На правах рукописи

МАРЬЯСОВ
Илья Владимирович

ВЕРИФИКАЦИЯ С-ПРОГРАММ С ПОМОЩЬЮ СМЕШАННОЙ АКСИОМАТИЧЕСКОЙ СЕМАНТИКИ

Специальность 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание учёной степени
кандидата физико-математических наук

Новосибирск — 2012

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

Научный руководитель:	Непомнящий Валерий Александрович, кандидат физико-математических наук
Официальные оппоненты:	Сафонов Владимир Олегович, доктор физико-математических наук, профессор
	Скопин Игорь Николаевич, кандидат физико-математических наук
Ведущая организация:	Ярославский государственный университет имени П. Г. Демидова

Защита состоится 14 мая 2012 г. в 16 ч 00 мин. на заседании диссертационного совета ДМ003.032.01 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 ИСИ СО РАН (пр. ак. Лаврентьева, 6).

Автореферат разослан 12 апреля 2012 г.

Учёный секретарь
диссертационного совета

к. ф.-м. н. Мурзин Ф. А.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность. Формальная верификация программ — актуальное направление современного программирования. Особый интерес представляет верификация программ, написанных на распространённых языках системного программирования таких, как C. Область применения этого языка весьма обширна: от программного обеспечения для бытовой техники до операционных систем и авионики.

Обозримая формальная семантика является необходимой предпосылкой того, что язык удобен для верификации. Значительный вклад в эту область внесли работы 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 формальной детерминированной семантики для языка C, полностью соответствующего стандарту ISO, не существует. Во-первых, сказывается заложенная в C возможность работать на низком уровне — обращение к памяти на уровне отдельных байтов и даже битов. Во-вторых, стандартизация C произошла намного позже его широкого распространения, поэтому многие аспекты поведения C-программ в стандарте ISO не специфицированы. В-третьих, сам стандарт написан на естественном языке, что влечёт за собой двусмысленности и неясности.

Актуальность формальной верификации подтверждается и современными работами зарубежных специалистов по верификации C программ, таких как Шармы, Дходапкара, Рамеша, Кларка, Кронинга, Хольцмана, Леруа. При общем сравнении известных проектов их можно разделить на два класса. К первой группе относятся проекты, ориентированные на максимальный охват языка C, но пригодные для поиска лишь некоторых классов ошибок, либо использующие трудно формализуемые методы, что усложняет доказательство их корректности. Во второй группе находятся проекты, использующие классические формальные подходы, но при этом исследователи вынуждены вводить ограничения на целевой язык.

В лаборатории теоретического программирования ИСИ СО РАН разработан язык C-light, являющийся представительным подмножеством языка C.

Формально этот язык определён с помощью структурной операционной семантики в стиле Плоткина. Хотя операционная семантика удобна для формального определения языка, она имеет слишком низкий уровень, что приводит к значительным трудностям при верификации. Поэтому обычно

используют аксиоматическую семантику, базирующуюся на логике Хоара, которая определяется как система вывода утверждений о свойствах программ. Однако аксиоматическая семантика для языка C-light была бы очень громоздкой.

В связи с этим был применён двухуровневый подход: в языке C-light выделяется подмножество — язык C-kernel, для которого разработана аксиоматическая семантика, и в этот язык транслируются исходные программы на языке C-light. По сравнению с языком C-light в языке C-kernel более простые выражения с минимальным числом побочных эффектов и ограниченный набор операторов. Это позволило упростить аксиоматическую семантику для языка C-kernel. При разработке метода верификации C-light программ была доказана теорема о непротиворечивости аксиоматической семантики языка C-kernel относительно операционной, а также описаны формальные правила перевода из языка C-light в язык C-kernel с обоснованием их корректности. Стоит отметить, что использование промежуточного этапа трансляции входных программ характерно для некоторых проектов указанных выше авторов. Но трансляция осуществляется в языки, отличные от C, что ставит под вопрос её корректность.

При верификации реальных программ (особенно среднего и большого объёма) процесс ручной генерации условий корректности очень трудоёмкий, поэтому его целесообразно автоматизировать. Предложенная ранее аксиоматическая семантика языка C-kernel удобна для теоретических исследований, но её применение к верификации реальных программ приводит к громоздким условиям корректности.

Цель и задачи диссертации. Целью диссертационной работы является разработка непротиворечивой формальной аксиоматической семантики языка C-light, позволяющей существенно упрощать как сами условия корректности, так и процесс их генерации. Для достижения цели были поставлены и решены следующие задачи:

1. Разработана смешанная аксиоматическая семантика языка C-kernel, позволяющая упрощать как сами условия корректности, так и процесс их генерации.
2. Разработана соответствующая смешанная операционная семантика языка C-light.
3. Разработан и обоснован алгоритм перевода инвариантов циклов при трансляции программ из C-light в C-kernel.
4. Доказана непротиворечивость смешанной аксиоматической семантики языка C-kernel относительно операционной семантики языка C-light.

5. Предложен набор примеров, на которых успешно продемонстрирован метод упрощения верификации C-light программ посредством использования смешанной аксиоматической семантики.

Методы исследования. В работе использовались методы математической логики, структурный операционный подход Плоткина, аксиоматический метода Хоара.

Научная и практическая ценность. Полученные результаты являются теоретической основой для разработки генератора условий корректности C-light программ, являющегося составной частью системы верификации СПЕКТР-2, разрабатываемой в лаборатории теоретического программирования ИСИ СО РАН.

Доклады и печатные публикации. Основные результаты работы были представлены на 8-й международной конференции «Perspectives of System Informatics» (Новосибирск, 2011), на международном семинаре «Program Semantics, Specification and Verification: Theory and Applications» в рамках «5th International Computer Science Symposium in Russia» (Казань, 2010), на международном семинаре «Program Understanding» (Алтай, 2009 г.), проводившемся в рамках 7-й международной конференции «Perspectives of System Informatics», VIII Всероссийской конференции молодых учёных по математическому моделированию и информационным технологиям (Новосибирск, 2007 г.) и на конференции-конкурсе «Технологии Microsoft в теории и практике программирования» (Новосибирск, 2007 г.).

Полученные результаты обсуждались на семинаре «Теоретическое и экспериментальное программирование» ИСИ СО РАН.

По материалам диссертации опубликовано 11 работ, включая 2 работы в журнале из списка ВАК.

Структура и объём диссертации. Диссертация состоит из введения, пяти глав и заключения. Объём работы составляет 110 страниц в формате машинописного текста. Список литературы содержит 66 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Во **введении** даётся обоснование актуальности темы диссертации, формулируются её цели и задачи, характеризуется практическая ценность.

В **главе 1** даются предварительные понятия, используемые в работе.

В разд. 1.1 дано определение языка C-light. Подробно рассматриваются допустимые типы, декларации, выражения, операторы, а так же накладываемые на них ограничения по сравнению со стандартом ISO C99.

В языке C-light есть пустой, целочисленные, вещественные типы, перечисления. Из составных типов присутствуют указатели, массивы, структуры и функции. Комплексные типы и объединения не допускаются. Имеются некоторые дополнительные ограничения, среди которых запрет неполных типов массивов (кроме случая использования в аргументах функций), битовых полей и функций с переменным числом аргументов.

Неокончательные определения запрещены. Не допускаются абстрактные декларации аргументов. Использование спецификатора **static** в размере массива, являющегося параметром функции, запрещено. Не допускается использование любых спецификаторов и модификаторов типов, кроме класса памяти, наличия знака и размера для скалярных типов. Имена всех статических объектов в программе уникальны.

В языке C-light строго фиксирован порядок вычисления выражений: аргументы операций и функций вычисляются справа налево, списки инициализирующих выражений слева направо.

Составные литералы в C-light разрешены только в качестве инициализаторов в декларациях. Операция приведения типа для составного инициализатора запрещена.

Приведения типов указателей ограничены случаем приведения от типа **void*** к произвольному τ.

На операторы накладывается два ограничения: все **case**-метки и метка **default** в операторе **switch** должны находиться на одном уровне вложенности; запрещён переход по оператору **goto** внутрь блока.

Исходный текст программы есть последовательность внешних деклараций. На внешнем уровне можно объявить тип (**typedef**-декларации), функцию (прототип или определение), структуру и данные.

В языке C-light препроцессор отсутствует, поскольку исходная программа препроцессируется до начала её верификации. Модульность не поддерживается, поэтому с точки зрения семантики любая исходная программа состоит из одного файла.

Язык C-kernel рассмотрен в разд. 1.2: описываются различия по сравнению с языком C-light.

Язык C-kernel — это промежуточный язык двухуровневой схемы верификации программ, в который транслируются программы на языке C-light.

Число побочных эффектов в выражениях языка C-kernel сведено до минимума, а операции, содержащие контрольные точки (например, логические && и ||), отсутствуют.

Нормализованным выражением называется выражение, не содержащее условных операций, операций последовательного вычисления, логических операций && и ||, простых и составных присваиваний, операций инкремента и декремента.

Пусть e (возможно с индексами) обозначает нормализованное выражение, τ — его тип. Оператор-выражение языка C-kernel имеет вид:

- e = e';
- e'(e_1, ..., e_n);
- e = e'(e_1, ..., e_n).

Списки декларируемых объектов разрешены только в декларациях функций, любая другая декларация объявляет ровно один объект. Декларации находятся в нормальной форме и содержат в инициализирующей части только нормализованные выражения. Спецификаторы класса памяти **static** и **auto** обязательны, других спецификаторов класса памяти не допускается.

В языке C-kernel допустимыми являются следующие операторы:

1. Оператор вычисления выражения (возможно, пустой).
2. Условный оператор **if** с обязательной ветвью **else** (возможно, пустой) и нормализованным условием.
3. Оператор цикла **while** с нормализованным условием.
4. Оператор передачи управления **goto** с теми же ограничениями, что и в C-light.
5. Оператор возврата значения **return**, причём возвращаемым выражением может быть только нормализованное выражение.
6. Составной оператор (блок).

Разд. 1.3 подробно описывает язык утверждений, используемый для спецификации программ, его типы и алфавит (п. 1.3.1), выражения (п. 1.3.2), в п. 1.3.3 вводится понятие подстановки и формулируется лемма о подстановке. П. 1.3.4 даёт определение интерпретации выражений, необходимое для формального определения операционной семантики языка C-light. В п. 1.3.5 вводится понятие истинности логического утверждения, формулируется лемма о свойствах стандартных булевских операций и лемма о свойствах подстановки.

Глава 2 представляет разработанную операционную семантику языка C-light. Данная семантика рассматривает процесс выполнения программы в терминах изменения состояний абстрактной машины. Её определение вводится в разд. 2.1. Прежде всего даётся определение состояния:

Состояние абстрактной машины языка C-light — это отображение, означивающее следующие переменные:

1. MD — переменная типа Locations → CTypes (где Locations — множество адресов, CTypes — объединение всех допустимых типов языка C-light) определяет значения, хранимые в памяти.
2. Val — переменная типа CTypes × LogTypeSpecs, где LogTypeSpecs — множество логических имён типов данных языка C-light. В этой переменной хранится значение вычисленного выражения и его тип.
3. Переменные типа Names, к значениям которых нет доступа через указатели (далее такие переменные называем неразделяемыми, остальные переменные — разделяемыми).

Для описания работы абстрактной машины языка C-light требуются специальные функции, называемые абстрактными. Их определение даётся в разд. 2.2, перечислим основные:

1. UnOpSem(•, v, τ) — функция, которая возвращает результат применения унарной операции • к значению v типа τ.
2. BinOpSem(•, v_1, τ_1, v_2, τ_2) — функция, которая возвращает результат применения бинарной операции • к значениям v_1 и v_2 типов τ_1 и τ_2 соответственно.
3. mb(e, id) — функция, вычисляющая адрес элемента массива e[id] или поля структуры e.id.
4. Функция upd(A, e, c) осуществляет обновление отображения A в точке e на значение c и определяется стандартным образом.
5. Функция retype(τ) получает в качестве аргумента логический тип τ и возвращает тип τ_0, если τ имеет вид $\tau_1 \times \ldots \times \tau_n \to \tau_0$ и τ в противном случае.
6. Семейство функций type(n) возвращает тип конструкции n (константы, переменной, выражения) языка C-light.
7. Функция addr(e, MD) вычисляет адрес выражения e в соответствии со значением MD.

8. Функция val(e, MD) возвращает значение выражения e в соответствии со значением MD.

9. Функция cast(e, τ, τ') преобразует значение e типа τ к значению типа τ'.

Разд. 2.3 посвящён аксиомам и правила вывода операционной семантики C-light. Вначале даётся определение конфигурации абстрактной машины — это пара $\langle P, \sigma \rangle$: программа P и состояние σ.

Произвольная аксиома операционной семантики имеет вид

$$\langle A, \sigma \rangle \to \langle B, \sigma' \rangle.$$

Эта запись означает, что один шаг исполнения программного фрагмента A, начинающийся в состоянии σ, приводит в состояние σ' и B — тот фрагмент исходной программы, который остаётся для исполнения. Произвольное правило семантики имеет вид

$$\frac{P_1, \ldots, P_n}{\langle A, \sigma \rangle \to \langle B, \sigma' \rangle}$$

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

Таким образом, исполнение программ в операционной семантике приводит к цепочками конфигураций, которые в общем случае могут быть и бесконечными:

$$\langle S_1, \sigma_1 \rangle \to \langle S_2, \sigma_2 \rangle \to \ldots \to \langle S_i, \sigma_i \rangle \to \ldots$$

Конфигурация $\langle S_1, \sigma_1 \rangle$ называется начальной. Если же цепочка конечна и $\langle S_n, \sigma_n \rangle$ — последняя конфигурация в этой цепочке, то говорим, что исполнение фрагмента S_1, начинающееся в состоянии σ_1 приводит в заключительную конфигурацию $\langle S_n, \sigma_n \rangle$. Если обозначить транзитивное рефлексивное замыкание отношения $\to$ как $\to^*$, то в общем случае такое исполнение можно обозначить как

$$\langle S_1, \sigma_1 \rangle \to^* \langle S_n, \sigma_n \rangle.$$

Пустой фрагмент будем обозначать символом ε. Пустым фрагментом может быть как пустая программа, так и пустое выражение.

В качестве примера типичного правила вывода модифицированной операционной семантики языка C-light приведём правило вывода для операции присваивания. Пусть τ'' — тип, отличный от массива, а e — разделяемая переменная. Правило для простого присваивания имеет вид

$$\frac{\langle e_0, \sigma\rangle \to^* \langle \mathsf{Val}(v', \tau'), \sigma'\rangle, \langle e, \sigma'\rangle \to^* \langle \mathsf{Val}(v'', \tau''), \sigma''\rangle, \quad v = \mathsf{cast}(v', \tau', \tau''), \ c = \mathsf{addr}(v'')}{\langle e = e_0, \sigma\rangle \to \langle \mathsf{Val}(v, \tau''), \sigma''(\mathsf{MD} \leftarrow \mathsf{upd}(\mathsf{MD}, c, v))\rangle}$$

В случае, когда e — неразделяемая переменная, правило имеет вид

$$\frac{\langle e_0, \sigma\rangle \to^* \langle \mathsf{Val}(v', \tau'), \sigma'\rangle, \langle e, \sigma'\rangle \to^* \langle \mathsf{Val}(e, \tau''), \sigma''\rangle, \ v = \mathsf{cast}(v', \tau', \tau'')}{\langle e = e_0, \sigma\rangle \to \langle \mathsf{Val}(v, \tau''), \sigma''(e \leftarrow v)\rangle}$$

Вычисление присваивания состоит из следующих этапов: вычисление правой части e_0 и сохранение результата (значения и его типа) в метапеременной Val; вычисление левой части e; вычисление адреса выражения левой части присваивания c; приведение типа значения правой части τ' присваивания к типу значения левой части τ''. Полученное значение объявляется результатом операции присваивания, при этом состоянием абстрактной машины объявляется состояние σ'', в которое перешла машина после вычисления левой части, модифицированное в точке MD(c).

Случай неразделяемой переменной, находящейся в левой части присваивания, отличается тем, что нет необходимости вычислять её адрес.

На основе введённых в предыдущих разделах понятий в разд. 2.4 даётся определение семантики частичной корректности. Семантика программы S есть отображение из множества состояний абстрактной машины в множество подмножеств этих состояний, т. е. семантика рассматривает все возможные исполнения программы S, начинающиеся в состоянии σ, при которых программа завершается нормальным образом:

$$\mathcal{M}[[S]](\sigma) = \{\sigma' \mid \langle S, \sigma\rangle \to^* \langle \varepsilon, \sigma'\rangle\} \cup \{\sigma'(\mathsf{Val} \leftarrow (v, \tau)) \mid \langle S, \sigma\rangle \to^* \langle \mathsf{Val}(v, \tau), \sigma'\rangle\}.$$

Отображение $\mathcal{M}$ называется семантикой частичной корректности для языка C-light.

Разд. 2.5 посвящён трансляции программ из C-light в C-kernel. В п. 2.5.1 рассмотрены правила для операторов и деклараций, п. 2.5.2 даёт правила для выражений. В заключение раздела приводится теорема о корректности и завершимости процесса перевода.

В главе 3 описана смешанная аксиоматическая семантика языка C-kernel. В разд. 3.1 вводятся необходимые понятия. Для формализации операторов перехода и функций вводится понятие окружения — это пятёрка (f, τ, B, bid, lab), где f — имя текущей функции, τ — тип значения, возвращаемого этой функцией, B — её тело, bid — уникальный идентификатор текущего обрабатываемого блока, lab — метка, на которую осуществляется переход по встреченному в процессе вывода оператору goto, либо идентификатор блока тела функции, иначе lab = ω. Функция main является текущей функцией не только для программных конструкций, входящих в её тело, но и для всей программы, поскольку в силу специфики этой функции выполнение программы начинается с её вызова.

Информация о пред- и постусловиях функции и об инвариантах помеченных операторов задаётся спецификацией $SP = (SP_{fun}, SP_{lab})$, включающей спецификацию функций SP_{fun} и спецификацию меток SP_{lab}.

Спецификация функций SP_{fun} — это пара (SP_{pre}, SP_{post}) отображений SP_{pre} и SP_{post}, называемых спецификациями пред- и постусловий функций соответственно.

Спецификация предусловий функции SP_{pre} — это отображение из имён функций f в функции P_f от метапеременных. Утверждение $P_f(MD)$ называется предусловием функции f.

Спецификация постусловий функции SP_{post} — это отображение из имён функций f в функции Q_f от метапеременных. Утверждение $Q_f(MD)$ называется постусловием функции f.

Спецификация меток SP_{lab} — это отображение из меток в утверждения. Утверждение $SP_{lab}(L)$ называется инвариантом метки L.

Пусть символы P, Q обозначают аннотации, S — последовательность операторов, окружение E имеет вид (f, τ, B, cur, ω).

Теперь мы можем определить смешанную аксиоматическую семантику MHSC языка C-kernel как исчисление троек Хоара с окружениями. Выводимость тройки {P} S {Q} в системе MHSC в окружении E относительно спецификации SP обозначим как $E, SP \vdash_{MHSC} \{P\}\ S\ \{Q\}$. Термин «смешанная» означает то, что правила вывода с участием неразделяемых переменных могут иметь специальный (более простой) вид.

Для получения однозначности вывода условий корректности применяется подход прямого прослеживания, когда вывод проводится по программе от начала к концу (т. е. слева направо), и при этом преобразуется предусловие программы посредством последовательной элиминации самых левых операторов.

Символы ' и " стоящие рядом с именем переменной означают введение новой переменной (т. е. не встречавшейся в аннотациях до применения правила, в котором она вводится).

Далее подробно рассмотрены правила вывода для выражений (разд. 3.2), деклараций (разд. 3.3), операторов (разд. 3.4) и другие (разд. 3.5). В качестве примера рассмотрим правило вывода для операции присваивания. Пусть выражение e_0 не содержит вызовов функций и операторов приведения.

$$\frac{E, SP \vdash \{\exists MD'\ P(MD \leftarrow MD') \wedge MD = upd(MD', addr(val(e, MD'), MD'), cast(val(e_0, MD'), type(e_0), type(e)))\}\ A;\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{e = e_0;}\ A;\ \{Q\}}$$

где e является разделяемой переменной.

Правило представляет собой модификацию правила классической системы Хоара: подстановка осуществляется на метапеременной MD, и происходит приведение типа выражения e_0 к типу e.

В случае, когда e — простая неразделяемая переменная, правило имеет вид:

$$\frac{E, SP \vdash \{\exists e'\ P(e \leftarrow e') \wedge e = cast(val(e_0(e \leftarrow e')), type(e_0), type(e))\}\ A;\ \{Q\}}{E, SP \vdash \{P\}\ \mathbf{e = e_0;}\ A;\ \{Q\}}$$

При переводе программы на языке C-light в программу на языке C-kernel возникает проблема вычисления инвариантов циклов и меток полученной программы: при элиминации операторов **break** и **continue** внутри циклов и оператора **switch**, а также самого оператора **switch** возникают новые метки, каждой из которых в соответствии с правилами смешанной аксиоматической семантики языка C-kernel для помеченного оператора должен быть сопоставлен некоторый инвариант, который отсутствует в исходной программе. Решение данной проблемы рассмотрено в разд. 3.6. Если

при переписывании местоположение инварианта цикла в программе не меняется, то сам инвариант так же не изменяется. Для остальных случаев, а так же для случая появления новых меток применяется следующий алгоритм. В процессе вывода условий корректности неизвестные варианты обозначаются как переменные в логических формулах, затем составляется система, состоящая из всех условий корректности, где есть неизвестные инварианты. Потребовав одновременную истинность всех таких условий корректности, вычисляем неизвестные инварианты. После подстановки найденных значений получаем условия корректности без неизвестных подформул.

Применение данного подхода проиллюстрировано примером.

Глава 4 посвящена обоснованию корректности смешанной аксиоматической семантики относительно операционной: сформулирована и доказана теорема о непротиворечивости.

В разд. 4.1 вводятся необходимые дополнительные понятия. Для программы prg истинность тройки Хоара {P} prg {Q} определяется следующим образом: тройка Хоара {P} prg {Q} истинна в смысле частичной корректности (обозначение $\models$ {P} prg {Q}), если $\mathcal{M}$[[prg]](||P||) $\subseteq$ ||Q|| (то есть если утверждение Q истинно во всех заключительных состояниях всех возможных исполнений программы prg, начавшихся в тех состояниях, в которых истинно P).

Семантика частичной корректности относительно спецификации SP определяется для произвольной конструкции S языка C-light как отображение $\mathcal{M}_{SP}$[[S]] из States в 2^{States}:

$$\mathcal{M}_{SP}[[S]](\sigma) = \{\sigma' \mid \langle S\ gotoStop(S), \sigma\rangle \to^* \langle \varepsilon, \sigma'\rangle\} \cup$$
$$\{\sigma'(Val \leftarrow (v, \tau)) \mid \langle S\ gotoStop(S), \sigma\rangle \to^* \langle Val(v, \tau), \sigma'\rangle\} \cup$$
$$\{\sigma' \mid \langle gotoStart(L)\ S\ gotoStop(S), \sigma''\rangle \to^* \langle \varepsilon, \sigma'\rangle \text{ для некоторой метки } L,$$
$$\text{входящей в } S, \text{ и состояния } \sigma'', \text{ такого что } \sigma'' \models SP_{lab}\}$$

То есть семантика рассматривает все возможные исполнения программной конструкции S, при которых эта конструкция завершается нормальным образом, при условии, что выполнение конструкции начинается либо в состоянии σ, либо в состоянии, в котором происходит переход на эту конструкцию по оператору goto L. В последнем случае состояние должно удовлетворять инварианту $SP_{lab}(L)$ метки L. Добавление конструкции

gotoStop(S), использующейся в операционной семантике для обработки оператора goto позволяет учесть циклы, образованные с его помощью.

Для доказательства, что спецификации $SP_{lab}(L)$ действительно является инвариантами меток, нам потребуется ещё один вид семантики. Семантика выхода по метке L относительно спецификации SP определяется для произвольной конструкции S языка C-light как отображение $\mathcal{M}^{L}_{SP}[[S]]$ из States в 2^{States}:

$$\mathcal{M}^{L}_{SP}[[S]](\sigma) = \{\sigma' \mid \langle S\ gotoStop(S), \sigma\rangle \to^{*} \langle gotoStart(L), \sigma'\rangle \text{ для некоторой метки } L\} \cup$$
$$\{\sigma' \mid \langle gotoStart(L')\ S\ gotoStop(S), \sigma''\rangle \to^{*} \langle gotoStart(L), \sigma'\rangle \text{ для некоторых меток } L' \text{ и состояния } \sigma'', \text{ таких что } L' \text{ входит в } S \text{ и } \sigma'' \models SP_{lab}\}$$

То есть семантика рассматривает все возможные исполнения программной конструкции S, при которых эта конструкция завершается переходом на метку L при условии, что выполнение конструкции начинается либо в состоянии σ, либо в состоянии, в котором происходит переход на эту конструкцию по оператору goto L. В последнем случае состояние должно удовлетворять инварианту $SP_{lab}(L)$ метки L. Добавление конструкции gotoStop(S) позволяет учесть циклы, образованные с помощью оператора goto.

Пусть окружение E имеет вид $(f, \tau_f, \{S\}, bid, lab)$. Пусть Dom(E) обозначает формулу

$$\begin{cases} retType(type(\&f)) = \tau_f \wedge snd(MD(\&f)) = S, \text{ если } f \neq main, \\ true, \text{ в противном случае.} \end{cases}$$

Тройка Хоара {P} S {Q} истинна в смысле частичной корректности в окружении E относительно спецификации SP (обозначим этот факт $E, SP \models \{P\}\ S\ \{Q\}$), если

- $\models P \Rightarrow Dom(E)$;
- $\mathcal{M}[[S]](||P||) \subseteq ||Q||$.

Разд. 4.2 посвящён доказательству теоремы о непротиворечивости смешанной аксиоматической семантики.

Теорема 1. Система вывода MHSC непротиворечива для свойства частичной корректности, т. е. $E \vdash_{MHSC} \{P\}\ prg\ \{Q\}$ влечёт $E \models \{P\}\ prg\ \{Q\}$.

Идея доказательства состоит в подробном рассмотрении всех правил вывода с помощью индукции по размеру программного фрагмента.

Заключительная глава 5 иллюстрирует применение смешанной операционной семантики. Рассмотрена верификация программ вычисления факториала (разд. 5.1), поиска в линейном односвязном списке (разд. 5.2) и топологической сортировки (разд. 5.3).

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

1. Разработана смешанная аксиоматическая семантика языка C-kernel, позволяющая автоматически получать условия корректности и упрощать их.
2. Разработана новая версия операционной семантики языка C-light.
3. Описан и обоснован алгоритм перевода инвариантов циклов при трансляции программ из C-light в C-kernel.
4. Сформулирована и доказана теорема о непротиворечивости смешанной аксиоматической семантики языка C-kernel.
5. Разработан набор примеров с целью использования предложенной смешанной аксиоматической семантики C-kernel для упрощения верификации.

В настоящее время в лаборатории теоретического программирования ведётся разработка мультиязыковой автоматизированной системы верификации СПЕКТР-2 [7], которая поддерживает и язык C-light. Предложенная в настоящей работе смешанная аксиоматическая семантика языка C-kernel является основой автоматического генератора условий корректности — одного из модулей системы СПЕКТР-2.

Первые эксперименты по верификации в данной системе подтвердили полученные теоретические результаты.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. **Ануреев И. С., Марьясов И. В., Непомнящий В. А.** Верификация С-программ на основе смешанной аксиоматической семантики. // Моделирование и анализ информационных систем. — 2010. — Т. 17. — № 3. — С. 5 — 28. — URL: http://mais.uniyar.ac.ru/sites/default/files/journal/private/17_3__5-28.pdf.

2. **Марьясов И. В.** Автоматическая верификация программ на языке C-light. // Тезисы докладов VIII Всероссийской конференции молодых учёных по математическому моделированию и информационным технологиям. — Новосибирск, 2007. — С. 103.

3. **Марьясов И. В.** Автоматическая верификация программ на языке C-light. // Тезисы докладов конференции-конкурса «Технологии Microsoft в теории и практике программирования». — Новосибирск, 2007. — С. 25 — 27.

4. **Марьясов И. В.** Автоматическая генерация условий корректности в системе верификации C-light программ. // Материалы XLIV Международной научной студенческой конференции «Студент и научно-технический прогресс». — Новосибирск, 2006. — С. 253 — 254.

5. **Марьясов И. В.** На пути к автоматической верификации C-light программ. Смешанная аксиоматическая семантика языка C-kernel. — Новосибирск, 2008. — 32 с. — (Препринт / РАН Сибирское отделение. ИСИ; № 150). — URL: http://www.iis.nsk.su/files/preprints/150.pdf.

6. **Марьясов И. В.** Применение смешанной аксиоматической семантики языка C-kernel к верификации программы топологической сортировки. — Новосибирск, 2010. — 36 с. — (Препринт / РАН Сибирское отделение. ИСИ; № 155). — URL: http://www.iis.nsk.su/files/preprints/155.pdf.

7. **Непомнящий В. А., Ануреев И. С., Атучин М. М., Марьясов И. В., Петров А. А., Промский А. В.** Верификация С-программ в мультиязыковой системе СПЕКТР. // Моделирование и анализ информационных систем. — 2010. — Т. 17. — № 4. — С. 88 — 100. — URL: http://mais.uniyar.ac.ru/sites/default/files/journal/private/17_4__88-100.pdf.

8. **Непомнящий В. А., Атучин М. М., Марьясов И. В., Петров А. А., Промский А. В.** Система анализа и верификации С-программ СПЕКТР-2. // Труды семинара «Program Semantics, Specification and Verification». — 5th International Computer Science Symposium in Russia. — Казань, 2010. — С. 76 — 81.

9. **Anureev I., Maryasov I., Nepomniaschy V.** The Mixed Axiomatic Semantics Method for C-program Verification. // Ershov Informatics Conference: PSI Series, 8th Edition (Preliminary Proceedings). — Novosibirsk: A. P. Ershov Institute of Informatics Systems, 2011. — P. 261 — 266.

10. **Maryasov I. V.** The mixed axiomatic semantics method. — Novosibirsk, 2011. — 43 p. — (Preprint / Siberian Division of RAS. IIS; # 160). — URL: http://www.iis.nsk.su/files/preprints/160.pdf.

11. **Maryasov I. V.** Towards automatic verification of C-light programs. Mixed axiomatic semantics of C-kernel language. // Perspectives of Systems In-

formatics (PSI): A. Ershov 7th Int. Conf.: Int. workshop on Program Understanding. — Novosibirsk, 2009. — P. 44 — 52.

Личный вклад автора. Описанные в диссертации модифицированная операционная семантика языка C-light и смешанная аксиоматическая семантика языка C-kernel разработаны автором самостоятельно на основе работ В. А. Непомнящего, И. С. Ануреева и А. В. Промского по языкам C-light и C-kernel. Доказательство теоремы о непротиворечивости и верификация приведённых в работе примеров проведены автором самостоятельно.

Марьясов И. В.

ВЕРИФИКАЦИЯ С-ПРОГРАММ С ПОМОЩЬЮ СМЕШАННОЙ АКСИОМАТИЧЕСКОЙ СЕМАНТИКИ

Автореферат

Подписано в печать 09.04.2012 Объём 1,13 уч.-изд. л.

Формат бумаги 60 × 90 1/16 Тираж 100 экз.

Отпечатано в ЗАО РИЦ «Прайс-курьер»

630128, г. Новосибирск, ул. Кутателадзе, 4г, 310 к., тел. (383) 330-72-02

Заказ № __ 167