Применение формальных методов для тестирования компиляторов

М.А. Посыпкин

Аннотация. В работе предлагается технология автоматизированной генерации тестовых наборов для компиляторов по формальному описанию синтаксиса, статической и динамической семантики языка программирования. Предложены новые решения некоторых известных задач тестирования компиляторов. В частности, определяются новые критерии покрытия для тестов, основанные на формальной спецификации семантики языка программирования и предлагаются методы генерации корректных программ с однозначным наблюдаемым поведением. Также в статье рассматриваются существующие подходы к тестированию компиляторов и приводятся результаты практической апробации предложенной технологии.

1. Введение

Языки высокого уровня давно стали основным средством разработки программного обеспечения. Это явилось причиной широкого распространения программ, обеспечивающих процесс такой разработки, в частности, компиляторов.

Компиляторы решают задачу перевода языка высокого уровня в представление, которое может быть выполнено ЭВМ. Ошибки в компиляторах приводят к тому, что программы на исходном языке транслируются в исполняемые модули, поведение которых отличается от поведения, определяемого семантикой исходной программы. Такого рода ошибки очень сложно выявлять и исправлять, их наличие ставит под сомнение качество компонент, для генерации которых использовался компилятор. Можно с уверенностью утверждать, что корректность работы компилятора с языка программирования является необходимым условием надежной работы любого программного обеспечения, реализованного с его помощью, а контроль правильности работы компиляторов - одной из важнейших мер по обеспечению надежности программного обеспечения. Для компилятора, как и для любого другого вида программного обеспечения, одним из важнейших методов обеспечения надежности является тестирование.

Процесс тестирования включает в себя три основные задачи:

- Генерация (написание) тестов.
- Вынесение вердикта о прохождении теста. Эта задача решается с помощью так называемого тестового оракула. *Тестовым оракулом* называется процедура, определяющая корректность работы тестируемой системы на данном тесте.
- Оценка качества тестов. Для этой задачи обычно применяются так называемые *критерии покрытия*. *Покрытием* называется набор подмножеств множества входных данных. Говорят, что тестовый набор удовлетворяет *критерию покрытия*, если для каждого множества из покрытия найдется по крайней мере один принадлежащий ему тест из тестового набора.

Для решения этих задач применяются различные методы, которые можно разделить на две большие группы - методы “черного” и “белого” ящика. В первом случае единственным источником информации для создания тестов является описание функциональности тестируемого продукта, также называемое *спецификацией*. Во втором, для создания тестов также используется информация о реализации, исходный код тестируемого продукта.

Для тестирования компиляторов важны как методы “черного ящика”, так и методы “белого ящика”. Данная работа посвящена тестированию компилятором на основе формальных спецификаций синтаксиса и семантики языка программирования, т.е. методам “черного ящика”'. Наборы тестов, созданные по методу “черного ящика”, предназначены для проверки соответствия компилятора требованиям синтаксиса и семантики языка программирования. Корректная обработка таких тестовых наборов является необходимым требованием, предъявляемым к качеству любой реализации компилятора для данного языка программирования.

В следующем разделе рассматриваются особенности тестирования компиляторов и дается общая характеристика предлагаемого подхода к решению основных задач тестирования.

2. Проблема тестирования компиляторов и общая характеристика предлагаемого метода

Язык программирования определяется синтаксисом, статической семантикой и динамической семантикой. Спецификация каждого из этих аспектов языка может быть основой для создания тестов. Спецификации синтаксиса и семантики задают систему вложенных подмножеств множества всех возможных цепочек терминальных символов (Рис. 1).

Синтаксис языка задается грамматикой, содержащей терминальные символы, нетерминальные символы и правила вывода. Цепочки терминальных символов, выводимые из стартового символа грамматики, называются *синтаксически*

корректными программами. Множество синтаксически корректных программ является подмножеством множества всех цепочек терминальных символов (Рис. 1). Они используются для тестирования того, что фаза синтаксического анализа распознает тест как корректный. Цепочки, не выводимые из стартового символа, используются для проверки способности компилятора распознавать синтаксическую ошибку.

Статическая семантика, определяемая только для синтаксически корректных программ, задает правила вычисления свойств программы, не требующих для своего вычисления ее выполнения. К таким свойствам относятся, в частности, типы переменных и выражений. Помимо правил вычисления задаются также правила проверки статической корректности программы, или *контекстные условия*, которые накладывают ограничения на возможные комбинации значений статических свойств программы.

Интерес с точки зрения тестирования представляют как программы, удовлетворяющие контекстным условиям, так и программы, нарушающие их. Программы, удовлетворяющие контекстным условиям, называются *статически корректными*. Статически корректные программы являются подмножеством множества синтаксически корректных программ (Рис. 1). Они используются для тестирования того, что фаза статического анализа в компиляторе правильно распознает корректные программы. Синтаксически корректные программы, нарушающие контекстные условия, используются для проверки способности компилятора распознавать статические ошибки. Такие программы в дальнейшем также будем называть *негативными тестами*.

Динамическая семантика определяет смысл выполнения статически корректных программ языка. Для тестирования ее реализации в компиляторе используются статически корректные программы, которые компилируются тестируемым компилятором, полученные в результате загрузочные модули выполняются, после чего их наблюдаемое поведение сравнивается с эталонным, определяемым динамической семантикой языка. Очевидно, что если программы не обладают *однозначно определенным наблюдаемым поведением*, то такое сравнение становится очень сложной задачей. Программы с однозначным наблюдаемым поведением составляют подмножество во множестве всех статически корректных программ (Рис. 1) и далее в статье называются *позитивными тестами*.

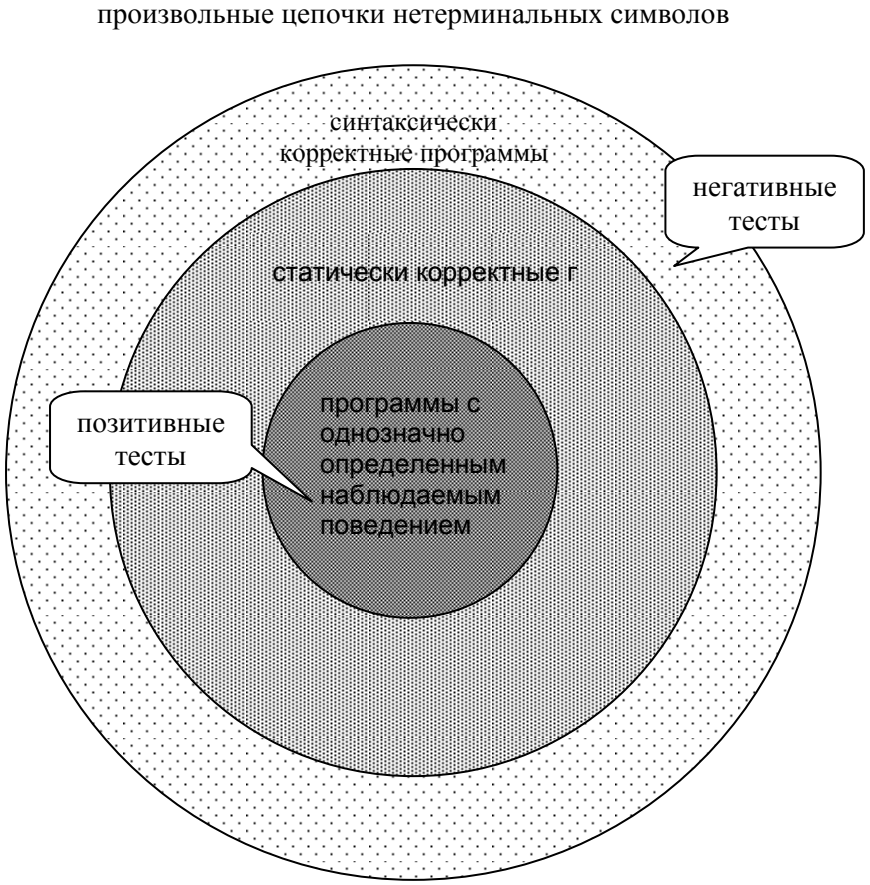

Рис. 1. *Структура входных данных компилятора.*

Тестированию компиляторов посвящено большое число публикаций. Краткий обзор этих работ приводится в конце данной статьи. Анализ опубликованных в них результатов позволяет сделать выводы о степени решенности основных задач тестирования в случае компиляторов. Эти выводы приведены в таблице 1. Строки таблицы соответствуют задачам тестирования, а столбцы - составляющим описания языка программирования. Жирным шрифтом выделены нерешенные проблемы.

	Синтаксис	*Статическая семантика*	*Динамическая семантика*
Генерация тестов	Предложен ряд алгоритмов генерации синтаксически корректных и некорректных программ.	Предложены алгоритмы генерации корректных программ. **Нет алгоритмов генерации тестов, содержащих ошибки.**	Решена для ряда частных случаев. **Отсутствуют алгоритмы генерации программ с однозначно определенным поведением.**
Тестовый оракул	Для тестов, нарушающих требования языка, проверяется то, что компилятор обнаруживает ошибку. Для тестов, удовлетворяющих требованиям языка, проверяется то, что компилятор обработал их как правильные, т.е. не выдал диагностику об ошибке и не вызвал внутреннюю ошибку.		Основывается на сравнении наблюдаемого поведения.
Критерий покрытия	Предложен ряд критериев, для которых доказана достижимость.	Предложен ряд критериев. **Достижимость не доказана.**	**Отсутствуют какие-либо критерии покрытия.**

Таблица 1. *Степень решенности основных задач, связанных с тестированием компиляторов.*

В данной работе предлагается новый метод автоматизированной генерации программ для проверки реализации статической и динамической семантики в компиляторе, с помощью которого удается решить следующие задачи, для которых (согласно таблице 1) не было ранее предложено удовлетворительного решения:

- Генерация тестов, нарушающих контекстные условия (негативные тесты).
- Генерации тестов, предназначенных для проверки реализации динамической семантики в компиляторе которые представляют собой статически корректные программы с однозначно определенным поведением (позитивные тесты).
- Разработка критериев покрытия для негативных и позитивных тестов.

Тестовые программы генерируются для некоторого подмножества языка, для которого формально задаются синтаксис и семантика. Эти формальные спецификации используются для генерации тестов, тестового оракула и измерения покрытия. Выбор подмножества языка, для которого генерируются тесты, осуществляется тестировщиком исходя из особенностей реализации

тестируемого компилятора. Это подмножество обычно включает в себя наиболее сложные с точки зрения реализации или наименее стабильно работающие конструкции языка.

3. Генератор тестовых программ

Генератор тестовых программ представляет собой многокомпонентную систему, общая схема которой представлена на Рис. 2.

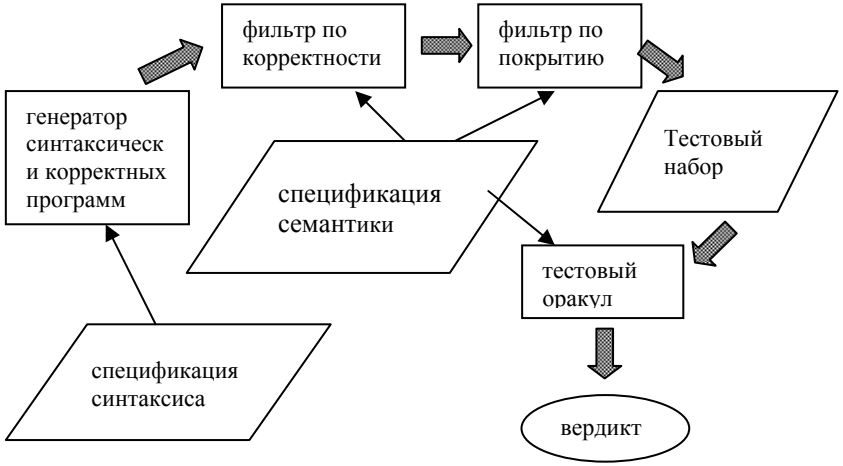

Рис. 2. *Общая схема работы генератора тестовых программ.*

На основе описания синтаксиса языка, *генератор синтаксически корректных программ* синтезирует синтаксически корректные программы, которые поступают на вход *фильтру по корректности*, который оставляет только позитивные либо только негативные тесты (в зависимости от целей тестирования). Генерируемые программы подаются в качестве входных данных *фильтру по покрытию*, который отбрасывает программы, не увеличивающие покрытие. Таким образом происходит формирование *тестового набора*, который затем используется для проверки правильности работы тестируемой системы с помощью *тестового оракула*. Компоненты фильтр по покрытию, фильтр по корректности и тестовый оракул используют формальную спецификацию семантики языка программирования. Далее в этом разделе мы подробно рассмотрим каждую из перечисленных компонент.

3.1. Генератор синтаксически корректных программ

Входными данными алгоритма являются множество предопределенных фраз (цепочек терминальных символов, выводимых из нетерминалов языка),

соответствующих некоторым нетерминальным символам грамматики, и множество продукционных правил. Эти два множества определяют синтаксис подмножества языка, для которого генерируются тесты. Работа алгоритма заключается в последовательном построении фраз снизу-вверх: новые фразы генерируются с помощью применения продукционных правил к ранее сгенерированным фразам.

3.2. Тестовый оракул

Для негативных тестов оракул должен проверять, что компилятор диагностировал ошибку статической семантики. Реализация такого оракула не представляет сложностей.

Для позитивных тестов оракул работает следующим образом: тестовая программа компилируется тестируемым компилятором и выполняется. После чего полученное наблюдаемое поведение сравнивается с эталонным, т.е. соответствующим семантике языка.

Для реализации такого оракула, необходимо, во-первых, чтобы эталонное наблюдаемое поведение было однозначно определено и, во-вторых, уметь получать это наблюдаемое поведение. Неоднозначность наблюдаемого поведения является критическим фактором с точки зрения тестового оракула. Рассмотрим следующую программу на Си:

```
int *p = NULL;
main()
{
    printf("%d\n", *(p + 1));
}
```

Эта программа выводит на экран целое число, расположенное в памяти по адресу (p+1). Согласно стандарту Си, поведение этой программы не определено, т.к. она включает в себя вычисления с указателем на память, не принадлежащую адресному пространству процесса программы. Неопределенное поведение означает, что возможны любые варианты реакции тестируемой системы на такую программу, например:

- компилятор выдаст диагностику об ошибке;
- во время выполнения будет выдана диагностика об ошибке;
- будет напечатано некоторое значение;

Таким образом, любое поведение компилятора на таком тесте можно считать правильным и проверка правильности этого поведения является очень сложной задачей.

Можно сформулировать ряд требований, которым должны удовлетворять позитивные тесты:

- тесты не должны вызывать поведения, не определенного в описании языка;

- наблюдаемое поведение тестов должно быть детерминировано (однозначно определяться входными данными);
- наблюдаемое поведение не должно зависеть от параметров описания языка, определяемых реализацией;

Такие программы будем называть *строго-конформными*. Задача отбрасывания программ, не являющихся строго-конформными, решается фильтром строго-конформных программ, который является частью фильтра по корректности, рассматриваемого в следующем разделе.

В подходе, предлагаемом в данной работе, для получения эталонного наблюдаемого поведения используется *интерпретатор динамической семантики языка программирования*. Для реализации такого интерпретатора используется инструмент Gem-Mex [14], который позволяет автоматически генерировать интерпретаторы по спецификациям в формализме Montages[15]. Формализм Montages основан на другом формализме – Abstract State Machines, предложенном Гуревичем [18].

3.3. Фильтр по корректности

Задачей фильтра по корректности является задача отбрасывание программ, не являющихся позитивными или негативными тестами. Общая схема работы фильтра по корректности представлена на Рис. 3.

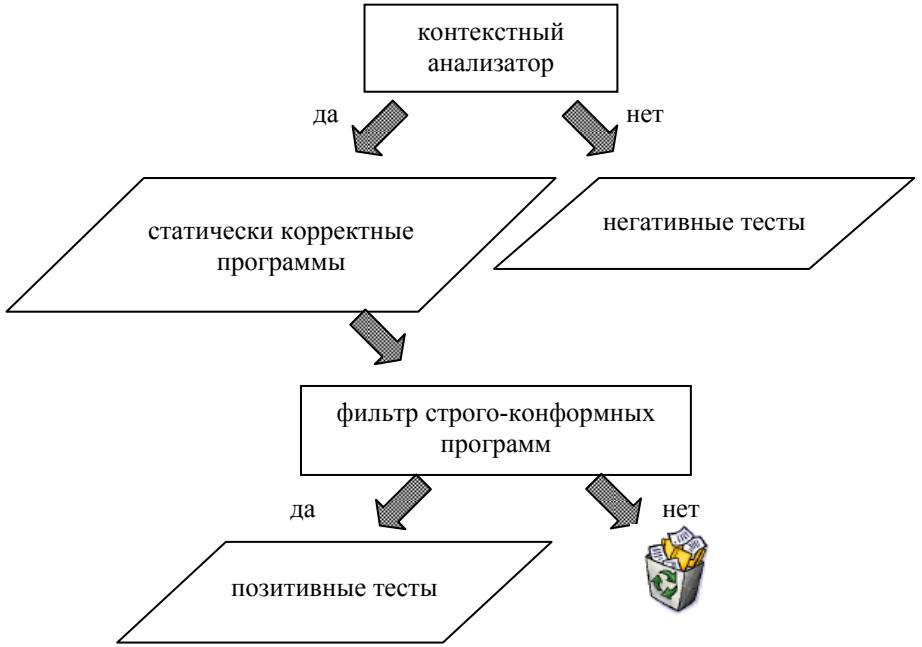

Рис. 3. *Общая схема фильтра по корректности.*

Синтаксически корректные программы обрабатываются контекстным анализатором. Если хотя бы одно из контекстных условий нарушено, то тест добавляется к негативному тестовому набору. В противном случае программа является статически корректной и поступает для дальнейшей обработки на вход фильтру строго-конформных программ, который отбрасывает программы, не являющиеся строго-конформными.

В общем случае для сложных используемых на практике языков программирования, в частности для языка Си, автоматическое выяснение вопроса строгой конформности произвольной программы является очень сложной и возможно алгоритмически неразрешимой задачей.

Часто для целей тестирования достаточно использования некоторого подмножества языка, для программ из которого возможно автоматическое решение вопроса о строгой конформности. Предлагаемый в данной работе подход заключается том, что для подмножества языка, которое предполагается использовать для генерации тестов, формально определяется семантика таким образом, что любая динамически корректная программа в этой семантике является строго конформной программой в исходном языке. В соответствии с формальным определением этой семантики строится интерпретатор семантики, который выносит вердикт о динамической корректности программы, которая означает строгую конформность программы по отношению к исходному языку.

Построение таких подмножеств и формальное определение их семантики является нетривиальной задачей, которая может решаться по-разному для разных языков. Автором построены такие подмножества и формально определена семантика для языков Си и mpC. Случай языка Си подробно рассматривается в работе [16]. Подмножество включает в себя адресную арифметику, что демонстрирует способы обработки конструкций, которые являются основными источниками неопределенного поведения в Си-программе. Для этого подмножества была формально определена статическая и динамическая семантика, для которой было показано, что любая программа, выполнение которой не приводит в ней к специально выделенному ошибочному состоянию, является строго-конформной. Данный пример показывает, что подмножества, для которых строгая конформность может быть формально показана, могут быть достаточно сложными.

3.4. Фильтр по покрытию

После фильтра по корректности негативные и позитивные тесты поступают в качестве входных данных для фильтров по покрытию. Задачей фильтра по покрытию является измерение и сохранение информации о покрытии и отбрасывание тестов, не увеличивающих покрытие. Для измерения покрытия тестов нами были разработаны ряд новых критериев покрытия для спецификаций семантики языка программирования в формализме Montages.

Для позитивных тестов информация о покрытии строится на основе анализа путей выполнения в интерпретаторе динамической семантики. Тестовый

набор, удовлетворяющий критерию покрытия, должен проходить все достижимые пути заданной длины в интерпретаторе. Этот критерий называется критерием *покрытия по n-путям*. Формальное определение этого критерия приведено в работе [19]. Критерий покрытия по n-путям является усилением критерия покрытия правил, предложенного ранее в работе Gargantini и Riccobene [17] для ASM-спецификаций.

Для негативных тестов применяется *критерий покрытия ограничений*, основанный на анализе причины нарушения контекстного условия. В большинстве случаев контекстное условие может быть сформулировано как одновременное выполнение нескольких семантических условий, называемых базовыми условиями. При этом, причиной нарушения контекстного условия может быть нарушение любого из его базовых условий. Покрытие ограничений достигается на наборе негативных тестов, если для каждой такой причины найдется тест, нарушающий семантику в соответствии с ней. Формально определение критерия покрытия ограничений приводится в работе [19].

4. Практическая апробация

Методика тестирования, изложенная в данной работе, была применена для генерации тестовых наборов для компиляторов с языков Си и mpC.

Для тестирования компилятора с языка Си было взято подмножество, содержащее целый, все вещественные и комплексные типы данных, а также конструкторы типа массив и указатель, объявления переменных и ряд операций. С помощью системы Gem-Mex была создан интерпретатор этого подмножества, способный интерпретировать программы из описанного подмножества, диагностировать ошибки синтаксиса, статической и динамической семантики.

Был сгенерирован набор из 862 тестовых программ. С помощью полученного тестового набора были протестированы компилятор gcc версии 3.3 и отладчик gdb версии 6.0 на платформе Intel. В компиляторе не было обнаружено ни одной ошибки, что косвенно подтверждает строгую конформность сгенерированных программ. В результате тестирования отладчика было выявлено две ошибки: обе на тестах, содержащих динамические массивы. В первом случае неверно вычислялось значение разности между указателями на соседние элементы двумерного динамического массива, а во втором отладчик зависал при вычислении разности двух указателей на один и тот же динамический массив. Также была обнаружена ошибка в бета-версии промышленного компилятора.

Тестирование компилятора языка mpC [20] проводилось в рамках промышленного проекта по созданию интегрированной среды программирования mpC-Workshop [21].

Компилятор mpC транслирует исходную программу в программу на Си, содержащую вызовы run-time библиотеки mpC. При этом конструкции языка,

не содержащие специфичных для mpC операций и объявлений, транслируются без изменений. Выражения, содержащие операции, отсутствующие в Си, транслируются в систему вложенных циклов, тела которых могут содержать вызовы функций из run-time библиотеки mpC для обменов данными или просто в последовательность пересылок данных.

Модуль трансляции выражений является самой большой и наиболее сложной частью компилятора, так как остальные конструкции транслируются практически без изменений или сводятся к выражениям с помощью простых правил переписывания абстрактного синтаксиса. Опыт использования прототипа mpC продемонстрировал, что именно этот модуль является основным источником ошибок в работе транслятора. Поэтому, для тестирования было выбрано подмножество mpC, содержащее объявления сетевых объектов, объектов данных и выражения. Для этого подмножества была написана Montages-спецификация и получен интерпретатор языка. В процессе работы над формализацией языка, выяснилось, что ряд аспектов семантики языка нуждаются в уточнении и переработке. В результате этой работы было написано уточненное выражений описания языка, опубликованное в [22].

Было сгенерировано несколько тестовых наборов для компилятора mpC, содержащих негативные и позитивные тесты. В результате тестирования удалось обнаружить более 30 ошибок в различных компонентах компилятора.

5. Сравнение с другими работами

В работах Purdom [1], Lammel [2], Maurer [3], Guilmette [4] рассматривается вопрос тестирования синтаксических анализаторов. В этих работах предлагаются различные алгоритмы для систематической или стохастической генерации тестов по контекстно-свободной грамматике, критерии покрытия для синтаксически корректных программ.

Неавтоматизированный подходы к тестированию реализации статической семантики в компиляторе на основе VDM описания рассматривается в работах Петренко и др. [5]. Автоматические методы генерации статически корректных программ по атрибутным грамматикам рассматриваются в работах Harm и Lammel [6, 7], а также в работе Duncan и Hutchison [8]. Авторами предложены критерии для покрытия статически корректных тестов. и не было предложено каких-либо критериев покрытия для тестов, нарушающих контекстные условия.

Использование формальных спецификаций для составления тестовых наборов для проверки реализации динамической семантики “вручную” рассматривается в работах Кауфмана [9], Сухомлина [10], Баскакова [11]. Автоматизированная система для генерации тестов для языка Java описана в работе Sirer и Bershad [12], в ней для получения эталонного наблюдаемого поведения предлагается использовать либо альтернативный компилятор либо формальную

денотационную семантику. Работа McKeeman [13] посвящена автоматической генерации тестов для компиляторов с языка Си. В этой работе ставится вопрос о том, что генерируемые программы могут опираться на зависящие от реализации параметры семантики и по-разному работать для разных корректных реализаций этой семантики. Предлагаемые автором методы для решения этой проблемы основаны на инструментировании генерируемого код дополнительными функциями, осуществляющими проверку принадлежности значения выражения диапазону его типа. Хотя такой подход и позволяет выявить значительную часть нарушений динамической семантики, он не решает задачу в полном объеме, поскольку использование компилятора для проверки семантики не позволяет полностью исключить влияние ряда реализационных параметров на поведение программы (таких как размеры типов в байтах, последовательность размещения переменных в памяти, порядок вычисления операндов операций и др.).

6. Заключение

В статье рассмотрена технология автоматической генерации тестов для компилятора по спецификациям синтаксиса и семантики подмножества языка программирования.

Эта технология была применена для генерации тестовых наборов для подмножеств языков Си и mpC. Полученные тестовые наборы позволили обнаружить ряд ошибок в отладчике GDB, бета-версии промышленного компилятора с языка Си и компиляторе mpC.

Хотя предложенная методика в принципе позволяет генерировать тесты для произвольного подмножества языка, существует ряд факторов, ограничивающих спектр ее возможных применений. Наиболее существенным является большое число генерируемых тестов в случае, когда грамматика подмножества содержит много правил. Возможным развитием дальнейших исследований может быть создание более экономных алгоритмов генерации тестовых наборов, удовлетворяющих критерию покрытия.

Благодарности. Автор благодарен в.н.с. ИСПРАН к.ф.м.н. А.С. Косачеву за научное руководство при получении результатов, вошедших в публикацию, и обсуждение ее текста. Также автор признателен в.н.с. ИСПРАН, д.ф.м.н. А.К. Петренко и с.н.с. ИСПРАН к.т.н. А.Я. Калинову за плодотворное обсуждение результатов, вошедших в статью.

Список литературы

1. Purdom P. A sentence generator for testing parsers. BIT, 2:336-375, April, 1972.
2. Lammel R. Grammar testing. In Proc. of FASE’2001, LNCS, v 2029, pp. 201-216, 2001.
3. Maurer P.M. Generating test data with enhanced context-free grammars. IEEE Software, pp. 50-55, 1990.
4. Guilmette R. TGGS: A flexible system for generating efficient test case generators. 1999.

5. Петренко А., Чацкина Т., Борисова М., Морозова Т. Тестирование компиляторов на основе формальной модели языка.
6. Harm J. Automatic test program generation from formal language specifications. RIP(1997), 20: 33-56, 1997.
7. Harm J., Lammel R., Testing attribute grammars. In Proc. of WAGA'00, pp. 79-98, 2000.
8. Duncan A., Hutchison J., Using attribute grammars to test design and implementations. In Proc. of ICSE'81, pp. 170-178, 1981.
9. Кауфман В. Стандартизация и контроль трансляторов. Различные аспекты системного программирования, pp. 47-85, 1984.
10. Сухомлин В. Система программирования тройного стандарта 3С++. 4-я международная конференция "Развитие и применение открытых систем", тезисы докладов, pp. 37-47, 1997.
11. Баскаков Ю.В. Принципы построения тестовых комплектов для тестирования конформности компиляторов стандартам языков программирования. Теоретические и прикладные аспекты информационных технологий: сборник трудов под редакцией проф. Сухомлина. В.А., 2001.
12. Sirer E., Bershad B. Using production grammars in software testing. In Proc. of DSL'99, 1999.
13. McKeeman W. Differential testing for software. DTG, 10(1):100-107, 1998.
14. eXtensible Abstract State Machines. http://www.xasm.org.
15. Kutter P., Pierantonio A. Montages: specifications of realistic programming languages. JUCS, 3(5):443-503, 1997.
16. Косачев А., Куттер Ф., Посыпкин М. Автоматическая генерация строго-конформных тестов по формальной спецификации языков программирования. Программирование (в печати), 2004.
17. Garagantini A,, Riccobene E. ASM-based testing: coverage criteria and automatic tests generation. In Proc. of Eurocast'2001, pp. 262-265, 2001.
18. Gurevich Y. Evolving algebras: Lipari guide. Specification and Validation Methods, pp. 9-36, 1995.
19. Kalinov A., Kossatchev A., Petrenko A., Posypkin M., Shishkov V. Coverage-driven automated compiler test suite generation. ENTSC:82(3): 2003.
20. Lastovetsky A. mpC – a multi-paradigm programming language for massively parallel computers. ACM SIGPLAN Notices, 31(5):13-20, 1996
21. mpC Workshop for Windows. http://mpcw.ispras.ru.
22. Калинов А., Ластовецкий А., Ледовских И., Посыпкин М. Пересмотренное сообщение о языке программирования С[]. Программирование 1,1995.