Реализация поддержки диалектов в YaccConstructor/YARD

Курсовая работа Никоновой О.А., 445 группа

Научный руководитель: Кириленко Я.А.

Оглавление

Введение

Диалект - это вариация или расширение языка программирования. Так же диалектами в общем случае можно считать и более поздние версии языка, в чем-то теряющие совместимость с более ранними.

Проблема диалектов всегда была актуальна в сфере реинжиниринга. Особенно ярко она видна на примере старых языков программирования, таких как Cobol и Fortan, у которых существует множество несовместимых диалектов. Но те же проблемы присущи и вполне современным языкам. Существуют различные диалекты SQL, разные версии многих языков тоже можно рассматривать в этой роли.

Вопрос упрощения работы с диалектами вполне закономерен. Грамматики родственных языков обычно имеют большое множество общих правил и различаются лишь небольшим набором. Отсюда естественным образом вытекает задача оптимизации.

Решение данной задачи актуально как на уровне языка описания грамматик, так и на уровне инструментов реинжиниринга и генераторов синтаксических анализаторов. В рамках данной курсовой работы были сделаны попытки решить этот вопрос для конкретно взятого языка и инструмента.

Постановка задачи

Задачей данной курсовой работы являлось:

- Разработка подходов к поддержке диалектов в инструментеYaccConstructor для языка описания грамматик YARD

- Реализация механизмов, реализующих данные подходы

Основными требованиями к решению являлись:

- Возможность синхронного изменения родственных частей грамматик всех диалектов как основа одновременной поддержки семейства языков

- Независимость от степени сходства между диалектами

Описание решения

YaccConstructor

YaccConstructor - это инструмент для построения синтаксических анализаторов под платформу .NET. Он позволяет задавать грамматику, используя различные языки описания. Затем она транслируется во внутреннее представление и определенным образом преобразовывается. Это позволяет применить к грамматике некоторый генератор, результатом которого будет либо еще одна преобразованная грамматика, либо синтаксический анализатор.

YaccConstructor имеет модульную архитектуру, в основе которой лежат следующие элементы:

- Фронтенд - парсер входной грамматики для опредеденного языка описания грамматик
- Внутреннее представление грамматики - промежуточное звено между фронтендами и генераторами. На этом этапе могут проводиться преобразования над грамматикой с целью ее упрощения
- Генератор (бэкенд) произоводит завершающий этап обработки грамматики.Это может быть как генерация соответствующего синтаксического анализатора, так и просто вывод грамматики в определенном виде

YARD

YARD - это выразительный предметно-ориентированный язык описания грамматик. Он является языком, используемым по умолчанию в YaccConstructor.

YARD поддерживает EBNF и позволяет использовать метаправила. В рамках данной работы в YARD была добавлена возможность условной генерации.

Директивы #if #else #endif в YARD

Первым этапом поддержки диалектов была выбрана поддержка директив #if .. #else .. #endif в языке YARD. В синтаксис языка были внесены изменения, разрешающие оборачивать одно и более грамматических правил в конструкции, позволяющие изменять набор правил грамматики. В основу были взяты классические условные выражения, в качестве условия в которых используется строковый символ.

Пример применения конструкции #if .. #else .. #endif :

```
+s:
#if first
 A
#elif second
 B
#else
 C
#endif
```

Поддержка этой функцианальности в инструмента YaccConstrutor реализована на уровне фронтенда YARD. Список символов указывается при запуске из командной строки с помощью ключа -D. После этого на уровне лексера происходит препроцессинг грамматики, фильтрующий правила.

Данная функциональность повышает выразительную способность языка YARD. Но при всей ее универсальности наиболее интересно ее использование для упрощения работы с диалектами.

Подобные условные конструкции позволяют в удобной форме записывать в одном файле грамматики сразу нескольких диалектов. В блоки условных выражений можно выносить как целые правила, так и их части, например отдельные альтернативы.

Механизм передачи опций в YaccConstructor

В то время как функциональность, добавленная на первом этапе, затрагивает в первую очередь начальный этап обработки грамматики, механизм, разработанный на втором этапе, затрагивает все уровни работы инструмента YaccConstructor.

Идея состоит в создании механизма, позволющего связывать правила грамматики с произвольным набором атрибутов (опций) и передавать эти данные насквозь для использования при создании генератора.

Это потребовало расширения языка YARD еще одной директивой #set, дающей возможность указывать для правила список опций вида “атрибут-значение”. Также это естественным образом повлекло изменения во фронтеде для YARD.

В результате после преобразования грамматики ко внутреннему представлению YaccConstructor в объект с ее описанием добавляется структура данных, содержащая указанные опции для каждого правила грамматики.

Пример использования диркективы #set:

```
+expr :
a=expr ADD b=expr {a + b}
| a=expr SUB b=expr {a - b}
| f=fact {f};

fact : fact1 | fact2;

fact1 :
#set dialect = "first"
a=fact1 MUL b=fact1 {a * b}
| a=fact1 DIV b=fact1 {a / b}
| n=num {n};

fact2 :
#set dialect = "second"
a=fact2 MUL b=fact2 {a * b}
| a=fact2 COLON b=fact2 {a : b}
| n=num {n};

num : A {3} | B {5};
```

Механизм передачи опции может использован для различных целей, но основной причиной его добавления в YaccConstructor были возможности его использования для поддержки диалектов. Этот механизм позволяет добавлять правилам атрибут, указывающий к какому диалекту (или диалектам) принадлежит правило, и делать эту информацию доступной для генератора.

Подсчет статистики использования диалектов

Примером использования описанного в предыдущем разделе механизма для поддержки диалектов является задача посчета статистики использования правил.

Может быть удобно иметь грамматику языка, являющего объединением нескольких диалектов. Таким образом анализатор, сгенерированный по такой грамматике, может распознавать текст на любом из диалектов, или даже смесь из нескольких языков. Задачей является при анализе определять, к какому именно диалекту принадлежит входной текст.

Механизм передачи опций, описанный в предыдущем пункте, позволяет генерировать анализаторы, посчитывающие статистику использования правил, принадлежащим к определенным диалектам. На основе такой статистики можно делать вывод о принадлежности входного текста к конкретному языку.

В качестве основы для примера подобного поведения был использован генератор RNGLR, , реализованый Дмитрием Авдюхиным в рамках его курсовой работы.

Заключение

Итоги

Таким образом, в рамках курсовой работы был разработан ряд средств для поддержки диалектов в YaccConstructor/YARD.

Первое средство - механизм использования директив #if .. #else, который позволяет объединять описания грамматики целого семейства диалектов в единое целое. Это существенно упрощает поддержку грамматик. Также этот механизм позволяет с меньшими затратами создавать грамматику для новых диалектов семейства.

При этом даннный механизм прозрачен для генератора, поэтому его использование не сказывается на производительности генерируемого анализатора.

Второе средство - механизм передачи в генератор опций, связанных с грамматическими правилами, позволяет генерировать анализаторы с учетом информации о диалектах. При использовании грамматики языка, являющего объединением нескольких диалектов, становится возможным создание генераторов, результатом работы которых становятся анализаторы, способные определять диалект обрабатываемых текстов.

Перспективы развития

Наиболее интересным направлением развития для данной темы является разработка генератора, более содержательно использующего знания о принадлежности правил к диалектам.

Одной из возможных задач для дальнейшей работы в этой области является использование механизма опций для генерации по грамматике объединения нескольких диалектов анализатора для конкретного языка, не уступающего по производительности аналогичному анализатору, сгенерированному по грамматике конкретного диалекта.

Список литературы

1. Ralf Lämmel, Chris Verhoef “Cracking the 500-Language Problem”

2. Massimiliano Di Penta, Kunal Taneja “Towards the Automatic Evolution of Reengineering Tools”

3. Бреслав А.А. “Средства повторного использования формальных грамматик и их применение для создания диалектов”, 2009

4. Бреслав А.А. “Применение принципов MDD и аспектно-ориентированного программирования к разработке ПО, связанного с формальными грамматиками”, 2008

5. Чемоданов И.С. “Генератор синтаксических анализаторов для решения задач автоматизированного реинжиниринга программ”, 2007

6. Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д. “Компиляторы”

7. Aho A.V., Johnson S.C. “LR Parsing”

8. Elizabeth Scott, Adrian Johnstone “Right Nulled GLR Parsers”, 2006

9. Don Syme “Expert F# 2.0”

10. http://code.google.com/p/recursive-ascent/ (сайт разработки инструмента YaccConstructor)