в машине фон Неймана, операционно не различаем программу и ее данные, но уже, так сказать, на более высоком уровне, когда программа и ее данные в равной степени обрабатываются программным процессором.»[14].

К сожалению, на наш взгляд, важность разработки трансформационной машины, т.е. включение преобразователей программ, таких как суперкомпиляторы, в рамки конкретной реализации языка программирования, не вполне осознаётся.

Список литературы

1. Абрамов С. М., Гурин Р. Попытка построения суперкомпилятора для языка РЕФАЛ+. 1992, (не опубликовано, частное сообщение).
2. Абрамов С. М. Метавычисления и их применения. 1995, Наука-Физматлит, Москва.
3. Библиотека по языку рефал. http://refal.botik.ru/library/library.htm , 2010-2011 гг.
4. Chang C., Lee R. C. Symbolic Logic and Mechanical Theorem Proving, 1973, Academic Press.

[14] Русский оригинал статьи доступен на странице: http://ershov.iis.nsk.su/russian/articles_pdf/08.pdf

5. Chmutov S. V., Gaydar E. A., Ignatovich I. M., Kozadoy V. F., Nemytykh A. P., Pinchuk V.A. Implementation of the symbol analitic transformation language FLAC. In the Proc. of DISCO'90, LNCS, Vol. **429** (1990) p.276, Springer-Verlag. URL: http://www.botik.ru/pub/local/scp/flac/FLAC_LNCS_N428_1990.djvu
6. Ershov A. P. Mixed computation: potential applications and problems for study. In: Theoretical Computer Science. Vol. **18**, no. **1**, (1982), pp:41–67.
7. Ershov A. P., Ляпунов А. А. О формализации понятия программы. 1967 г. URL: http://ershov.iis.nsk.su/archive/eaindex.asp?lang=1&did=13796
8. Futamura Y. Partial Evaluation of Computation Process – An Approach to a Compiler-Compiler. In: Systems. Computers. Controls. **2(5)** (1971) 45–50.
9. Futamura Y., Nogi K. Generalized partial computation In the Proc. of the IFIP TC2 Workshop, (1988) 133–151. Amsterdam: North-Holland Publishing Co.
10. Futamura Y., Nogi K., Takano A. Essence of generalized partial computation. Theoretical Computer Science. **90** (1991) 61–79. Amsterdam. North-Holland Publishing Co.
11. Futamura Y., Konishi Z., Glück R. Program Transformation System Based on Generalized Partial Computation. New Generation Computing. Vol. **90** (2002) 75–99. Ohmsha Ltd. and Springer-Verlag.
12. Гайдар Е. А., Игнатович И. М., Козадой В. Ф., Немытых А. П., Пинчук В. А., Чмутов С. В. Реализация системы программирования FLAC. Технический отчёт, ИПС АН СССР, Переславль-Залесский, 1988, 58 с. URL: http://www.botik.ru/pub/local/scp/flac/fl_impl.pdf
13. Гайдар Е. А., Игнатович И. М., Козадой В. Ф., Немытых А. П., Пинчук В. А., Чмутов С. В. Функциональный язык для алгебраических вычислений FLAC. Технический отчёт, ИПС АН СССР, Переславль-Залесский, 1988, 42 с. URL: http://www.botik.ru/pub/local/scp/flac/fl_lang.pdf
14. Glück R., Klimov And. V. Occam's razor in metacomputation: the notion of a perfect process tree. In Proc. of the Static Analysis Symposium, LNCS, Vol. **724** (1993) 112–123, Springer-Verlag.
15. Glück R., Turchin V. F. Application of metasystem transition to function inversion and transformation. In the Proc. of the ISSAC'90 (1990), 286–287. ACM Press.
16. Гурин Р. Ф., Климов Ю. А., Орлов А. Ю., Романенко С. А. РЕФАЛ+: исполняемые модули и исходные тексты. URL: http://rfp.botik.ru/
17. Гурин Р. Ф., Романенко С. А. Язык программирования Рефал Плюс. - М.: ИНТЕРТЕХ, 1991
18. Higman G. Ordering by divisibility in abstract algebras. Proc. London Math. Soc. **2(7)** (1952) 326–336.
19. Hughes J. Type Specialization for the Lambda-Calculus; or a new Paradigm for Partial Evaluation Based on Type Inference. In Proc. the PEPM'96, LNCS, Vol. **1110** (1996) 183–215, Springer-Verlag.
20. Jones N.D. MIX ten years later. In the Proc. of the ACM SIGPLAN PEPM'95, (1995) 24–38, ACM Press.
21. Jones N.D. What not to do when writing an interpreter for specialization. In Proc. the PEPM'96, LNCS, Vol. **1110** (1996) 216–237, Springer-Verlag.
22. Jones N. D. Computability and Complexity from a Programming Perspective. (1997) The MIT Press.
23. Jones N.D., Gomard C.K., Sestoft P. Partial Evaluation and Automatic Program Generation. (1993) Prentice Hall International.
24. Jones N.D., Sestoft P., Søndergaard H. An experiment in partial evaluation: the generation of a compiler generator. In Proc. of Conf. on Rewriting Techniques and Applications, LNCS, **202** (1985), 125–140. Springer-Verlag.

25. Jones N.D., Sestoft P., Søndergaard H. Mix: A Self-Applicable Partial Evaluator for Experiments in Compiler Generation. Lisp and Symbolic Computation, **2(1)** (1989) 9–50.
26. Jonsson P.A., Nordlander J. Positive Supercompilation for a higher order call-by-value language. ACM SIGPLAN Notices., (2009), Vol. **44**, no. **1**, pp:277–288, ACM Press.
27. Климов Анд. В., Романенко С. А. Метавычислитель для языка РЕФАЛ, базисные понятия и примеры. Препринт № **71** (1987), Институт прикладной математики им. М. В. Келдыша АН СССР, Москва.
28. Ключников И. Г. Суперкомпилятор HOSC 1.5: гомеоморфное вложение и обобщение для выражений высшего порядка. Препринт № **62** (2010), Институт прикладной математики им. М. В. Келдыша РАН, Москва.
29. Knuth D. E., Morris J. H., Pratt V. R. Fast Pattern Matching in strings. SIAM J. Comput., Vol. **6(2)** (1977) 323–350.
30. Кондратьев Н. В. Подходы к построению суперкомпилятора. (1990), (не опубликовано, частное сообщение).
31. Корлюков А. В. Пособие по суперкомпилятору SCP4. (1999) URL: http://www.refal.net/supercom.htm
32. Kruskal J.B. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. Trans. Amer. Math. Society, **95** (1960) 210–225.
33. Lee C. S., Jones N. D., Ben-Amram A. M. The Size-Change Principle for Program Termination. ACM Symposium on Principles of Programming Languages. **28** (2001) 81–92, ACM press.
34. Leuschel M. Homeomorphic Embedding for Online Termination. In Proc. of the 8th Int. Workshop on Logic Program Synthesis and Transformation (LOPSTR), LNCS, Vol. **1559**, 199–218. Springer-Verlag.
35. Mitchell N., Runciman C. A supercompiler for core Haskell. In Implementation and Application of Functional Languages, LNCS, Vol. **5083**, 147–164. Springer.
36. Mogensen T. Evolution of Partial Evaluators: Removing Inherited Limits. In Proc. the PEPM'96 LNCS, Vol. **1110** (1996) 303–321, Springer-Verlag.
37. Nemytykh A.P. A Note on Elimination of Simplest Recursions.. In Proc. of the ACM SIGPLAN Asian Symposium on Partial Evaluation and Semantics-Based Program Manipulation, (2002) 138–146, ACM Press.
38. Nemytykh A.P. The Supercompiler SCP4: General Structure (extended abstract). In Proc. of the Perspectives of System Informatics, LNCS, **2890** (2003) 162–170, Springer-Verlag.
39. Nemytykh A.P. Playing on REFAL. In Proc. of the International Workshop on Program Understanding, (2003) 29–39, A.P. Ershov Institute of Informatics Systems, Syberian Branch of Russian Academy of Sciences. Accessible via: ftp://www.botik.ru/pub/local/scp/refal5/nemytykh_PU03.ps.gz
40. Немытых А. П. Суперкомпилятор SCP4: общая структура. Монография, М: Издательство УРСС, 2007, ISBN 978-5-382-00365-8. - 152 с.
41. Nemytykh A. P., Pinchuk V. A., Turchin V. F. A Self-Applicable Supercompiler. In Proc. the PEPM'96 LNCS, Vol. **1110** (1996) 322–337, Springer-Verlag. (ftp://ftp.botik.ru/pub/local/APP/self-appl.ps.gz).
42. Nemytykh A.P., Turchin V.F. The Supercompiler SCP4: sources, on-line demonstration, http://www.botik.ru/pub/local/scp/refal5/, (2000).
43. Романенко С. А. Генератор компиляторов порожденный само-применимым специализатором может иметь ясную и естественную структуру. Препринт № **20** (1987), Институт прикладной математики им. М. В. Келдыша АН СССР, Москва.

44. Романенко С. А. РЕФАЛ-4 – расширение РЕФАЛа-2, обеспечивающее выразимость прогонки. Препринт № **147** (1987), Институт прикладной математики им. М. В. Келдыша АН СССР, Москва.
45. Romanenko S. A. A compiler generator produced by a self-applicable specializer can have a surprisingly natural and understandable structure. In The Proc. of the IFIP TC2 Workshop, Partial Evaluation and Mixed Computation, (1988) 445–463, North-Holland Publishing Co.
46. Romanenko S. A. Arity raiser and its use in program specialization In the Proc. of the ESOP'90, LNCS, Vol. **432** (1990) 341–360, Springer-Verlag.
47. Sestoft P. The structure of a self-applicable partial evaluator. In the Proc. of the Programs as Data Objects, LNCS, Vol. **217** (1986) 236–256, Springer-Verlag.
48. Sørensen M. H., Glück R. An algorithm of generalization in positive supercompilation. Logic Programming: Proceedings of the 1995 International Symposium (1995) 486–479, MIT Press.
49. Sørensen M. H., Glück R., Jones N. D. A positive supercompiler. Journal of Functional Programming, Vol. **6(6)** (1996) 811–838, MIT Press.
50. Турчин В.Ф. Эквивалентные преобразования рекурсивных функций, описанных на языке РЕФАЛ. В сб.: Труды симпозиума «Теория языков и методы построения систем программирования», Киев-Алушта: 1972. Стр. 31 – 42.
51. Turchin V.F. The use of metasystem transition in theorem proving and program optimization. In Proc. the 7th Colloquium on Automata, Languages and Programming, LNCS, Vol. **85** (1980) 645–657, Springer-Verlag.
52. Turchin V.F. The language Refal – The Theory of Compilation and Metasystem Analysis. Courant Computer Science Report, Num. **20** (February 1980), New York University.
53. Turchin V.F. The algorithm of generalization in the supercompiler. In the Proc. of the IFIP TC2 Workshop, (1988) 531–549.
54. Turchin V.F. The concept of a supercompiler. ACM Transactions on Programming Languages and Systems. **8** (1986) 292–325, ACM Press.
55. Turchin V.F. Metacomputation: Metasystem transition plus supercompilation. In Proc. the PEPM'96, LNCS, Vol. **1110** (1996) 481–509, Springer-Verlag.
56. Turchin V.F. Supercompilation: Techniques and results. In the Proc. of PSI'96, LNCS, Vol. **1181** (1996) 227–248, Springer-Verlag.
57. Turchin V.F. Refal-5, Programming Guide and Reference Manual. Holyoke, Massachusetts. (1989) New England Publishing Co.
(electronic version: http://www.botik.ru/pub/local/scp/refal5/ ,2000)
58. Турчин В. Ф. Metacomputation in the language Refal. Обнинск. (1990). (Электронная версия: Библиотека по языку рефал, http://refal.botik.ru/library/library.htm, 2010)
59. Turchin V. F., Nemytykh A. P. Metavariables: Their implementation and use in Program Transformation, Technical Report CSc. **TR 95-012** (1995), City College of the City University of New York.
60. Turchin V. F., Nireberg R., Turchin D. V. Experiments with a supercompiler Conference Record of the ACM Symposium on LISP and Functional Programming, (1982) 47–55, ACM Press.
61. Turchin V.F., Turchin D.V., Konyshev A.P., Nemytykh A.P. Refal-5: sources, executable modules. URL: http://www.botik.ru/pub/local/scp/refal5/, (2000)
62. Wadler P. Deforestation: Transforming programs to eliminate trees. Theoretical Computer Science, Vol. **73** (1990) 231–238.
63. Шура-Бура М. Р. Мой Келдыш. URL: http://келдыш.рф/shura-bura.htm, 2001