Модел. и анализ информ. систем. Т. **17**, № **4** (2010) 88–100

УДК 519.681

Верификация С-программ в мультиязыковой системе СПЕКТР

Непомнящий В.А., Ануреев И.С.[1], Атучин М.М.,
Марьясов И.В., Петров А.А., Промский А.В.[2]

Институт систем информатики имени А.П. Ершова СО РАН
Новосибирский государственный университет

e-mail: {vnep, anureev, promsky}@iis.nsk.su

получена 18 октября 2010

Ключевые слова: верификация, спецификация, операционная семантика, аксиоматическая семантика, предметно-ориентированные языки, системы верификации

Представлена расширяемая мультиязыковая система анализа и верификации СПЕКТР, разрабатываемая в рамках одноименного проекта, и показаны перспективы ее использования на примере верификации С-программ. Проект СПЕКТР направлен на создание нового интегрированного подхода к верификации императивных программ, который позволяет интегрировать, унифицировать и комбинировать методы и техники верификации императивных программ, накапливать и использовать знания о них. Особенностью подхода является использование специализированного языка выполнимых спецификаций Atoment для разработки средств верификации программ, который позволяет представить в едином унифицированном формате как методы и техники верификации, так и данные для них (программные модели, аннотации, логические формулы). С-компонента системы СПЕКТР использует двухуровневый метод верификации С-программ. Этот метод является хорошей иллюстрацией интегрированного подхода, поскольку он обеспечивает комплексную верификацию С-программ, базирующуюся на комбинации операционного, аксиоматического и трансформационного подходов.

1. Введение

Современная тенденция в области верификации программ — переход от разработки методов верификации программ, применяемых для небольших программ на модельных языках программирования, к верификации больших программных систем на

[1]Работа частично поддержана интеграционной программой РАН 2/12.

[2]Работа частично поддержана молодежным грантом Президиума СО РАН «Верификация программ на языке С с эффективной локализацией ошибок»

индустриальных языках программирования. Эта тенденция состоит в выделении практически значимых свойств программ и построении специализированных методов и техник анализа и верификации. Унификация и формализация процессов описания таких свойств и построения для них методов и техник является важной открытой проблемой. Для индустриальной верификации также характерно использование комбинации различных методов верификации. Это приводит к появлению новых гибридных методов верификации программ. Создание средств накопления, анализа и формализации опыта, накопленного в области интеграции различных методов верификации, — еще одна важная открытая проблема.

Цель проекта СПЕКТР — разработка нового подхода к верификации императивных программ, который позволяет интегрировать, унифицировать и комбинировать методы и техники верификации императивных программ, накапливать и использовать знания о них. На его основе предполагается создать мультиязыковую программную систему ускоренной разработки и апробации методов и техник верификации и их комбинаций.

Особенностью подхода является использование предметно-ориентированного языка (Domain Specific Language), предназначенного для разработки средств верификации программ, который позволит представить в едином унифицированном формате как методы и техники верификации, так и данные для них (программные модели, аннотации, логические формулы). Систему, базирующуюся на таком языке, можно рассматривать в качестве как специализированной среды разработки инструментов в области верификации программ, так и информационной системы, которая аккумулирует опыт в этой области, отчуждаемый в виде знаний, представленных на этом языке, и обеспечивает доступ к ним. В частности, знаниями, представленными в информационной системе, являются методы и техники верификации императивных программ.

Предполагается интегрировать в систему репрезентативный набор методов и техник верификации, базирующихся на операционном, аксиоматическом, трансформационном и логическом подходах и их комбинациях, и провести эксперименты по верификации программ для трех групп целевых языков — учебные языки, процедурные языки общего назначения и объектно-ориентированные языки.

В статье представлен прототип расширяемой мультиязыковой системы анализа и верификации программ СПЕКТР и показаны перспективы ее использования на примере верификации С-программ. С-компонента системы СПЕКТР использует двухуровневый метод верификации С-программ [2, 3]. Этот метод является хорошей иллюстрацией интегрированного подхода, поскольку он обеспечивает комплексную верификацию С-программ, базирующуюся на комбинации операционного, аксиоматического и трансформационного подходов.

Наша работа состоит из 6 разделов. В разд. 2 дается обзор двухуровневого подхода. Архитектура и принципы работы системы СПЕКТР представлены в разд. 3. В разд. 4 описано добавление новой языковой компоненты в систему на примере языка С. Разд. 5 посвящен экспериментам по верификации С-программ. Полученные результаты, их сравнение с другими известными работами и планы по развитию системы СПЕКТР рассмотрены в заключении.

2. Двухуровневый метод верификации C-программ

Двухуровневый метод верификации C-программ [2, 3] применяется к подмножеству C-light языка C, которое имеет формальную операционную семантику, описываемую C-light машиной, и покрывает существенную часть конструкций языка C. Аннотированная программа на языке C-light транслируется в промежуточный язык C-kernel, а затем для полученной программы порождаются условия корректности. Метод позволяет упростить процесс верификации, как за счет трансляции в промежуточный язык, так и за счет комбинации операционной и аксиоматической семантик при порождении условий корректности. Математический базис двухуровневого метода составляют теоремы о корректности перевода аннотированных C-light программ в C-kernel и о непротиворечивости аксиоматической семантики C-kernel относительно операционной.

Нормализованный вариант двухуровневого метода основан на предварительной нормализации C-light программ, которая представляет собой последовательность трансформаций C-light программ [5]. Применение двухуровневого метода к нормализованным C-light программам позволяет упростить C-light машину и, за счет этого, оптимизировать правила операционной семантики языка C-light и аксиоматической семантики языка C-kernel. Таким образом, нормализованный двухуровневый метод позволяет повысить эффективность доказательства корректности аннотированных программ за счет перехода от доказательства условий корректности к применению алгоритмов статического анализа и операционной семантики, используемых в трансформациях.

Примерами нормализующих трансформаций являются вычисление константных выражений (в частности, case-меток операторов switch) и добавление специальных атрибутных аннотаций к каждому вхождению оператора return. Вычисление константных выражений позволяет оптимизировать правила аксиоматической семантики. Атрибутные аннотации сопоставляются вершинам синтаксического дерева программы. Для оператора return атрибутная аннотация содержит информацию об имени функции, в которой встретился оператор return, типе возвращаемого значения, именах всех автоматических переменных, определенных в теле этой функции, включая формальные параметры, а также именах логических переменных, которые обозначают значения соответствующих автоматических переменных. Эта информация используется в аксиоматическом правиле для оператора return.

Еще одна модификация двухуровневого метода заключается в использовании смешанной аксиоматической семантики языка C-kernel [1]. Метод смешанной аксиоматической семантики использует несколько моделей памяти для C-программ в рамках одной аксиоматической семантики, обеспечивая для каждой модели свои варианты правил вывода. В частности, для переменных, значения которых в программе доступны только через имя переменной, используется "паскалевская" модель памяти "переменная — ее значение" и соответствующие ей упрощенные правила вывода. Метод позволяет упростить условия корректности за счет выбора и применения наиболее подходящего правила вывода.

Объектно-ориентированным расширением двухуровневого метода является трехуровневый метод верификации C#-программ [4]. Как и для двухуровневого метода,

выделяются языки C#-light и C#-kernel. Язык C#-light охватывает все конструкции C# за исключением конструкций, связанных с параллельным выполнением, атрибутами и небезопасным кодом. Язык C#-kernel представляет собой компактное подмножество языка C#, расширенное набором метаинструкций абстрактной машины, задающей операционные семантики для языков C#-light и C#-kernel. Аннотированные C#-light программы транслируются в C#-kernel программы и для последних порождаются условия корректности на основе аксиоматической семантики языка C#-kernel. Особенностью трехуровневого метода является то, что генерируемые условия корректности содержат нелогические фрагменты, обусловленные объектно-ориентированной спецификой языка C# и описывающие алгоритмы операционной семантики языка C# (например, фрагмент, вычисляющий динамический тип выражения). Поэтому они называются ленивыми условиями корректности. Дополнительный этап уточнения ленивых условий корректности разрешает эти фрагменты, базируясь на эвристическом алгоритме, комбинирующем алгоритмы операционной семантики с логическим выводом, который формирует входные данные для этих алгоритмов.

3. Архитектура и принципы работы системы

Система СПЕКТР является мультиязыковой системой анализа и верификации программ. В настоящее время она применяется для верификации С-программ, но открытая архитектура системы обеспечивает добавление новых языков программирования. Система (см. Рис. 1.) включает следующие компоненты: рабочее место верификатора (специалиста в области верификации программ), построитель программных моделей, анализатор программных моделей, интерпретатор трансформаций, менеджер доказательства, интерпретатор результатов анализа.

Основной цикл работы СПЕКТР состоит в последовательном выполнении заданий, поступающих от рабочего места верификатора, анализатором программных моделей (АПМ), и возвращении результатов анализа обратно на рабочее место верификатора.

Задание содержит аннотированную программу и спецификацию анализа на языке Atoment, определяющую применяемые к ней техники анализа и верификации. Перед входом в АПМ аннотированная программа преобразуется построителем программных моделей в программную модель (ПМ). ПМ является некоторым внутренним представлением аннотированной программы в системе СПЕКТР, доступ к которому осуществляется через конструкции языка Atoment. Использование ПМ позволяет унифицировать формат данных для АПМ. Аннотированные программы, использующие различные языки программирования и различные языки аннотации, приводятся к единому формату ПМ. Это обеспечивает мультиязыковость системы СПЕКТР. Для создания ПМ построитель ПМ обращается в репозиторий языковых адаптеров, выбирает адаптеры, соответствующие языку программирования и языку аннотаций, на которых написана аннотированная программа, и использует их для построения ПМ. Перед возвращением на рабочее место верификатора результаты анализа обрабатываются интерпретатором результатов анализа. В частности,

Рис. 1. Архитектура системы СПЕКТР

интерпретатор обеспечивает отображение конструкций ПМ в конструкции анализируемого аннотированного программного кода (аннотированной программы).

Логика работы АПМ основана на последовательных трансформациях ПМ. Для выполнения трансформаций АПМ использует интерпретатор трансформаций, подавая ему на вход ПМ и спецификацию трансформации (составную часть спецификации анализа) для нее. Интерпретатор трансформаций возвращает результат трансформации, который включает преобразованную ПМ, обратные зависимости и статус трансформации. Обратные зависимости используются интерпретатором результатов анализа, чтобы привести результаты анализа в соответствие с исходной постановкой задания. Статус трансформации принимает значения «сохраняет корректность», «усиливает корректность», «ослабляет корректность». Статус «сохраняет корректность» означает, что исходная ПМ корректна тогда и только тогда, когда корректна преобразованная ПМ. Статус «усиливает корректность» означает, что если преобразованная ПМ корректна, то корректна исходная ПМ. Статус «ослабляет корректность» означает, что если исходная ПМ корректна, то корректна преобразованная ПМ. Результаты трансформаций сохраняются АПМ в дерево трансформаций, вершинами которого являются ПМ, а с дугами связаны имя трансформации, обратные зависимости и статус трансформации. Дерево трансформаций является составной частью результата анализа ПМ.

Трансформации разбиваются на три основных типа: нормализующие преобразования, упрощающие преобразования, аксиоматические преобразования. Нормализующие преобразования приводят ПМ к некоторому каноническому виду. Примером нормализующей трансформации является алгоритм нормализации C-light программ. Упрощающие трансформации преобразуют ПМ к более простым ПМ относи-

тельно выбранных критериев упрощения. Примером упрощающей трансформации является трансляция C-light программ в C-kernel программу. С помощью аксиоматических преобразований реализуются различные аксиоматические семантики и стратегии их применения. В результате аксиоматических трансформаций ПМ, как правило, сводится к некоторому набору формул (которые также являются ПМ). Примером аксиоматических трансформаций является генерация условий корректности, основанная на смешанной аксиоматической семантике языка C-kernel или стратегии прямого прослеживания для базовой аксиоматической семантики языка C-kernel.

Трансформации также задаются на внутреннем языке Atoment системы СПЕКТР.

Для доказательства формул АПМ использует менеджер доказательства, подавая ему на вход формулы, появляющиеся в результате трансформаций. Менеджер возвращает результат доказательства формулы, включающий статус доказательства со значениями «истинна», «ложна», «не доказана» и, возможно, контрпример. Контрпример вместе с обратными зависимостями используется интерпретатором результатов анализа. С менеджером доказательства связан репозиторий адаптеров к решателям, которые используются в системе СПЕКТР. В текущей версии системы поддерживается адаптер к решателю Z3.

4. C-компонента системы

Рассмотрим, как добавляется новый язык программирования в систему СПЕКТР на примере языка C. Для добавляемого языка (или подъязыка) L требуется описать программную модель этого языка на языке Atoment, реализовать L-адаптер, который транслирует аннотированную программу на языке L в программную модель, и специфицировать на языке Atoment методы и техники верификации L-программ. Вместе они образуют L-компоненту системы СПЕКТР.

C-компонента системы СПЕКТР включает программную модель языка C, C-адаптер и спецификацию на языке Atoment следующих составляющих двухуровневого метода верификации C-программ: алгоритма валидации C-light программ, нормализующих трансформаций для C-light программ, алгоритма трансляции аннотированных программ на языке C-light в программы на языке C-kernel, метода смешанной аксиоматической семантики, упрощающих процедур для условий корректности.

Алгоритм валидации C-light программ проверяет, является ли исходная аннотированная C-программа аннотированной C-light программой. С помощью нормализующих трансформаций для C-light программ специфицируется нормализованный вариант двухуровневого метода.

Система СПЕКТР обеспечивает доказательство условий корректности, используя адаптеры к известным системам машинной поддержки доказательства. Однако, основываясь на предварительном анализе порожденных условий корректности и применяя специальные эвристики, можно существенно упростить условия корректности, прежде чем они будут переданы универсальной системе доказательства.

Такие упрощающие процедуры для условий корректности также специфицируются на языке Atoment. Для языка C используются упрощающие процедуры, описанные в [10].

Программные конструкции языков программирования описываются в программных моделях на языке Atoment с помощью элементов и их свойств. Так, условный оператор языка C описывается элементом со свойствами `if`, `then` и `else`. Значением первого свойства является программная модель для управляющего выражения условного оператора. Значениями второго и третьего свойств — программные модели для операторов ветвей then и else условного оператора соответственно. Кроме того, с каждой конструкцией связывается в ее модели ряд служебных свойств. В частности, эти свойства описывают позицию программной конструкции в исходном тексте программы и используются для обратного отображения конструкций программной модели в конструкции анализируемого аннотированного программного кода (аннотированной программы).

Трансформации, используемые в двухуровневом методе, описываются с помощью механизма сопоставления с образцом, который в языке Atoment определяется на свойствах элементов. Рассмотрим в качестве примера спецификацию на языке Atoment правила аксиоматической семантики для условного оператора (для простоты выбран условный оператор языка Паскаль):

```
(ifpattern:(applyAxiomaticRule:x) where:x.[HoareTriple] var:x then:
 (if:x match:(precondition:u programFragment:v postcondition:w)
  var:(u v w) then:
  (if:v
   match:(if:a then:b else:c) var:(a b c) then:
   (return:
    (new:
     ((precondition:(u and a) programFragment:b postcondition:w)
      (precondition:(u and (not:a)) programFragment:c postcondition:w))))
   match:... then:...
   ...)))
```

Действие applyAxiomaticRule применяет к тройкам Хоара правила аксиоматической семантики, реализуя стратегию прямого прослеживания. Действия определяются схемами выполнения элементов. Схема выполнения элементов — это элемент со свойствами ifpattern, var, where и then (есть еще ряд свойств, например уникальный идентификатор схемы id, но для данного примера они не существенны). Значением свойства ifpattern является образец, который определяет класс элементов, реализующих действие. Значением свойства var является множество переменных образца. Эти переменные являются параметрами действия и получают значения при сопоставлении элемента из класса с образцом. Значением свойства where является ограничение на переменные образца. Значением свойства then является элемент, реализующий действие. В данном случае, элемент, удовлетворяющий схеме, должен обладать свойством applyAxiomaticRule, значением которого является тройка Хоара (выполняется ограничение x.[HoareTriple] на переменную образца x). Свойство в

квадратных скобках является логическим свойством, принимающим значения true и false, а элемент вида A.B реализует действие "получить значение свойства B элемента A".

Условное действие вида if:A match:B var:C then:D else:E сводится к выполнению действия D′, если элемент A сопоставляется с образцом B, и к выполнению действия E в противном случае. Действие D′ есть результат замены в D переменных из C их значениями, полученными в результате сопоставления элемента A с образцом B. Свойство else у этого действия может отсутствовать, что равносильно выполнению пустого действия, которое не меняет состояния и возвращает пустой элемент () в качестве значения. Это условное действие имеет расширение, в котором допускается вхождение нескольких конструкций вида match:B var:C then:D. При этом выполняется then-часть первой конструкции в порядке их следования, с образцом которой сопоставился элемент A.

В нашем случае первое условное действие проверяет, является ли элемент x тройкой Хоара, т. е. имеет ли он свойства precondition, programFragment и postcondition. Заметим, что элемент x может обладать и другими свойствами, это не влияет на сопоставление с образцом. В случае, если x — тройка Хоара, значения предусловия, программного фрагмента и постусловия сохраняются в переменных u, v и w образца, и далее последовательно выполняются действия, сопоставляющие программный фрагмент v с образцами конструкций целевого языка программирования.

Второе условное действие проверяет, является ли программный фрагмент v условным оператором, т. е. обладает ли он свойствами if, then и else. Если это так, то действие applyAxiomaticRule возвращает в качестве значения две тройки Хоара. Элемент return A реализует действие "возвратить значение A". Первая тройка имеет в качестве предусловия конъюнкцию исходного предусловия u с управляющим выражением условного оператора a, а в качестве программного фрагмента — then-ветвь b условного оператора. Вторая тройка имеет в качестве предусловия конъюнкцию исходного предусловия u с отрицанием управляющего выражения условного оператора a, а в качестве программного фрагмента — else-ветвь c условного оператора.

Элемент new:A реализует действие "породить новый элемент в соответствии с образцом A". Образец может иметь вложенную структуру — содержать подобразцы. В этом случае для каждого подобразца также порождается новый элемент. Образец вида (A op B) является синтаксическим сокращением для (applyfun:op 1:A 2:B), позволяющим удобно представлять применение бинарных операций в инфиксной форме.

В данном случае порождается элемент со свойствами 1 и 2 (такие свойства называются порядковыми), значениями которых являются элементы D_1 и D_2, порождаемые подобразцами

```
(precondition:(u and a) programFragment:b postcondition:w)
```

и

```
(precondition:(u and (not:a)) programFragment:c postcondition:w).
```

Элемент $\mathtt{D}_1$ имеет свойства precondition, programFragment и postcondition со значениями $\mathtt{D}_3$, b и w. Порождаемый элемент $\mathtt{D}_3$ имеет свойства applyfun, 1 и 2 со значениями and, u и a.

Элемент $\mathtt{D}_2$ имеет свойства precondition, programFragment и postcondition со значениями $\mathtt{D}_4$, c и w. Порождаемый элемент $\mathtt{D}_4$ имеет свойство applyfun, 1 и 2 со значениями and, u и $\mathtt{D}_5$. Порождаемый элемент $\mathtt{D}_5$ имеет свойство not со значением a.

Правила аксиоматической семантики для других конструкций целевого языка программирования в стратегии прямого прослеживания представляются аналогичным образом.

5. Эксперименты

В качестве экспериментов были верифицированы некоторые примеры из известной коллекции Java-программ, верификация которых сталкивается со значительными трудностями [9]. Из коллекции были отобраны примеры, относящиеся к процедурному случаю. Проблемы, представленные в примерах, включают совмещение ссылок (aliasing), побочные эффекты в выражениях, передачу управления. Дополнительно рассмотрены пример поиска в линейном односвязном списке и пример для указателей на функции.

Рассмотрим более подробно некоторые проблемы верификации, затрагиваемые в указанных примерах.

Проблема совмещения ссылок возникает всякий раз, когда доступ к объекту в памяти возможен более чем через один указатель (ссылку). Очевидно, что изменение объекта при работе в аксиоматической семантике при этом должно учитывать все эти ссылки, что приводит как к сложным правилам, так и к громоздким условиям корректности. Работа же без учета всех связей приведет к серьезным проблемам с корректностью семантики. Скажем, тройка Хоара вида

$$\{x = 1\}\mathtt{y} = 2\{x = 1\}$$

истинная в классических работах (напр. для языка Паскаль), становится ложной в языке C++ при наличии объявления вида `int &y = x;`. Еще серьезней ситуация становится, если некоторые массивы в памяти располагаются «внахлест». В случае нашей семантики совмещение ссылок не является проблемой, поскольку мы вместо классической схемы доступа к объектам вида "имя→значение" используем явную схему "имя→адрес→значение". В результате нам не нужно в правиле для присваивания сравнивать один указатель со всеми остальными. Некоторым недостатком является разрастание условий корректности, но с этим призван бороться наш подход по смешанной аксиоматической семантике, при котором явная работа с адресами происходит только там, где это нужно, а во всех остальных случаях используются классические Хоаровские правила.

Передача управления также относится к проблемным особенностям с точки зрения логики Хоара. При этом обычный оператор неструктурированного перехода `goto` оказывается предпочтительней операторов `break` и `continue`. Дело в том, что

последние операторы сложно аксиоматизировать в обособленном виде, т.е. задать тройку Хоара вида $\{P\}$`break`;$\{Q\}$. Их либо нужно рассматривать в совокупности с охватывающими операторами цикла, что усложнит правила вывода, либо определять семантику протягивания до нужной точки, что увеличит количество определений корректности. В нашем подходе при трансляции из C-light в C-kernel эти операторы заменяются на оператор `goto`, для которого развит Хоаровский подход с использованием инвариантов. Отметим также, что операторы `break` и `continue` передают управление «вперед», поэтому пользователю не нужно самому определять инварианты.

Для автоматического доказательства порожденных условий корректности применялись решатели Simplify и Z3. Отметим, что частое использование указателей и массивов в этих примерах приводит к разрастанию порождаемых условий корректности. В ходе анализа условий был предложен ряд стратегий по предварительному упрощению формул. Как правило, суть стратегий сводится к выделению основных целей при верификации конкретной программы и отбрасыванию несущественной в данном случае информации. Подробное описание примеров, спецификаций и условий корректности можно найти в [1, 10].

6. Заключение

В статье представлен прототип расширяемой мультиязыковой системы анализа и верификации, разрабатываемой в рамках проекта СПЕКТР, и показаны перспективы ее использования на примере верификации С-программ. Расссмотрен двухуровневый метод верификации С-программ, на котором базируется С-компонента этой системы. Из известной коллекции Java-программ [9], представляющих трудности для верификации, были отобраны, переписаны на языке С и верифицированы примеры, относящиеся к процедурному случаю.

Три проекта по верификации С-программ близки представленному проекту.

Примером узкоспециализированного проекта по верификации является проект VERISOFT [6], ориентированный в основном на встраиваемые системы. Одна из целей проекта — верификация ядра операционной системы для простого, но верифицированного процессора. Используется простое подмножество языка С — язык С0, семантика которого моделируется в системе доказательства теорем Isabelle/HOL. В силу ограниченности С0 в программах приходится использовать язык ассемблера, что усложняет верификацию. Вместе с тем, на языке С0 были написаны верифицированные библиотеки обработки строк и списков.

Перспективный подход к верификации С-программ предложен в рамках проекта Why [8]. Отметим, что Why — это платформа, пригодная для верификации многих императивных языков. В ней определен одноименный промежуточный язык, в который можно транслировать входные программы. Цель трансляции — генерация условий корректности в виде, не зависящем от системы доказательства теорем. На основе Why построен инструментарий Frama-C, предлагающий статический анализ для полного языка С и дедуктивную верификацию для ограниченного подмножества. Из ограничений отметим запреты на **goto** «назад» и внутрь блока, указатели

на функции, приведения типов между числами и указателями, объединения, функции с переменными списками параметров, вещественные вычисления. Также с помощью языка спецификаций ACSL было аннотировано подмножество стандартной библиотеки, включающее важные функции работы с памятью и файлами. Список верифицированных программ включает довольно простые программы поиска и сортировки.

Наконец, отметим проект VCC (A Verifier for Concurrent C) [7]. В нем аннотированные программы транслируются в логические формулы с помощью инструмента Boogie, который объединяет в себе промежуточный язык Boogie PL и генератор условий корректности. Для доказательства условий используется SMT-решатель Z3. Заметим, что язык Boogie PL, как и Why, не привязан только к C. В частности он же используется в проекте Spec#, также разрабатываемом в Microsoft Research. К недостаткам, как и для Why, можно отнести тот факт, что трансляция происходит в совершенно другой язык, и вопрос корректности трансляции остается открытым. Основной целью в проекте VCC является верификация гипервизора Hyper-V, поэтому информация о других успешно решенных примерах крайне скудна.

Анализ рассмотренных проектов позволяет указать ряд присущих им недостатков. Во-первых, часто теории предметных областей разрабатываются в них под конкретный, достаточно ограниченный класс задач. С другой стороны, для систем общего назначения (например Why) описаны довольно простые примеры верификации, что говорит о сложности их настройки на реальные примеры. Во-вторых, в них главенствует "закрытый" подход к разработке. Закрытость в данном контексте означает, что методы разрабатываются группами специалистов по верификации, участвующими в проекте. Пользователи таких систем вынуждены ожидать появления новых версий или включения новых методов. В то же время, сложность использованных подходов препятствует самостоятельному анализу и модификации таких систем обычными пользователями даже при наличии исходного кода.

Система СПЕКТР разрабатывается на базе интегрированного подхода, позволяющего преодолеть указанные недостатки. Особо подчеркнем мультиязыковость и расширяемость системы и использование специализированного языка описания методов и техник верификации программ. Удобный специализированный язык позволяет пользователю системы описывать в естественной нотации методы и техники верификации, переносить разработанные техники с одного языка программирования на другой, специфицировать верифицируемые алгоритмы в различных предметных областях, добавляя при необходимости свои языки для их представления, разделять методы и техники верификации с другими пользователями системы и комбинировать их. Процесс вовлечения пользователя в разработку методов верификации для его конкретных задач должен повысить как уровень проникновения формальных методов в сферу разработки программного обеспечения, так и качество верификации программ. Также очевидна польза универсального подхода для задачи обучения методам верификации.

Что касается дальнейшего развития проекта, то эффективность интегрированного мультиязыкового подхода к верификации императивных программ будет проверена, в первую очередь, на авторских разработках в области анализа и верификации программ. Планируется провести эксперименты по верификации учебных программ

на языке Паскаль и объектно-ориентированных программ на языке C#.

Список литературы

1. **Ануреев И.С., Марьясов И.В., Непомнящий В.А.** Верификация С-программ на основе смешанной аксиоматической семантики // Моделирование и анализ информационных систем. 2010. Том 17, № 3. С. 5–28.

2. **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.

3. **Непомнящий В.А., Ануреев И.С., Михайлов И.Н., Промский А.В.** Ориентированный на верификацию язык C-light // Системная информатика: Сб. науч. тр. Новосибирск: Издательство СО РАН, 2004. Вып. 9: Формальные методы и модели информатики. С. 51—134.

4. **Непомнящий В.А., Ануреев И.С., Промский А.В., Дубрановский И.В.** На пути к верификации C# программ: трехуровневый подход // Программирование. 2006. № 4. С. 4–20.

5. **Anureev I.S.** A three-stage method of C program verification // Joint NCC&IIS Bulletin, Series Computer Science. 2008. Vol. 28. P. 1–29.

6. **Alkassar E., Hillebrand M.A., Leinenbach D., Schirmer N.W., Starostin A.** The Verisoft Approach to Systems Verification // VSTTE 2008. Lect. Notes Comput. Sci. 2008. Vol. 5295. P. 209–224.

7. **Cohen E., Dahlweid M., Hillebrand M.A., Leinenbach D., Moskal M., Santen T., Schulte W., Tobies S.** VCC: A Practical System for Verifying Concurrent C // Proc. TPHOLs 2009. LNCS. 2009. Vol. 5674. P. 23—42.

8. **Filliâtre J.C., Marché C.** Multi-prover verification of C programs // Proc. ICFEM 2004. LNCS. 2004. Vol. 3308. P. 15-–29.

9. **Jacobs B., Kiniry J.L., Warnier M.** Java program verification challenges // Proc. FMCO 2002. Lect. Notes Comput. Sci. 2003. Vol. 2852. P. 202-–219.

10. **Promsky A.V.** Towards C-light Program Verification: Overcoming the Obstacles // Proc. International Workshop on Program Understanding, 19—23 June, Altai Mountains, Russia, 2009. P. 53—63.

C Program Verification in the Multilanguage System Spectrum

Nepomniashy V.A., Anureev I.S., Atuchin M.M., Maryasov I.V., Petrov A.A., Promsky A.V.

Keywords: verification, specification, operational semantics, axiomatic semantics, domain-specific languages, verification systems

This paper presents the expendable multi-language analysis and verification system SPECTRUM, which is being developed within the framework of the project SPECTRUM. The project prospects are discussed using the example of C program verification. The project aims at the development of a new integrated approach to program verification which will allow the integration, unification and combination of program verification techniques together with accumulation and reuse of knowledge about them. One of the project features consists in the use of the specialized executable specification language Atoment. This language provides a unified format to represent both verification methods and data for them (program models, annotations, logic formulas). The C-targeted component of the SPECTRUM system is based on our two-level C program verification method. This method represents a good illustration of the integrated approach, since it provides a complex C program verification that combines operational, axiomatic and transformational approaches.

Сведения об авторах:
Непомнящий Валерий Александрович,
Институт систем информатики имени А.П. Ершова СО РАН,
зав. лабораторией теоретического программирования,
Новосибирский государственный университет, доцент;
Ануреев Игорь Сергеевич,
Институт систем информатики имени А.П. Ершова СО РАН, ст. науч. сотр.;
Атучин Михаил Михайлович,
Новосибирский государственный университет, студент;
Марьясов Илья Владимирович,
Институт систем информатики имени А.П. Ершова СО РАН, мл. науч. сотр.;
Петров Анатолий Александрович,
Новосибирский государственный университет, студент;
Промский Алексей Владимирович,
Институт систем информатики имени А.П. Ершова СО РАН, науч. сотр.