Российская академия наук
Сибирское отделение
Институт систем информатики
им. А.П. Ершова

На правах рукописи

Промский Алексей Владимирович

ФОРМАЛЬНАЯ СЕМАНТИКА C-LIGHT ПРОГРАММ И ИХ ВЕРИФИКАЦИЯ МЕТОДОМ ХОАРА

05.13.11 — математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени
кандидата физико-математических наук

Новосибирск, 2004

Работа выполнена в Новосибирском государственном университете и в Институте систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук

Научный руководитель:	кандидат физико-математических наук Непомнящий В.А.
Официальные оппоненты:	доктор физико-математических наук Серебряков В.А. кандидат физико-математических наук Скопин И.Н.
Ведущая организация:	Ярославский государственный университет (г. Ярославль)

Защита состоится 28 декабря в 14 час. на заседании диссертационного совета К003.032.01 в Институте систем информатики им. А. П. Ершова Сибирского отделения РАН по адресу:

630090, г. Новосибирск, пр. ак. Лаврентьева, 6.

С диссертацией можно ознакомиться в читальном зале библиотеки ИСИ СО РАН (пр. ак. Лаврентьева, 6).

Автореферат разослан "_____" ноября 2004 г.

Ученый секретарь
диссертационного совета
К003.032.01
к.ф.-м.н. Мурзин Ф.А.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность. Рост производительности компьютеров привел к росту размеров программ. Размеры и эффективность программ и коллективов, занимающихся их проектированием и сопровождением, не могут расти с одинаковой скоростью. При устоявшейся (и часто оптимистичной) оценке в одну ошибку на тысячу строк кода программы быстро могут стать неуправляемыми. Поэтому в ближайшие годы проблема *надежности программного обеспечения* может стать одним из основных вызовов для современных компьютерно-зависимых обществ.

До сих пор основным средством установления надежности программ в компьютерной индустрии остается тестирование. Оно же является и самым требовательным по времени и затратам. Поэтому рынок уже оказывает давление на разработчиков, требуя снижения издержек и развития академических исследований в области надежности программного обеспечения. Среди этих исследований важное место занимают методы формальной верификации программ.

Первым шагом на пути к верификации программ является формализация семантики языков программирования. Существенный вклад в эту область внесли исследования Д. Гриса, Э. Дейкстры, Р. Флойда, Ч. Э. Р. Хоара, Г. Плоткина. Среди отечественных авторов можно отметить А.П. Ершова, А.В. Замулина, С.С. Лаврова, А.А. Летичевского, В.А. Серебрякова и др. Из языков, для которых известны успешные результаты по формализации семантики, можно назвать языки Pascal, SML, Euclid, Eiffel. Однако для широко распространенных языков системного программирования успехи более скромные.

Среди таких языков большой интерес представляет язык C. Несмотря на появление и развитие более современных языков таких, как C++ и Java, язык C остается одним из самых распространенных в силу своих несомненных достоинств: низкоуровневые операции, быстрый код, компактное и легко переносимое ядро. К тому же система верификации для C могла бы стать базой для систем, ориентированных на язык C++.

Проблема верификации C-программ заключается в принципиальной трудности адекватной формализации этого языка. Сказывается его низкий уровень — возможность работать со значениями в памяти на уровне отдельных байтов и даже битов. Многие аспекты поведения C-программ в международном стандарте ISO/IEC не специфицированы и зависят от реализации. Поэтому актуальна проблема выделения выразительного подмножества языка C, для которого возможна строгая формализация.

Выбор семантики становится решающим фактором для сложности верификации. Наиболее популярны операционный и аксиоматический подходы. Первый хорошо подходит для формального описания исполнения программ абстрактным интерпретатором, что удобно для низкоуровневых аспектов. Но верификация в операционной семантике является громоздкой. Аксиоматический подход более абстрактный, он позволяет порождать условия, гарантирующие корректность программ. Однако аксиоматизация низкоуровневых конструкций затруднительна или сталкивается с теоретическими ограничениями. Поэтому в известных работах либо рассматриваются ограниченные подмножества языка C, либо предлагается формальное определение практически всего C, но не ориентированное на верификацию. В качестве примеров можно назвать работы Л. Андерсена, П. Блэка, М. Бофингера, И. Гуревича и Д. Хаггинса, М. Норриша, Н. Параспиру и др. К тому же большинство авторов не рассматривали последний стандарт языка C (C99).

Таким образом, актуальна проблема разработки и обоснования метода верификации C-программ, разумно сочетающего достоинства операционной и аксиоматической семантик. Для этого в исходном языке выделяется ограниченное ядро, для которого возможно создание аксиоматической семантики. Исходные программы транслируются в это ядро с их последующей верификацией. Методы трансляции, сохраняющие эквивалентность программ, также актуальны, особенно с появлением таких платформ, как “Microsoft .Net”, в которых может возникнуть задача трансляции между входными языками. Эквивалентные преобразования в C предлагались в работах М. Бокхолда, М. Хармана и др., однако корректность перевода не обосновывалась.

Наконец, актуально практическое воплощению теоретических методов упомянутой двухуровневой схемы верификации. Верификация программ на базе аксиоматической семантики использует генерацию условий корректности (УК). Из известных работ по генераторам можно назвать работы Р. Фраэра, Е. Грибомона, П. Хомейера, Д. Мартина, С. Игараши, Д. Лакхэма, Р. Лондона и Н. Сузуки. Также перспективным является дальнейшее развитие этих методов на основе концепции метагенерации, предложенной М. Морикони и Р. Шварцем. Метагенерация может упростить процесс создания и расширения генераторов.

Цель и задачи диссертации. Целью диссертационной работы является *исследование и разработка средств и методов формализации семантики C-light программ и их верификации методом Хоара.*

Для достижения цели поставлены следующие задачи:

- Выделение *представительного подмножества* C-light языка C99, расширенного операциями над динамической памятью и не содержащего конструкций, семантика которых существенно зависит от конкретной платформы. Создание *формального определения* языка C-light в виде *структурной операционной семантики*, обладающей свойством *детерминированности*.
- Выделение в языке C-light ядра C-kernel и создание *компактной и непротиворечивой аксиоматической семантики* этого ядра.
- Разработка *корректной системы правил перевода* из языка C-light в язык C-kernel.
- Разработка *(мета)генератора условий корректности* для языка C-kernel в *двухуровневой системе верификации* C-light программ.

Методы исследования. В работе использовались аппарат и методы математической логики, теории спецификации программ, структурного операционного подхода Плоткина, аксиоматического подхода Хоара и эквивалентных преобразований программ.

Научная новизна. В диссертационной работе получены новые научные результаты:

1. Структурная операционная семантика языка C-light поддерживает представительное подмножество C99 и обладает свойством детерминированности вычисления выражений, что позволяет придавать смысл широкому классу программ.
2. Система правил перевода из языка C-light в язык C-kernel обладает свойством корректности. Для доказательства корректности вместо обычного понятия функциональной эквивалентности предложено новое понятие семантического расширения.
3. Разработанная аксиоматическая семантика языка C-kernel является непротиворечивой и охватывает конструкции языка для работы с памятью. Благодаря двухуровневой схеме эта семантика применяется для верификации C-light программ.
4. Новый метод метагенерации условий корректности ориентирован на модифицированную семантику Хоара и позволяет упростить порождаемые условия корректности.

Практическая ценность. Полученные результаты могут стать основой для разработки методов формализации и верификации программ на современных языках системного программирования. Предложенные

методы в большой степени независимы от конкретной среды и могут быть адаптированы и расширены для различных реализаций языка C или таких языков, как C++, C#, Java.

Созданный прототип метагенератора УК используется для экспериментов в системе СПЕКТР-2, которая ориентирована на языки Pascal и C-light. Метагенерация позволила расширить систему правил вывода условий корректности правилами элиминации инвариантов циклов в программах линейной алгебры.

Аппробация работы. Основные положения работы докладывались на следующих научных конференциях:

1. *Четвертый Сибирский Конгресс по Прикладной и Индустриальной математике (ИНПРИМ-2000)*, Новосибирск, Россия, 2000.
2. *Конференция молодых ученых, посвященная 10-летию ИВТ СО РАН*, Новосибирск, Россия, 2001.
3. *Конференция, посвященная 90-летию со дня рождения А. А. Ляпунова*, Новосибирск, Россия, 2001.
4. *Международная конференция молодых ученых по математическому моделированию и информационным технологиям*, Новосибирск, Россия, 2002.
5. *International Conference "Perspectives of System Informatics"*, Novosibirsk, Russia, 2003.

Кроме того, полученные результаты обсуждались на совместных семинарах лаборатории теоретического программирования ИСИ СО РАН и кафедры программирования НГУ. Материалы диссертации вошли в отчеты по проектам РФФИ 00-01-00909 и 04-01-00114.

Публикации. По материалам диссертации опубликовано 11 работ.

Структура диссертации. Диссертация состоит из введения, пяти глав, заключения и приложений. Объем работы (за исключением приложений и библиографии) составляет 137 страниц в формате машинописного текста. Список литературы содержит 161 наименование.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Во **введении** обосновывается актуальность диссертации и формулируются ее цели, характеризуются научная новизна и практическая ценность работы.

В **главе 1** приводятся базовые понятия, используемые в работе, и краткий исторический обзор.

Разд. 1.1 представляет современный взгляд на формальную верификацию программ. Упомянуты основные подходы: дедуктивные, методы проверки моделей, анализ типов и статический анализ.

В разд. 1.2 рассмотрены используемые в работе виды семантик: *операционная* и *аксиоматическая.* Структурная операционная семантика (СОС) в стиле Г. Плоткина (п. 1.2.2) позволяет задать абстрактный интерпретатор для языка программирования. Базовое понятие СОС — *состояние*, в классических работах оно определяется как отображение программных переменных во множество их значений. Для привязки состояния к конкретной точке программы используется понятие *конфигурации.* Это пара $\langle S, \sigma \rangle$ — программный фрагмент S и состояние σ. Абстрактная машина определяет аксиомы и правила вывода, описывающие бинарное отношение перехода на множестве конфигураций. Такое описание, позволяющее выдавать следующее состояние, называется *семантикой мелкого шага* (small-step style). Исполнение программы соответствует цепочкам конфигураций, связанных переходами. Вводя транзитивное рефлексивное замыкание отношения перехода, можно по программе и начальному состоянию выдавать сразу заключительное состояние. Это так называемая *семантика крупного шага* (big-step style).

Аксиоматическая семантика (п. 1.2.3) предлагает более высокий уровень абстракции. Она представляет собой формальную систему вывода утверждений о свойствах программ. Основными формулами в ней являются конструкции вида $\{P\}\ S\ \{Q\}$, называемые *тройками Хоара.* Здесь S — это программа, а P и Q — логические утверждения, называемые пред- и постусловием, соответственно. Интуитивно истинность тройки *в смысле частичной корректности* означает следующее: если перед исполнением программы истинно предусловие P и программа S останавливается, то истинно постусловие Q. Свойства полноты и непротиворечивости логики Хоара обычно доказываются относительно исходной семантики СОС. В п. 1.2.5 сформулирована известная теорема Кларка о неполноте, которая описывает класс императивных языков, для которых недостижимо свойство полноты логики Хоара.

В разд. 1.3 рассмотрен основной объект изучения в диссертации — язык C. Приводятся краткий исторический обзор (п. 1.3.1) и особенности языка (п. 1.3.2), влияющие на практику его использования. Более подробно изложены проблемы его верификации (п. 1.3.3), в частности, рассмотрены особенности, перед которыми пасует классическая логика Хоара: недетерминизм вычислений, указатели, передача управления.

Разд. 1.4 содержит обзор некоторых теоретических работ по формализации семантики языка C и наиболее известных систем для анализа и верификации C-программ.

Глава 2 посвящена языку C-light и его структурной операционной семантике (СОС). Абстрактная машина языка C-light и СОС, задаваемая этой машиной, представляют основной материал данной главы.

В разд. 2.1 дается обзор языка C-light, который в основном является сравнением с языком C99. При этом для ряда элементов языка C99, не включенных в C-light, рассмотрены проблемы, возникающие при попытке их формализации. Для описания C-light в нем выделены *типы*, *декларации*, *выражения*, *операторы* и *программы*.

Допустимыми базовыми типами языка C-light являются все базовые типы языка C99 за исключением расширенных целых и комплексных. Допустимые составные типы включают указатели, массивы, структуры и функции. При этом на типы налагаются ограничения: 1) перечисления вводят не новые типы, а наборы констант типа `signed int`; 2) значения указателей — неинтерпретируемые символьные литералы. Допустимо приведение целого числа к типу указатель, но не наоборот; 3) запрещены указатели на функции; 4) неполные типы массивов разрешены только как типы аргументов функций; 5) в структурах запрещены битовые поля; 6) запрещены функции с переменным числом аргументов.

С декларациями связаны следующие ограничения: 1) множественные неокончательные определения для одного имени запрещены; 2) запрещены абстрактные декларации аргументов и спецификатор `static` в размере массива, являющегося параметром функции; 3) функция `main` не имеет параметров; 4) устаревший синтаксис определений функций (в стиле K&R) не поддерживается, требуются прототипы; 5) выделенные инициализаторы не поддерживаются; 6) запрещены все спецификаторы и модификаторы типов, кроме спецификаторов класса памяти, спецификаторов наличия знака и спецификаторов размера; 7) имена *всех* статических объектов в программе уникальны.

С выражениями связаны такие ограничения: 1) строго фиксирован порядок вычисления выражений; 2) побочные эффекты не откладываются до контрольных точек, а срабатывают на месте; 3) запрещены битовые операции, в том числе в составных присваиваниях; 4) составные литералы разрешены только в качестве инициализаторов; 5) приведение типов указателей ограничено приведением от `void*` к произвольному T`*`; 6) для выделения/освобождения памяти используются операции

`new` и `delete`, позаимствованные из языка C++.

Для операторов вводятся два ограничения: 1) все `case`-метки в операторе `switch` должны находиться на одном уровне вложенности; 2) запрещено передавать управление по `goto` внутрь блоков. Как и в C++, запрещено передавать управление по `goto` в обход инициализации.

Программа на языке C-light — это последовательность внешних деклараций. На внешнем уровне можно объявить тип, функцию, структуру и данные. Кроме обычных конструкций C на верхнем уровне могут присутствовать и аннотации. В языке C-light препроцессор отсутствует, поскольку исходная программа препроцессируется до начала ее верификации. Не предусмотрена никакая предварительная компиляция. Модульность не поддерживается. Поэтому любая исходная C-light программа состоит из одного файла и пользовательские библиотеки верифицируются отдельно от программы, которая использует их.

В разд. 2.2 рассмотрен язык утверждений, используемый для выражения свойств программ и состояний языка C-light. В дополнение к типам языка C-light он вводит идентификаторы, абстрактные адреса, абстрактные имена типов, функции над дополнительными типами, декартовы произведения и типы локаторов объектов. Алфавит языка утверждений состоит из алфавита языка логики первого порядка, расширенного знаками пунктуации и символами операций языка C-light. Все базовые типы расширены неопределенным значением ω. Выражения языка утверждений строятся стандартным образом и имеют префиксный синтаксис. Далее выражения типа `_Bool` называем просто *утверждениями*. Также индукцией по структуре выражений определяется понятие *подстановки* терма t вместо переменной u в выражении s, которая записывается в виде $s(u \leftarrow t)$.

В разд. 2.3 рассмотрена статическая часть абстрактной машины. Во-первых, в п. 2.3.1 модифицируется классическое понятие состояния. В абстрактной машине C-light *состояние* — это отображение, означивающее следующие метапеременные: 1) MeM — переменная типа Names $\times$ Nat $\to$ Locations, т.е. адресация объектов с учетом уровня их вложенности; 2) MD — переменная типа Locations $\to$ CTypes. Она определяет значения, хранимые в памяти; 3) Γ — переменная типа Names $\times$ Nat $\to$ TypeSpecs. Она определяет типы значений объектов; 4) STD — переменная типа Names $\to$ TypeSpecs, т.е. информация о тегах и синонимах типов; 5) Val — переменная типа CTypes $\times$ TypeSpecs, в которой хранится значение вычисленного выражения вместе с его типом.

При работе машины используются специальные функции. Они не являются функциями языка C-light и применяются только к состояниям. Перечислим основные: 1) Labels — функция, определяющая множество символических меток в блоке; 2) Cases — функция, определяющая множество меток-констант для оператора `switch`; 3) UnOpSem — функция, реализующая семантику унарных операций языка C-light; 4) BinOpSem — реализация семантики бинарных операций; 5) InstParms — реализация подстановки фактических аргументов вместо формальных параметров при вызове функции; 6) FindL(id) — определение уровня вложенности для имени id (разрешение области определенности); 7) Ip — реализация особого вида приведений — *целочисленных протяжек.*

Далее в п. 2.3.2 определяется система типов, сопоставляющая выражению его тип. Аксиомами этой системы являются утверждения о типах констант, которые включают числа, символы, строковые литералы, нулевой указатель. Правила вывода определяют типы более сложных выражений, содержащих идентификаторы, операции, вызовы функций.

В п. 2.3.3 расширяется классическое понятие конфигурации абстрактной машины. Теперь *конфигурация* — это тройка $\langle S, \sigma, lf \rangle$ — программа (фрагмент), текущее состояние и натуральное число lf, соотвествующее текущему уровню вложенности. Вводятся два бинарных отношения перехода между конфигурациями: 1) отношение для вычисления выражений: $\rightarrow_e$; 2) отношение для исполнения операторов: $\rightarrow_s$.

Произвольная аксиома COC для любого отношения выглядит как

$$\langle A, \sigma, n \rangle \rightarrow \langle B, \tau, m \rangle \ .$$

Она означает, что один шаг исполнения фрагмента A, начинающийся в состоянии σ на уровне n, приводит в состояние τ на уровне m и B — тот фрагмент исходной программы, который остается для исполнения. Произвольное правило семантики имеет вид

$$\frac{P_1 \quad ... \quad P_n}{\langle A, \sigma, n \rangle \rightarrow \langle B, \tau, m \rangle} \ .$$

Это означает, что при выполнении условий P_1,...,P_n можно перейти от первой конфигурации ко второй.

Вводится ряд специальных значений, которые могут храниться в компоненте Val наряду с обычными значениями. Однако эти значения не имеют типов и сигнализируют о том или ином событии в ходе работы программы: 1) OkVal — нормальное завершение работы оператора/програмы; 2) GoVal(L) — был встречен оператор `goto L`; 3) BreakVal

— был встречен оператор `break`; 4) ContVal — был встречен оператор `continue`; 5) RetVal(v) — был встречен оператор `return` и v — возвращаемое значение; 6) CaseVal(c) — управление передано на метку `case` c`:` в операторе `switch`; 7) DefVal — управление передано на метку `default:` в операторе `switch`; 8) Fail — ошибка в программе.

В разд. 2.4 представлена динамическая часть абстрактной машины, задающая отношения перехода. Аксиомы и правила разбиты на три группы: вычисление выражений (п. 2.4.1), обработка деклараций (п. 2.4.2), исполнение операторов (п. 2.4.3). В качестве примера типичного правила COC выбрано правило для присваивания:

$$\frac{\begin{array}{cc} \sigma_0 \models e_1 : \mathrm{lv}[\tau_1] & \tau_1 - \text{ не массив} \\ \langle e_2, \sigma_0, lf\rangle \to_e^* \langle \mathcal{E}, \sigma_1, lf\rangle & \langle \&e_1, \sigma_1, lf\rangle \to_e^* \langle \mathcal{E}, \sigma_2, lf\rangle \\ \mathrm{Val}_{\sigma_1} = (v_2, \tau_2) & \mathrm{Val}_{\sigma_2} = (c, \tau_1 *) \qquad IC(\tau_1, \tau_2) \end{array}}{\langle e_1 = e_2, \sigma_0, lf\rangle \to_e \langle \epsilon, \sigma_2'', lf\rangle},$$

где $\sigma_2'' = \sigma_2(\mathrm{MD} \leftarrow \mathrm{upd}(\mathrm{MD}, c, \gamma_{\tau_2,\tau_1}(v_2)))(\mathrm{Val} \leftarrow (\gamma_{\tau_2,\tau_1}(v_2), \tau_1))$. Т.е. вычисление присваивания разбито на следующие шаги: 1) с помощью системы типов проверяем, что e_1 — это модифицируемое l-значение; 2) вычисляем правую часть присваивания и запоминаем результат с его типом в Val; 3) вычисляем левую часть, одновременно определяя адрес объекта, обозначаемого выражением e_1; 4) определяем, связаны ли типы левого и правого подвыражений отношением приводимости; 5) значение e_2 приводим к типу левой части с помощью отображения $\gamma_{\tau_1,\tau 2}$; 6) это окончательное значение помещаем по адресу модифицируемого объекта и объявляем его значением всего выражения присваивания.

Наконец в п. 2.4.4 определяется семантика частичной корректности $\mathcal{M}$ и рассматриваются некоторые ее свойства. Отображение $\mathcal{M}$ строится на базе транзитивного рефлексивного замыкания отношения $\to_s$. По программе (фрагменту) и входному состоянию оно выдает множество заключительных состояний. При этом также происходит расширение класического метода. Входом отображения $\mathcal{M}$ может быть как допустимая программа, так и произвольный фрагмент. В случае фрагмента семантика определяется с точностью до уровня вложенности. Чтобы обрабатывать передачу управления во фрагменте, к нему добавляется специальная конструкция, связанная с обработкой блоков. Главными свойствами COC являются детерминизм и отсутствие блокировки.

В **главе 3** представлен язык C-kernel, определена интерпретация языка утверждений в виде денотационной семантики и описана аксио-

матическая система HSC для языка C-kernel.

Разд. 3.1 содержит описание языка C-kernel. Являясь ядром языка C-light, он отбрасывает конструкции, не поддающиеся формализации в логике Хоара. Любая декларация объявляет ровно один идентификатор, а спецификаторы static и auto обязательны.

При вычислении любого выражения в C-kernel допускается не более одного изменения содержимого памяти, т.е. побочные эффекты в выражениях единичны. Допустимыми операциями являются только следующие: первичные (`()`, `[]`, `"."`, `->`), унарные (`*`, `-`), бинарные (`*`, `/`, `%`, `+`, `-`, `<`, `>`, `<=`, `>=`, `==`, `!=`), простое присваивание, выделение/освобождение памяти, приведение типа.

Запрещены цепочки присваиваний вида `a=b=c`. В C-kernel нет операции «запятая», поэтому любой оператор вычисления выражения есть либо выражение присваивания, либо вызов процедуры. Фактическим аргументом функции может быть константа либо переменная.

В языке C-kernel разрешены следующие операторы: 1) оператор-выражение; 2) условный оператор if с обязательной ветвью else (возможно, пустой); 3) оператор цикла while, условием которого может быть только выражение без побочных эффектов; 4) оператор передачи управления goto; 5) оператор возврата значения return, причем выражением в нем может быть только константа или переменная; 6) блок.

В разд. 3.2 обосновывается связь операционной семантики языка C-light и аксиоматической семантики языка C-kernel. Эта связь задается посредством интерпретации языка утверждений с помощью состояний абстрактной машины, тем самым логические утверждения (пред-, постусловия и инварианты) задают множества состояний, в которых они истинны. При этом доказывается стандартная лемма о подстановке, связывающая обновления состояний и подстановки в утверждениях.

В разд. 3.3 представлена аксиоматическая система HSC для языка C-kernel. В системе HSC происходит вывод троек Хоара в окружении. *Окружение* $\mathcal{E}nv$ — это тройка: имя текущей функции, текущий уровень вложенности и множество гипотез, связанных с вызовами функций и передачей управления. В качестве примера приведено правило для присваивания переменной результата вызова функции.

$$\frac{\{P'\}\ \mathrm{f}(\overline{x})\ \{Q'\} \in \mathcal{E}nv \qquad \mathcal{E}nv \vdash P \Rightarrow (P'\alpha \wedge (Q'\gamma(\mathrm{Val} \leftarrow z)) \Rightarrow Q\gamma\beta\delta)}{\mathcal{E}nv \vdash \{P\}\ lval\ =\ \mathrm{f}(\overline{e});\ \{Q\}}\ .$$

Подстановки $\alpha, \beta, \gamma, \delta$ осуществляют такие действия, как отбрасывание

локальных объектов в точке вызова, замену формальных параметров фактическими аргументами и переименование метапеременных.

Выводимость тройки ϕ в системе HSC в окружении $\mathcal{E}nv$ обозначаем как $\mathcal{E}nv \vdash_{\text{HSC}} \phi$. Также в HSC вводится новое понятие частичной корректности. Тройка Хоара $\{P\}\ S\ \{Q\}$ *истинна в смысле частичной корректности в окружении* $\mathcal{E}nv$ (обозначение $\mathcal{E}nv \models \{P\}\ S\ \{Q\}$), если

1. $\forall\, \sigma$, если $\sigma \models P$ и $\langle S,\, \sigma, lf(\mathcal{E}nv)\rangle \to_s^* \langle \mathcal{E},\, \sigma', lf(\mathcal{E}nv)\rangle$, то $\sigma' \models Q$;
2. $\forall\, \mathrm{L} \in \mathrm{Labels(S)}\ \forall\, \sigma$ такого, что $\sigma \models \mathcal{I}nv(\mathrm{L})$, если

$$\langle S,\, \sigma(\mathrm{Val} \leftarrow \mathrm{GoVal(L)}), lf(\mathcal{E}nv)\rangle \to_s^* \langle \mathcal{E},\, \sigma', lf(\mathcal{E}nv)\rangle,$$

то $\sigma' \models \mathsf{Q}$;

Наконец в разд. 3.4 проводится доказательство основной теоремы о непротиворечивости системы HSC.

Теорема 1. Система вывода HSC непротиворечива для свойства частичной корректности, т.е. $\mathcal{E}nv \vdash_{\text{HSC}} \phi$ влечет $\mathcal{E}nv \models \phi$.

Глава 4 содержит описание системы правил перевода из языка C-light в язык C-kernel. Правила сгруппированы в соответствии с их предназначением: перевод деклараций (разд. 4.1), перевод операторов (разд. 4.2) и перевод выражений (разд. 4.3). В некоторых группах правила, в свою очередь, разбиты на правила элиминации, правила декомпозиции и правила нормализации. В качестве примера рассмотрим правило для перевода постфиксного инкремента:

RInc. Фрагмент вида

```
e++
```

заменяется на фрагмент

```
(q = &e, y = *q, *q = *q + 1, y)
```

где `q`, `y` — новые переменные, объявленные с типами выражений `&e` и `e`, соответственно.

Ряд свойств предложенной системы правил перевода рассмотрен в разд. 4.4. К этим свойствам относятся завершимость алгоритма перевода и нормализованность его выхода, т.е. принадлежность результирующе программы языку C-kernel.

Необходимые для обоснования корректности сведения рассмотрены в разд. 4.5. Они включают следующие определения. *Программа* на языке C-light — это либо обычный набор внешних деклараций, либо определение одной функции. Множеством *семантических* объектов программы S называется множество Obj(S) идентификаторов функций и всех

статических переменных программы. Множеством *семантических* объектов функции f называется множество $\mathsf{Obj(f)}$ идентификаторов автоматических и статических переменных функции f. *Атрибут* идентификатора id, принадлежащего множеству семантических объектов, — это либо пара $(\tau, \mathsf{storage})$, если id — переменная типа τ с классом памяти $\mathsf{storage}$, либо сигнатура функции, если id — имя функции.

Правила перевода вводят новые переменные, что позволяет ввести понятие объектного расширения. Программа (функция) T называется *объектным расширением* программы (функции) S *относительно* взаимно-однозначного всюду определенного отображения ϕ из $\mathsf{Obj(S)}$ в $\mathsf{Obj(T)}$, если для любого $\mathsf{id} \in \mathsf{Obj(S)}$ атрибут имени (id) совпадает с атрибутом для $\phi(\mathsf{id})$, т.е. множество объектов программы (функции) T получено переименованием объектов программы (функции) S и расширено новыми объектами. Обозначаем объектное расширение как $\mathsf{S} \lhd_\phi \mathsf{T}$.

Существование такого отображения еще не означает, что программы реализуют одну и ту же вычислимую функцию. Для этого требуется проверить совпадение значение объектов, связанных отображением ϕ. Программа T называется *семантическим расширением* программы S, если существует всюду определенное взаимно-однозначное отображение $\phi : \mathsf{Obj(S)} \to \mathsf{Obj(T)}$ такое, что

1) $\mathsf{S} \lhd_\phi \mathsf{T}$,
2) для любого начального состояния σ, для любого семантического объекта id его значение в заключительном состоянии $\mathcal{M}[\![S]\!](\sigma)$ совпадает со значением объекта $\phi(\mathsf{id})$ в состоянии $\mathcal{M}[\![T]\!](\sigma)$,
3) для любой пары функций $\mathsf{f_1} \in \mathsf{S}$ и $\mathsf{f_2} \in \mathsf{T}$ таких, что $\phi(\mathsf{f_1}) = \mathsf{f_2}$, существует всюду определенное взаимно-однозначное отображение $\phi_{(\mathsf{f_1}, \mathsf{f_2})} : \mathsf{Obj(f_1)} \cup \mathsf{Obj(S)} \to \mathsf{Obj(f_2)} \cup \mathsf{Obj(T)}$ такое, что свойства (1) и (2) рекурсивно выполнены для этих функций и расширенных множеств объектов.

Семантическое расширение обозначается как $\mathsf{S} \preccurlyeq_{\mathsf{op}} \mathsf{T}$. Это отношение рефлексивно и транзитивно (т.е. является предпорядком). Однако на статических объектах можно ввести эквивалентность. Обозначим множество статических объектов произвольной функции f как $\mathsf{Sobj(f)}$. Если в определении семантического расширения заменить множества $\mathsf{Obj(f_i)}$ на $\mathsf{Sobj(f_i)}$, то полученное отношение станет эквивалентностью. Обозначается *семантическая эквивалентность* как $\mathsf{S} \simeq_{\mathsf{op}} \mathsf{T}$.

Обоснование корректности системы правил перевода содержится в разд. 4.6. Основной результат формулируется в виде теоремы.

Теорема 2. Для любого правила перевода R и любой программы S на языке C-light $\mathsf{S} \preccurlyeq_{\mathsf{op}} \mathsf{R}(\mathsf{S})$.

Правила перевода не расширяют множество статических объектов программы, что позволяет доказать важное следствие.

Следствие. Для любой программы S на языке C-light и для любого правила перевода R из языка C-light в C-kernel $S \simeq_{op} R(S)$.

Глава 5 посвящена реализации системы HSC в виде прототипа генератора условий корректности. В разд. 5.1 рассмотрено понятие модифицированной семантики и приводится обобщенная схема автоматизированной системы верификации программ.

В разд. 5.2 рассмотрена формальная база для использования логической системы HSC для вывода условий корректности. Как и в обычной логике Хоара вывод в этой системе имеет недетерминированный характер, поскольку на каждом шаге применимо не единственное правило вывода. Решением является стратегия обратного просмотра, когда на правила вывода применяются для последней конструкции в программе, а вся предыдущая часть программы становится контекстом. В п. 5.2.1 рассмотрен входной язык генератора УК. Его независимая нотация позволяет задавать широкий класс императивных языков, в частности C-kernel и Pascal. В п. 5.2.2 вводится определение нормальной формы правил вывода. В п. 5.2.3 приводится схеме генератора УК в системе СПЕКТР-2.

Разд. 5.3 посвящен перспективному расширению метода генерации УК — метагенерации. Этот подход, в сочетании с независимым представлением, позволяет создать инструментарий для автоматизированного порождения генераторов УК по аксиоматической системе. Для этого вводятся понятия общей и нормальной формы правил вывода и описывается алгоритм перевода их общей формы в нормальную.

Приложение А содержит пример вывода условий корректности для рекурсивной процедуры обращения массива. Указаны 4 условия корректности для самой процедуры и условие корректности всей программы. Истинность условий была доказана вручную.

Приложение В описан эксперимент для прототипа метагенератора. Продемонстрирован фрагмент генератора условий корректности для части языка C-kernel, расширенной правилами элиминации инвариантов циклов для программ линейной алгебры. Генератор в виде рекурсивной ML-функции построен метагенератором.

Основные выводы и результаты. В рамках диссертации получены следующие результаты.

• Выделено представительное подмножество языка C, расширенное операциями для работы с динамической памятью и названное языком C-light. Для этого языка была создана структурная операционная семантика и семантика частичной корректности.

• Выделено ядро языка C-light — язык C-kernel и создана аксиоматическая семантика HSC для этого ядра. Введено новое понятие частичной корректности для троек Хоара в окружении. Доказана непротиворечивость системы HSC.

• Разработана система правил перевода из языка C-light в язык C-kernel. Предложено новое понятие семантического расширения. Доказана корректность системы правил перевода.

• Реализован прототип генератора условий корректности для языка C-kernel. Также предложен новый алгоритм метагенерации, способный порождать генераторы УК по исходной аксиоматической системе. Проведены успешные эксперименты с генератором УК.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C-программ. Язык C-light // Конференция посвященная 90-летию со дня рождения А.А. Ляпунова (пленарные доклады). — Новосибирск, 2001. — С. 423–432.

2. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C-программ. Часть 1. Язык C-light. — Новосибирск, 2001. — 48 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; N 84).

3. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C-программ. Часть 2. Язык C-light-kernel и его аксиоматическая семантика. — Новосибирск, 2001. — 58 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; N 87).

4. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C программ. Часть 3. Перевод из языка C-light в язык C-light-kernel и его формальное обоснование. — Новосибирск, 2002. — 82 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; N 97).

5. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации C программ. Язык C-light и его

формальная семантика. // Программирование. — 2002. — N 6. — С. 1–13.

6. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** На пути к верификации С программ. Аксиоматическая семантика языка C-kernel // Программирование. — 2003. — N 6. — С. 1–16.

7. **Промский А.В.** Операционная и аксиоматическая семантики языка Си // Четвертый сибирский конгресс по прикладной и индустриальной математике (ИНПРИМ-2000). Тезисы докладов, часть II. — Новосибирск: Изд-во Института математики СО РАН, 2000. — С. 125.

8. **Промский А.В.** Формальная семантика указателей языка C-light // Материалы конференции молодых ученых, посвященной 10-летию Института вычислительных технологий СО РАН, T.I. — Новосибирск, 2001. — С. 62–65.

9. **Промский А.В.** Автоматическая генерация условий корректности в системе верификации программ // Международная конференция молодых ученых по математическому моделированию и информатике. Тезисы докладов. — Новосибирск: Изд-во Института математики СО РАН, 2002. — С. 67.

10. **Промский А.В.** Генерация и метагенерация условий корректности в системе СПЕКТР-2. — Новосибирск, 2003. — 50 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; N 103).

11. **Nepomniaschy V. A., Anureev I. S., Promsky A. V.** Verification-oriented language C-light and its structural operational semantics (extended abstract) // 5th International A. Ershov conference “Perspectives of System Informatics” (PSI 2003). — Springer, Berlin, 2003. — P. 103–111. — (Lect. Notes Comput. Sci.; 2890).

Личный вклад автора. Все включенные в диссертацию результаты, касающиеся формализации семантик языков C-light и C-kernel, и доказательства двух основных теорем получены автором самостоятельно. Система правил перевода разработана автором в сотрудничестве с И.Н. Михайловым. Свойства завершимости алгоритма перевода и нормализации его выхода были доказаны И.С. Ануреевым. Автором самостоятельно проведены и описаны эксперименты с прототипом (мета)генератора в системе СПЕКТР-2.