Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

Демаков А.В., Зеленова С.А., Зеленов С.В.

Тестирование парсеров текстов на формальных языках[1]

Введение

В данной работе рассматриваются вопросы, связанные с систематическим тестированием на основе формальных спецификаций.

Спецификация – это абстрактное, независимое от реализации, описание поведения программы. Тестирование на основе спецификаций есть проверка соответствия поведения реальной системы ее специфицированному поведению.

Тестирование на основе спецификаций обладает рядом достоинств:

- спецификации позволяют отвлечься от деталей реализации, а значит предоставляют возможность использовать один и тот же набор тестов, построенный по этим спецификациям, для проверки нескольких различных реализаций тестируемой системы;
- тестирование на основе спецификаций, т.е. эталона поведения, дает возможность выбрать критерий полноты тестового набора, позволяющий выявить неполную функциональность программного продукта;
- спецификация обычно менее объемна и легче для восприятия, чем реализация, поскольку представляет собой некоторую факторизацию реальной системы.

В настоящее время в качестве спецификации компилятора рассматривают неформальное или частично-формальное описание входного языка этого компилятора. Лишь парсер обычно полностью формализован.

Для тестирования компиляторов в основном применяются комплекты тестовых программ, написанных вручную на основе такого неформального описания целевого языка.

Такой подход, безусловно, имеет право на существование. При этом для обеспечения систематичности тестирования главное – это задать критерий покрытия и создать множество тестов, которое ему

[1] Данная работа частично поддержана грантом РФФИ 99-01-00207.

Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

соответствует. Однако, использование неформальных спецификаций обладает существенными недостатками:

- при построении тестов для фиксированного критерия покрытия большую роль играет субъективный фактор, а следовательно, нельзя гарантировать качества тестового покрытия;
- ручная разработка требует слишком много ресурсов.

В то же время использование формальных спецификаций при тестировании имеет много положительных сторон:

- легко формулировать критерии полноты покрытия;
- возможна автоматизация построения тестов;
- для автоматически построенного тестового набора легко проверить, достигнуто ли требуемое покрытие.

1. Постановка задачи

В настоящей работе мы ограничимся рассмотрением парсеров, то есть синтаксических анализаторов. Более того, мы предполагаем, что целевой язык парсера является контекстно-свободным.

Пусть задан парсер, отвечающий некоторому формальному контекстно-свободному языку. Требуется проверить, что он работает правильно, т.е. корректно выносит вердикт о принадлежности входного предложения языку.

При этом задача разбивается на две:

- построение набора «правильных тестов», то есть предложений, принадлежащих языку;
- построение набора «неправильных тестов», то есть предложений, не принадлежащих языку.

Решение каждой из этих подзадач в свою очередь состоит из двух этапов: описание метрики тестового покрытия и разработка генератора входных тестовых предложений.

Здесь мы не будем останавливаться на алгоритме генерации, а подробно рассмотрим метрику тестового покрытия, которая позволяет построить семейство критериев, удобных для определения полноты тестирования парсеров на основе «правильных тестов».

Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

2. Метрика покрытия для правильных тестов

Основные определения.

Назовем *нетерминальным подвыражением* такую часть правила BNF, которая является либо ссылкой на некоторое правило, либо выражением в скобках (см. Приложение А).

С каждым нетерминальным подвыражением свяжем *фронтальное множество*, состоящее из всех токенов (т.е. терминальных символов), каждый из которых служит началом в каком-нибудь раскрытии исходного подвыраженя.

Будем говорить, что нетерминальное подвыражение является *точкой однозначного локального продолжения*, если его фронтальное множество состоит из одного элемента.

Нетерминальное подвыражение, не являющееся точкой однозначного локального продолжения, назовем *точкой ветвления*.

Назовем *множеством локальных ситуаций* множество всевозможных пар (*f*, **t**), где *f* – точка ветвления, а **t** – токен из фронтального множества точки *f*.

Критерий покрытия.

Предложенный выше набор понятий позволяет формулировать разнообразные критерии покрытия различной мощности. Остановимся подробно на критерии, который мы использовали в своей работе.

Рассмотрим какое-нибудь предложение языка и все выводы этого предложения в данной грамматике. Будем говорить, что предложение *покрывает* локальную ситуацию (*f*, **t**) если в некотором выводе встречается эта локальная ситуация.

Наш критерий требует построить такой набор предложений языка, которые в совокупности покрывают все элементы множества локальных ситуаций. Будем называть этот критерий «все локальные ситуации».

Заметим, что в предложенных терминах можно определить много других критериев.

Пример критерия меньшей мощности: для каждой точки ветвления *f* покрыть какую-нибудь локальную ситуацию (*f*, **t**).

Пример критерия большей мощности: для любой пары локальных ситуаций построить предложение языка, покрывающее их обе, если это возможно, или каждую по отдельности в противном случае.

Построенный по критерию «все локальные ситуации» набор тестов для некоторой BNF контекстно-свободного языка проверяет поведение парсера для этого языка в точках ветвления.

Существует широкий класс парсеров, именно – парсеров, осуществляющих LL-разбор (т.е. двигающихся сверху вниз), для

Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

которых поведение в этих точках особенно важно. Это утверждение основывается на том факте, что любой подобный парсер, дойдя в разбираемом предложении до места, соответствуюшего некоторой точке ветвления, должен будет сделать выбор на основании нескольких следующих символов.

Эту особенность LL-разбора не учитывают методики генерации тестов, которые пытаются покрыть лишь все правила или все альтернативы.

Пример.

Проиллюстрируем обсуждавшиеся выше понятия на примере.

Пусть *A, B, C* – правила грамматики; **a**, **b**, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{c}_1$, $\mathbf{c}_2$ – терминальные символы. Рассмотрим следующую BNF:

$$A ::= \mathbf{a}\, B\, C;$$
$$B ::= \mathbf{b}\, (\mathbf{b}_1 \mid \mathbf{b}_2);$$
$$C ::= (\mathbf{c}_1 \mid \mathbf{c}_2);$$

В ней можно указать три точки ветвления :

$$A ::= \mathbf{a}\, B \underset{\uparrow f_1}{\ } C;$$
$$B ::= \mathbf{b} \underset{\uparrow f_2}{\ } (\mathbf{b}_1 \mid \mathbf{b}_2);$$
$$C ::= \underset{\uparrow f_3}{\ } (\mathbf{c}_1 \mid \mathbf{c}_2);$$

Множество локальных ситуаций состоит из шести элементов:

$$\{ (f_1, \mathbf{c}_1), (f_1, \mathbf{c}_2), (f_2, \mathbf{b}_1), (f_2, \mathbf{b}_2), (f_3, \mathbf{c}_1), (f_3, \mathbf{c}_2) \}$$

Распишем вывод для предложения **a b** $\mathbf{b}_1$ $\mathbf{c}_1$:

$$A \to \mathbf{a}\, B\, C \to \mathbf{a}\, \mathbf{b}\, (\mathbf{b}_1 \mid \mathbf{b}_2)(\mathbf{c}_1 \mid \mathbf{c}_2) \to \mathbf{a}\, \mathbf{b}\, \mathbf{b}_1\, \mathbf{c}_1.$$

Легко заметить, что в нем встречаются три точки ветвления нашей BNF:

Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

$$A \to \mathbf{a}\ B \quad C \to \mathbf{a}\ \mathbf{b} \quad (\mathbf{b}_1 \mid \mathbf{b}_2) \quad (\mathbf{c}_1 \mid \mathbf{c}_2) \to \mathbf{a}\ \mathbf{b}\ \mathbf{b}_1\ \mathbf{c}_1.$$
$$\uparrow f_1 \qquad \uparrow f_2 \qquad \uparrow f_3$$

Поэтому можно указать три локальных ситуации, покрытых предложением :

$$A \to \mathbf{a}\ B \qquad C \to \mathbf{a}\ \mathbf{b} \qquad (\mathbf{b}_1 \mid \mathbf{b}_2) \qquad (\mathbf{c}_1 \mid \mathbf{c}_2) \to \mathbf{a}\ \mathbf{b}\ \mathbf{b}_1\ \mathbf{c}_1.$$
$$(f_1, \uparrow \mathbf{c}_1) \qquad (f_2, \uparrow \mathbf{b}_1) \qquad (f_3, \uparrow \mathbf{c}_1)$$

Тестовый набор, удовлетворяющий нашему критерию, выглядит так:

a b b₁ c₁;
a b b₂ c₁;
a b b₁ c₂.

3. Практические результаты

Нами разработан и реализован генератор тестового набора, отвечающего описанной выше метрике.

Характеристики генератора: память не является критическим фактором; скорость генерации на P-III-450MHz составила 60 тестов в секунду. Для удобства использования генератора введены некоторые параметры, позволяющие строить тестовые последовательности, отвечающие более слабым критериям.

Генератор используется для построения тестов в промышленной разработке компиляторов.

Следует отметить, что в современной практике автоматизированной разработки парсеров в основоном стараются свести грамматику к виду LL(1). В тех местах, где грамматика не укладывается в такой вид, дописывают ручные компоненты анализатора, например, процедуры “LookAHead” в инструменте JavaCC [1], обеспечивающие «заглядываение вперед».

Для грамматик языков Java и его спецификационного расширения J@va [2, 3] был сгенерирован набор тестов (около 170 тысяч). При прогоне построенного набора было выявлено восемь ошибок. Все они были в ручных компонентах.

Для грамматики языка mpC [4, 5] при различных значениях параметров генератора было получено два набора тестов – «маленький» (около 30 тысяч предложений) и «большой» (почти 1.4 миллиона

Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

предложений). В результате прогона полученных тестов было обнаружено двенадцать ошибок в парсере и семантическом анализаторе компилятора с языка mpC, которые приводили к аварийному завершению или зацикливанию. Интересно отметить, что все ошибки, которые были найдены с помощью «большого» набора тестов, также обнаруживались и с помощью «маленького» набора.

4. Направления дальнейшей работы

Для второй задачи, именно, задачи построения «неправильного» тестового набора, традиционен следующий подход: неправильные тесты строят на основе правильных путем некоторых мутаций. Основной проблемой при этом является обеспечение гарантии того, что полученные тесты действительно не принадлежат целевому языку.

Кроме тестирования парсеров, большой интерес представляет также разработка методов тестирования других компонентов инструментов для работы с текстами на формальном языке, таких как:

- парсер с сообщениями об ошибках;
- tree builder, т.е. парсер, возвращающий дерево разбора входного предложения;
- анализатор статической семантики.

В настоящее время нами ведутся исследования в этих направлениях.

Приложение А. Форма Бэкуса-Наура

BNF, используемая в данной работе, имеет следующий вид:

grammar ::= (*rule*)+ ;
rule ::= *id* **::=** *expression* **;** ;
expression ::= *term* (| *term*)* ;
term ::= (*factor*)+ ;
factor ::= *id*
| <LEXEME_TEXT>
| <LEXEME_NAME>
| **(** *expression* **)** (* | + | **?**)?
;
id ::= <ID> ;

Опубликовано в: А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Тестирование парсеров текстов на формальных языках. // Программные системы и инструменты (тематический сборник факультета ВМиК МГУ), 2001, № 2, 150–156.

Литература.

1. <http://www.metamata.com/JavaCC>
2. <http://www.ispras.ru/~RedVerst/RedVerst/White Papers/Java Specification Extension for Automated Test Development/Main.html>
3. I. B. Bourdonov et al. “Java Specification Extension for Automated Test Development”. In proceedings of the Fourth A. P. Ershov International Conference, PSI’2001
4. http://www.ispras.ru/~mpc, “The mpC Programming Language Specification”, The Institute for System Programming of Russian Academy of Science.
5. “The mpC Programming Language Specification”, The Institute for System Programming of Russian Academy of Science.