Санкт-Петербургский Государственный
Университет
Математико-механический факультет

Кафедра системного программирования

GLR-анализ

Дипломная работа студента 361 группы
Григорьева Семёна Вячеславовича

Научный руководитель	..... /подпись/	к.ф.-м.н. А.С. Лукичев
Рецензент	..... /подпись/	??? ???
“Допустить к защите” заведующий кафедрой	..... /подпись/	д.ф.-м.н., проф. А.Н. Терехов

Санкт-Петербург
2010

Содержание

Содержание

1 Введение

Задачи автоматизированного реинжиниринга программ выдвигают особые требования к генераторам синтаксических анализаторов.

Для устаревшего языка сложно (а зачастую и невозможно) задать однозначную контекстно-свободную грамматику. Необходимо существенно преобразовать его спецификацию, которая приводится в документации, чтобы получить такую грамматику, но при этом она перестает быть сопровождаемой [1]. Поэтому устаревший язык обычно задается с помощью неоднозначной контекстно-свободной грамматики.

При поддержке нескольких диалектов языка необходима возможность лёгкой трансформации грамматики. Однако,зачастую, изменение одного правила приводит к появлению десятков конфликтов в грамматике [1], которые необходимо разрешать вручную. Это требует большого количества времени.

Как вариант решения этих задач предлагается использовать GLR грамматики и соответствующие инструменты построения анализаторов [1]. Действительно, GLR-алгоритм разрешает неоднозначности в грамматике на уровне концепции. По этому задание спецификации трансляции становится проще, требует меньше времени. Получившийся код компактнее и сопровождаемее.

Главным достоинством GLR-алгоритма является обработка неоднозначных грамматик. Анализатор, построенный по неоднозначной грамматике с помощью данного алгоритма, в результате разбора строит не единственное дерево, а несколько деревьев - лес, который можно сократить, используя специальные фильтры, а можно ,при задании в одной спецификации нескольких диалектов, вернуть весь лес для дальнейшего выбора нужного дерева/диалекта.

Работу алгоритма GLR можно рассматривать как параллельное исполнение набора LR-анализаторов. При этом данный набор дополняется процедурой управления стеками, оптимизирующей представление стеков путем их «склеивания» и «расклеивания», что позволяет хранить и строить параллельные выводы в рамках одного LR-анализатора, лишь в моменты их различия добавляя параллельный анализатор.

Оказалось, что весьма наглядно такой алгоритм может быть представлен в виде двух взаимно-рекурсивных функций (рекурсивно-восходящий алгоритм, recursive ascent). При этом расклеивание стека получается естественным образом как ветвление в одной из функций, а обратное склеивание может быть реализовано как кэширование результата функции.

Стоит отметить, что по производительности такой анализатор, яв-

ляясь некоторой "надстройкой" над LR-анализатором, незначительно ему уступает. На сегодняшний день в соотношении производительность/класс разбираемых языков GLR-алгоритм выглядит наиболее предпочтительно.

Удобным способом формального определения грамматики, элементов и атрибутов языка программирования является расширенная нормальная форма Бэкуса-Наура. Поэтому инструмент должен работать с расширенными контекстно-свободными грамматиками.

При работе с инструментом пользователь ожидает получить результат описанный в терминах заданной им грамматики. Это выдвигает дополнительные требования к алгоритму. В случае, если входная грамматика была каким-либо образом преобразована, например с целью раскрыть конструкции EBNF, то появляется необходимость в построении "обратного" преобразования. Это преобразование должно "перевести" результат обратно в термины входной грамматики. Такие преобразования требуют дополнительных ресурсов и усложняют инструмент. Поэтому наиболее предпочтительными является алгоритмы, позволяющие обойтись без дополнительных преобразований грамматики.

2 Обзор

Предпочтительным алгоритмам анализа является алгоритм разбора произвольной контекстно-свободной грамматики. Поэтому были рассмотрены инструменты, основанные на этом алгоритме. Важен способ реализации алгоритма, так как существуют несколько альтернатив: алгоритм Томиты (GLR-алгоритм), алгоритм Эрли (Early), рекурсивно-восходящий алгоритм. Так-же следует обратить преобразования грамматики, необходимые для построения анализатора.

В настоящее время существуют следующие инструменты основанные на GLR-алгоритме.

- ASF+SDF [11] (Algebraic Specification Formalism + Syntax Definition Formalism) - генератор с широкими возможностями, но достаточно сложным входным языком. Является SGLR-инструментом (Scannerless, Generalized-LR).

- Bison [12] - развитие инструмента YACC. Все грамматики, созданные для оригинального YACC, будут работать и в Bison. Является одним из самых популярных и совершенных "потомков" YACC. При включении соответствующей опции использует GLR-алгоритм (по умолчанию LALR).

- Elkhound [13] - позиционируется как быстрый и удобный GLR-инструмент, созданный в университете Беркли (США), тем не менее обладает достаточно "бедным" входным языком (например, он не поддерживает конструкций расширенной формы Бэкуса-Наура).

В работе [4] проведён подробный анализ этих инструментов.

Так же был рассмотрен инструмент Jade. Jade это генератор рекурсивно-восходящих LALR(1) парсеров с целевым языком C. Его подробное описание приводится в статье [7]. При реализации данного инструмента появилась проблема объёма кода целевого парсера. Так как при построении детерминированного парсера необходимо генерировать процедуры для каждого состояния, то объём кода быстро растёт, с ростом количества правил в грамматике. Так для языка Java объём кода составляет примерно 4 мегабайта [7]. В Jade эта проблема решается путём создания глобальной структуры(массива состояний), где хранится информация, позволяющая переиспользовать процедуры [7].

2.1 Вычисление атрибутов

При использовании GLR алгоритма выдвигаются дополнительные требования к алгоритму вычисления атрибутов. В качестве атрибута пользователь может указать действие, обладающее побочным эффектом. Так как в момент разбора мы не можем сказать, завершиться ли текущая ветвь удачно, то нельзя совершать действия непосредственно по ходу разбора входной строки, так как в этом случае могут быть совершены лишние действия.

Таким образом

Были рассмотрены два подхода к решению этой проблемы: отложенные вычисления(continuation passing style, CPS) и CPS

дерево вывода - вычисления над деревом

3 Реализация

3.1 Генерация action-кода

3.2 Вычисление атрибутов

4 Заключение

Полученные результаты:

- проведён обзор инструментов построения анализаторов. Выяснено, что предпочтительным алгоритмом анализа является GLR алгоритм;
- изучен рекурсивно-восходящий алгоритм анализа;
- предложен прототип инструмента реализующего рекурсивно-восходящий алгоритм и применимого для работы с неоднозначными грамматиками;

4.1 Свойства прототипа

Предложенный прототип имеет следующие характеристики:

- по однозначной LR-грамматике строится анализатор с линейной сложностью;
- по неоднозначной грамматике строится анализатор, возвращающий все возможные деревья вывода для данной входной цепочки;
- показана возможность поддержки регулярных выражений в правых частях правил;

4.2 Дальнейшее развитие

Задачи, требующие решения:

- задача1;
- задача1;

Список литературы

[1] *Mark G.J. van den Brand, Alex Sellink, Chris Verhoef* Current Parsing Techniques in Software Renovation Considered Harmful.// IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension. - IEEE Computer Society, Washington,1998.

[2] *h*ttp://www.research.microsoft.com/fsharp (дистрибутивы и документация по языку F#)

[3] *I*SO/IEC 14977 : 1996(E)

[4] *Чемоданов И.С.* Генератор синтаксических анализаторов для решения задач автоматизированного реинжиниринга программ.

[5] *Dick Grune, Ceriel Jacobs* PARSING TECHNIQUES A Practical Guide

[6] *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии, инструменты. М:. Издательский дом <Вильямс>2003. 768 с.

[7] *Ronald Veldena* Jade, a recursive ascent LALR(1) parser generator. September 8,1998

[8] *Rene Leermakers* Non-deterministic Recursive Ascent Parsing. Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands.

[9] *Larry Morell, David Middleton* RECURSIVE-ASCENT PARSING. Arkansas Tech University Russellville, Arkansas.

[10] *Lex Augusteijn* Recursive Ascent Parsing (Re: Parsing techniques). lex@prl.philips.nl (Lex Augusteijn) Mon, 10 May 1993 07:03:39 GMT

[11] http://www.cwi.nl/projects/MetaEnv (сайт разработчиков ASF+SDF)

[12] http://www.gnu.org/software/bison (сайт разработчиков Bison)

[13] http://www.cs.berkeley.edu/smcpeak/elkhound (сайт разработчиков Elkhound)