О суперкомпиляции

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

(Тезисы доклада)

Андрей П. Немытых

Институт программных систем РАН
nemytykh@math.botik.ru

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. Исследования в области построения систематических методов специализации программ по отношению к фиксированным свойствам их аргументов были начаты в 1970-х годах А.П. Ершовым («смешанные вычисления»), В.Ф. Турчиным («суперкомиляция») и Ё. Футамурой («generalized partial computation»).

Идеи суперкомиляции в основном изучались в предметной области функционального языка рефал [40], хотя ряд результатов отшлифовывался на экспериментальной базе функционального языка LISP и его потомков. В настоящее время, наряду с несколькими примитивными моделями суперкомпиляторов [32], [19], [14], [16], построенными для простейших чисто теоретических языков, существует единственный экспериментальный суперкомпилятор SCP4 для реального языка программирования рефал-5 [40], [44]. Ниже нам понадобится следующее определение:

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

Будем использовать обозначение $\mathtt{p(x)}$ для $\mathtt{U(p,x)}$. В литературе рассматриваются две постановки задачи специализации как таковой. Сформулируем их. Пусть дана реализация функционального языка программирования $\langle \mathtt{P,D,U,T}\rangle$, где $\mathtt{D} = \bigcup_{n\in\mathbb{N}} M^n$ для некоторого множества M.

Задача №1. Пусть $\mathtt{p(x,y)} \in \mathtt{P}$ 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 $\mathtt{q(y)} \in \mathtt{P}$ такую, что

$$\forall \mathtt{y} \in \mathtt{D}.(\mathtt{q(y)} = \mathtt{p(x_0,y)}) \wedge (\mathtt{T(q,y)} \leq \mathtt{T(p,x_0,y)}),$$

где значение $\mathtt{q(y)}$ определено тогда и только тогда, когда определено $\mathtt{p(x_0,y)}$. В случае неопределённости $\mathtt{p(x_0,y_0)}$ и $\mathtt{q(y_0)}$ их типы неопределённости должны совпадать: программа $\mathtt{q(y)}$ представляет ту же самую *частично рекурсивную функцию*, что и программа $\mathtt{p(x_0,y)}$, а именно – $F(\mathtt{x_0,y})$.

Задача №2. Пусть $\mathtt{p(x,y)} \in \mathtt{P}$ реализует *общерекурсивную функцию* $F(x,y) : X \times Y \mapsto \mathtt{D}$, где $X \subset \mathtt{D}, Y \subset \mathtt{D}$. Зафиксируем значение первого аргумента этой функции $x_0 \in X$. Требуется построить $\mathtt{q(y)} \in \mathtt{P}$ такую, что

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

Т.е., во второй задаче программа q представляет некоторое *продолжение общерекурсивной функции* $F(x_0, y) : Y \mapsto \mathtt{D}$ по второй компоненте.

Программу q называют *остаточной* программой. *Содержательная сторона задачи состоит в построении оптимальной q (по времени исполнения).* Разные уточнения понятия оптимальности (функциии T) определяют конкретные аппроксимации задачи специализации как таковой. Первая задача, грубо говоря, требует от q сохранения операционной семантики исходной программы p. Вторая задача более естественна: пользователя обычно не интересует каким будет операционное поведение q на входных данных, не принадлежащих к области определения его предметной области. Условия второй задачи оставляют бóльшую свободу методам специализации, что часто позволяет строить более оптимальные остаточные программы по сравнению с методами действующими в рамках первой задачи.

1 Обзор результатов в специализации программ

На пути реализации идей, сформулированных А.П. Ершовым, В.Ф. Турчиным и Ё. Футамурой, встретились серьёзные трудности. Н.Д. Джонс предложил ослабить первоначально поставленные цели за счёт упрощения методов специализации [11], [10]. Эта 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 1983 г. им удалось решить аппроксимирующие задачи самоприменения частичного вычислителя mix. Заметим, что всегда существует функция времени T, делающая любую задачу специализации программы $\mathtt{p}(\mathtt{x}_0,\mathtt{y})$ бессодержательной: функция T, которая позволяет построить следующую остаточную программу $\mathtt{q(y)}$ { = $\mathtt{p}(\mathtt{x}_0,\mathtt{y})$; }. Первые результаты самоприменения mix, содержательно, незначительно отличались от приведённой тривиальной остаточной программы. Как пишет Н.Д. Джонс [10], длина остаточной программы, полученной в результате решения простейшей задачи самоприменения $\mathtt{mix}(\underline{\mathtt{mix}(\mathtt{p}_0,\mathtt{x},\mathtt{y})})$[1] mix по данной трёхстрочной программе $\mathtt{p}_0\mathtt{(x,y)}$, была пятьсот страниц текста. Здесь значения y неизвестны обеим копиям mix; значение аргумента x известно специализируемому mix, но неизвестно специализирующему mix. Анализируя остаточную программу, группа Джонса предложила понятия «online» и «offline» методов специализации [11]. Выбор более простых offline методов

[1] Здесь подчёркивание означает кодировку.

позволил в 1984 г. решить более приемлемые аппроксимирующие задачи самоприменения частичного вычислителя `mix` [11], [30]. Введя инструменты повышения местности специализируемых программ в рамках offline подхода, С.А. Романенко [26], [28], [29] удалось в 1987 г. существенно улучшить временны́е и структурные характеристики остаточных программ задач самоприменения частичного вычислителя `unmix`.

Offline специализация разделяет в отдельные стадии 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[2]), подаётся `p` и указание – какие из её аргументов будут известны на второй стадии преобразований, а не сами значения этих аргументов, и какие неизвестны. Первые называются статическими, вторые динамическими. На выходе у BTA размеченная программа $\mathtt{p}^{\mathtt{ann}}$, 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 `p` производится BTA. Задача, поставленная перед BTA, как таковая алгоритмически неразрешима. Повсюду здесь имеется в виду некоторая аппроксимация BTA-задачи. На вход второй стадии, которая собственно и называется при этом подходе «*специализацией*», подаётся результат BTA – $\mathtt{p}^{\mathtt{ann}}$ и значения её статических аргументов. «Специализация» 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 $\mathtt{mix}(\mathtt{mix}^{\mathtt{ann}}(\mathtt{p}_0^{\mathtt{ann}},\mathtt{x},\mathtt{y}))$[3], которая существенно проще классической: обе копии `mix` производят лишь вторую стадию преобразований, не занимаясь связыванием по времени. Здесь важно, что входные данные каждого шага программы просматриваются только один раз на этапе «специализации».

Online специализация производит метавычисления шагов преобразуемой программы `p` по ходу дела анализа свойств этой программы; никак не ограничивая, вообще говоря, *a priori* себя ни в средствах, ни в количестве проходов по программе `p`. 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. Алгоритмически неразрешимая и разрешимая части этой логики никак не разделены. Резюме: разработка методов online специализации

[2] Binding time analysis.

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

сложнее разработки методов offline специализации. По определению, online специализация менее ограничена в методах, чем offline специализация, и, потому, потенциально значительно более сильная. Самым важным является то, что при offline специализации происходят преобразования только исходной программы p; а при online специализации могут происходить также преобразования подпрограмм, построенных самим специализатором, а не только подпрограмм программы p, написанных человеком.

Суперкомпиляция [39], [23] и generalized partial computation (GPC) [4], [6], 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. Методы суперкомпиляции [37], [39], [24], [24], [23], [20] и GPC [4], [5], 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. По-видимому, наименее разработанными на данный момент нужно считать идеи GPC. В отличие от частичных вычислений и суперкомпиляции, подход GPC к специализации незамкнут.

2 Обзор развития методов суперкомпиляции

Мы именует основные этапы истории развития идей суперкомпиляции следуя терминологии В.Ф. Турчина [38], [39]. Первая публикация В.Ф. Турчина «Эквивалентные преобразования рекурсивных функций, описанных на языке рефал» датируется 1972 г. [33]. Рефал изначально проектировался Турчиным как метаязык, ориентированный на преобразование программ. В этой статье описывается подмножество рефала, в котором время отождествления входных данных функции с образцом равномерно ограничено по размеру входных данных. На синтаксис образцов накладывается ограничение. Эти ограниченные образцы названы L-выражениями. Вводится понятие прогонки 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., [35]): работа изобилует примерами неалгоритмизированных преобразований программ, постановками интересных задач, многие из которых до сих пор не решены.

SCP1. Первая простейшая модель суперкомпилятора была реализована в Нью-Йорке в 1981 г. В.Ф. Турчиным, Б. Ниренбергом и Д.В. Турчиным [43] на одном из диалектов рефала. SCP1 работал в диалоговом режиме, запрашивая у человека результат обобщения конфигураций: 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 на этапе суперкомпиляции. Авторы проспециализировали несколько простых примеров: SCP1 решал *Задачу №2.*

SCP2 был разработан В.Ф. Турчиным в 1984 г. Статья В.Ф. Турчина «The concept of a supercompiler», опубликованная в 1986 г. [37] и описывающая некоторые идеи реализации суперкомпилятора SCP2, стала основным классическим трудом по методам суперкомпиляции. В языке описания параметризованных конфигураций программы появился конструктор отрицания, что позволило решить методами суперкомпиляции классическую задачу преобразования наивного алгоритма `p(s,x)` поиска подстроки `s` в строке `x` в алгоритм КМР [17]; посредством специализации исходной программы по первому аргументу $\texttt{p(s}_0\texttt{,x)}$. Показана возможность использования SCP2 для автоматического доказательства простых теорем существования. Алгоритм обобщения работает *ad hoc*, и для получения интересных преобразований требуется вмешательство человека в работу этого алгоритма.

Во второй половине 80-х гг. на одном из рефал семинаров А. Веденов объявил о скором завершении им реализации суперкомпилятора рефала. Публикаций и сообщений о фактической реализации не последовало. В 1987 г. выходят два препринта ИПМ АН, рассматривающие проблемы суперкомпиляции: «Рефал-4 – расширение рефала-2, обеспечивающее выразимость прогонки» (С.А. Романенко, [27]) и «Метавычислитель для языка рефал, основные понятия и примеры» (Анд. В. Климов, С.А. Романенко, [15]).

Второй по значимости работой В.Ф. Турчина стала статья «The algorithm of generalization in the supercompiler» (1988 г., [36]), в которой описывается алгоритм обобщения стека вызовов функций и доказывается теорема об остановке этого алгоритма. Алгоритм получил название «Обнинский алгоритм разрезания стека». Этот алгоритм является одним из самых важных алгоритмов суперкомпиляции. SCP2 был расширен этим алгоритмом [41].

В 1989 г. 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 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 [8]. Таким образом, было достигнуто самоприменение алгоритмически разрешимой части задачи специализации как таковой, – без участия аппроксимирующего алгоритма обобщения.

В 1990 г. Н.В. Кондратьев предпринял попытку реализации суперкомпилятора рефала, которая осталась незавершённой. Внутренним языком преобразований являлся вариант языка рефал графов. Аналогичную незавершённую попытку в 1992 г. сделали С.М. Абрамов и Р.Ф. Гурин. В 1993 г. Анд. В. Климов и Р. Глюк опубликовали статью [7], в которой алгоритм

прогонки излагался в терминах данных языка LISP. Целью этой публикации было ознакомление зарубежных исследователей с некоторыми простыми идеями суперкомпиляции, демонстрируя их на бинарных деревьях.

SCP3. В 1993 г. В.Ф. Турчин решил ограничить объектный язык суперкомпилятора 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. Критическим шагом было расширение языка параметров: введение дополнительной типизации параметров, позволяющей более точно формулировать задачи на самоприменение. В 1994 г. удалось достигнуть *полностью автоматического самоприменения* SCP3 на нескольких простых задачах. В 1996 г. появилась статья [24] (В.Ф. Турчин, А.П. Немытых, В.А. Пинчук) с изложением основных идей, позволивших провести эти удачные эксперименты. Алгоритм обобщения всё еще не имел под собой прочной теоретической основы, хотя и работал *полностью автоматически*.

В 1995 г. М. Соренсен [31] предложил использовать отношение Хигмна-Краскала [9], [18] для принятия аппроксимирующего решения: *Обобщать или не обобщать две данные конфигурации?* Это поставило на теоретическую основу алгоритм обобщения «положительной» части конфигураций, описанной *без использования логического отрицания*. В 1996 г. вышла статья М. Соренсена, Р. Глюка и Н.Д. Джонса, описывающая модельный суперкомпилятор для простейшего подмножества LISPа: язык описания параметризованных конфигураций не использовал связки отрицания. В 1995 г. выходит монография С.М. Абрамова «Метавычисления и их применение» [1], в которой дан алгоритм обобщения части параметризованной конфигурации, описанной с использованием связки отрицания, но заданной только в терминах данных *единичного размера*.

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. В процессе этой работы были разработаны и реализованы принципиально новые алгоритмы преобразований. Самым ключевым из которых стал алгоритм автоматического построения выходного формата функции F по ходу дела суперкомпиляции; что позволяет сразу использовать построенный формат для специализации по нему как других функций, вызывающих F, так и функции F. До настоящего времени SCP4 является единственным экспериментальным суперкомпилятором, который доступен также в режиме online. В 2007 г. вышла монография А.П. Немытых «Суперкомпилятор SCP4: Общая структура» [23]. В полной версии данной статьи даётся общая структура SCP4.

В 2008-2010 гг. [19], [14], [16] были разработаны и реализованы простые суперкомпиляторы для модельных языков высшего порядка. В них не реализован ключевой Обнинский метод обобщения параметризованных стеков.

Список литературы

1. Абрамов С.М. Метавычисления и их применения. 1995, Наука-Физматлит.
2. Библиотека по языку рефал. http://refal.botik.ru/library/library.htm
3. Ershov A. P. Mixed computation: potential applications and problems for study. In: Theoretical Computer Science. Vol. **18**, no. **1**, (1982), pp:41–67.
4. Futamura Y., Nogi K. Generalized partial computation In the Proc. of the IFIP TC2 Workshop, (1988) 133–151. Amsterdam: North-Holland Publishing Co.
5. Futamura Y., Nogi K., Takano A. Essence of generalized partial computation. Theoretical Computer Science. **90** (1991) 61–79. North-Holland Publishing Co.
6. 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.
7. 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.
8. 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.
9. Higman G. Ordering by divisibility in abstract algebras. Proc. London Math. Soc. **2(7)** (1952) 326–336.
10. Jones N.D. MIX ten years later. In the Proc. of the ACM SIGPLAN PEPM'95, (1995) 24–38, ACM Press.
11. Jones N.D., Gomard C.K., Sestoft P. Partial Evaluation and Automatic Program Generation. (1993) Prentice Hall International.
12. 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.
13. 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.
14. 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.
15. Климов Анд. В., Романенко С. А. Метавычислитель для языка РЕФАЛ, базисные понятия и примеры. Препринт № **71** (1987), ИПМ АН СССР, Москва.
16. Ключников И. Г. Суперкомпилятор HOSC 1.5: гомеоморфное вложение и обобщение для выражений высшего порядка. Препринт № **62** (2010), ИПМ РАН.
17. Knuth D. E., Morris J. H., Pratt V. R. Fast Pattern Matching in strings. SIAM J. Comput., Vol. **6(2)** (1977) 323–350.
18. Kruskal J.B. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. Trans. Amer. Math. Society, **95** (1960) 210–225.
19. Mitchell N., Runciman C. A supercompiler for core Haskell. In Implementation and Application of Functional Languages, LNCS, Vol. **5083**, 147–164. Springer.
20. 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.
21. Nemytykh A.P. The Supercompiler SCP4: General Structure (extended abstract). LNCS, **2890** (2003) 162–170, Springer-Verlag.
22. 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

23. Немытых А. П. Суперкомпилятор SCP4: общая структура. Монография, М: Издательство УРСС, 2007, ISBN 978-5-382-00365-8. - 152 с.
24. 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.
25. Nemytykh A.P., Turchin V.F. The Supercompiler SCP4: sources, on-line demonstration, http://www.botik.ru/pub/local/scp/refal5/, (2000).
26. Романенко С. А. Генератор компиляторов порожденный само-применимым специализатором может иметь ясную и естественную структуру. Препринт № **20** (1987), ИПМ АН СССР, Москва.
27. Романенко С. А. РЕФАЛ-4 – расширение РЕФАЛа-2, обеспечивающее выразимость прогонки. Препринт № **147** (1987), ИПМ АН СССР, Москва.
28. 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.
29. 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.
30. 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.
31. Sørensen M. H., Glück R. An algorithm of generalization in positive supercompilation. Logic Programming: Proc. of the 1995 Int. Symposium (1995) 486–479, MIT Press.
32. Sørensen M. H., Glück R., Jones N. D. A positive supercompiler. Journal of Functional Programming, Vol. **6(6)** (1996) 811–838, MIT Press.
33. 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.
34. Turchin V.F. The use of metasystem transition in theorem proving and program optimization.LNCS, Vol. **85** (1980) 645–657, Springer-Verlag.
35. Turchin V.F. The language Refal – The Theory of Compilation and Metasystem Analysis. Courant Computer Science Report, Num. **20** (1980), New York University.
36. Turchin V.F. The algorithm of generalization in the supercompiler. In the Proc. of the IFIP TC2 Workshop, (1988) 531–549.
37. Turchin V.F. The concept of a supercompiler. ACM Transactions on Programming Languages and Systems. **8** (1986) 292–325, ACM Press.
38. Turchin V.F. Metacomputation: Metasystem transition plus supercompilation. In Proc. the PEPM'96, LNCS, Vol. **1110** (1996) 481–509, Springer-Verlag.
39. Turchin V.F. Supercompilation: Techniques and results. In the Proc. of PSI'96, LNCS, Vol. **1181** (1996) 227–248, Springer-Verlag.
40. 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)
41. Турчин В. Ф. Metacomputation in the language Refal. (1990) (Электронная версия: Библиотека по языку рефал, http://refal.botik.ru/library/library.htm)
42. 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.
43. 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.

44. 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)