ДЕРЕВО РАЗБОРА ВЫРАЖЕНИЙ АЛГЕБРЫ А

ХАМУЕВ Леонид, ОПРЯ Дмитрий
Coordonator: САРАНЧУК Дориан

Технический Университет Молдовы

Аннотация: *В работе приведены исследования Алгебры А. Описаны технологии для построения дерева разбора. Разработано приложение для автоматизации преобразований выражений Алгебры А в другие языки запросов.*

Ключевые слова: *Сканирование, парсенг, построение абстрактного синтаксического дерева, формальные языки, терминальные символы.*

1. Введение

Одним из теоритических языков управления реляционной моделью является реляционная алгебра, предложенная Э.Коддом в 1970-е годы [1]. Однако реляционная алгебра Кодда является избыточной. Кристофер Дейт и Хью Дарвен предложили альтернативную алгебру отношений под названием Алгебра А [2]. Разработка приложения для преобразования Алгебры А в другие языки запросов является задачей, которою мы себе поставили. Если не вдаваться в подробности языков, то первое, что необходимо сделать, это подумать над тем как, создать такую структуру, по которой было бы легко осуществить преобразование. Данной структурой является построение абстрактного синтаксического дерева разбора. Главной практической ценностью использования Фреймворка Irony является разбор предложений.

Для того что бы выполнить поставленную задачу нужно овладеть знаниями в области формальных языков, терминальных и не терминальных символах, сканирования, парсенга, построения абстрактного синтаксического дерева.

2. Базовые операции Алгебры А

Операция <NOT> производит дополнение *s* для заданного отношения *r*. Тело *s* включает все кортежи, соответствующие этому заголовку и не входящие в тело *r*. Следует пояснить, почему реляционный аналог операции логического отрицания называется здесь операцией реляционного дополнения. Термин «дополнение» полностью соответствует сути операции <NOT>: тело результата операции <NOT> *r* является дополнением *Br* до полного множества кортежей, соответствующих *Hr* [1].

Операция <REMOVE> производит отношение *s*, формируемое путем удаления указанного атрибута A из заданного отношения *r*. Операция эквивалентна взятию проекции *r* на все атрибуты, кроме A. Заголовок *s* получается теоретико-множественным вычитанием из заголовка *r* множества из одного элемента {<A, T>}. Тело *s* состоит из таких кортежей, которые соответствуют заголовку *s*, причем каждый из них является подмножеством некоторого кортежа тела отношения *r* [1].

Операция <RENAME> производит отношение *s*, которое отличается от заданного отношения *r* только именем одного его атрибута, которое изменяется с A на B. Заголовок *s* такой же, как заголовок *r*, за исключением того, что пара <B, T> заменяет пару <A, T>. Тело *s* включает все кортежи тела *r*, но в каждом из этих кортежей триплет <B, T, v> аменяет триплет <A, T, v> [1].

Операция <AND> является реляционной конъюнкцией, в некоторых случаях выдающей в результате отношение *rs*, ранее называвшееся естественным соединением двух заданных отношений *r1* и *r2*. Заголовок *rs* является объединением заголовков *r1* и *r2*. Тело *s* включает каждый кортеж, соответствующий заголовку *s* и являющийся надмножеством некоторого кортежа из тела *r1* и некоторого кортежа из тела *r2* [1].

Операция <OR> является реляционной дизъюнкцией и обобщением того, что ранее называлось объединением. Заголовок *s* есть объединение заголовков *r1* и *r2*. Тело *s* состоит из всех кортежей, соответствующих заголовку *s* и являющихся надмножеством либо некоторого кортежа из тела *r1*, либо некоторого кортежа из тела *r2* [1].

3. Фреймворк Irony

Альтернативой технологии построения грамматики является технология SableCC. Требуется много усилий для того что бы заставить SableCC принимать грамматику. Более того, грамматика получается очень сложной и много времени уходит на дальнейшее обрабатывания приоритета операторов. Если проигнорировать производительность и сосредоточиться на лёгком понимании технологии, а так же на лёгкое управление, то грамматика для парсера будет больше похожа на EBNF [3].

Irony является парсером и имплементируется как внутренний DSL в C# для спецификации близкой к формату EBNF. Код, определённый в теле конструктора, наследован от Irony Grammar Class [4].

```
    var table = TerminalFactory.CreateCSharpIdentifier("table");
var attr_remove = TerminalFactory.CreateCSharpIdentifier("attr_remove");
var attr_rename1 = TerminalFactory.CreateCSharpIdentifier("attr_rename");
```

Эти строки являются инициализацией терминальных символов. Это довольно тривиально, однако несколько вещей стоит отметить. Irony очень богатый Фреймворк, который предоставляет большое количество предварительно подготовленной функциональности. Одна из них это TerminalFactory для создания часто используемых терминальных символов [3].

Интересное начинается после того как мы закончили инициализацию и начинаем установку правил для нетерминальных символов. Irony в значительной степени опирается на перегрузку операторов так что использование наших символов и операторов EBNF мы можем довольно строго описать язык. Так же мы должны специфицировать какой символ является корнем в нашей программе. Этот символ будет определять входную точку для всего языка. В наших нетерминальных символах мы можем определить, в какие узлы они должны быть преобразованы [3].

```
var not = new NonTerminal("not");
var and = new NonTerminal("and");
var or = new NonTerminal("or");
not.Rule = "<not>";
and.Rule = "<and>";
or.Rule = "<or>";
```

4. Абстрактное синтаксическое дерево

Написанное выше представляет реализацию построения абстрактного синтаксического дерева, а далее покажем что представляет из себя дерево разбора (рис. 1).

Абстрактное синтаксическое дерево (АСД) в информатике - это конечное ориентированное дерево в котором вершины сопоставлены с командами языка программирования или запроса, а его листья конечными операндами. Листья в нашем случае представляют собой терминальные символы либо константы. Данные деревья используются в парсерах для представления между деревом разбора и структурой данных. Данная структура в дальнейшем будет использоваться для внутреннего представления и генерации кода [5].

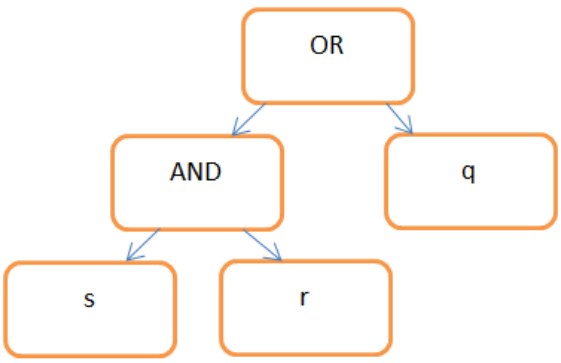

Рисунок 1 – Абстрактное синтаксическое дерево

АСД отличается от дерева разбора тем, что в нём отсутствуют узлы и рёбра для тех синтаксических правил, которые не влияют на семантику программы. Классическим примером такого отсутствия являются группирующие скобки, так как в АСД группировка операндов явно задаётся структурой дерева. Для языка, который описывается контекстно-свободной грамматикой, какими являются почти все языки программирования, создание абстрактного дерева в синтаксическом анализаторе является тривиальной задачей. Большинство правил в грамматике создают новую вершину, а символы в правиле становятся рёбрами. Правила, которые ничего не привносят в АСД такие, например, как группирующие правила, просто заменяются в вершине одним из своих символов. Кроме того, анализатор может создать полное дерево разбора и пройти по нему, удаляя узлы и рёбра, которые не используются в абстрактном синтаксисе чтобы получить АСД [5].

5. Пример построения дерева разбора

В следующем примере будет продемонстрировано каким образом из выражения Алгебры А происходит построение абстрактного синтаксического дерева. Возьмём выражение следующего вида: <not> ((s <or> r) and q). Анализ начинается с операции not (рис. 2). Создаётся новая ячейка, в которую добавляется операция not после добавления операции создаётся ещё одна ячейка и текущей ячейкой становится та, которая была только что создана (рис. 2a). После операции not следует скобка, которая добавляется в текущую ячейку. Следующий символ будет добавлен, так же как и предыдущий (рис. 2b).

Следует заметить, что на рисунке 2c все скобочки отсутствуют. Причинной этого является то, что когда следующим символом является закрывающая скобка, происходит поиск вверх до первой открывающей скобки. Когда скобка найдена, она удаляется, и текущей ячейкой становится та, в которой было удаление. В пустую ячейку записывается операция <and>. Конечное дерево представлено на рисунке 3. В ходе построения дерева иногда происходит создание лишних ячеек, которые после его построения будут удалены. Процесс удаление пустых ячеек осуществляется функцией оптимизации.

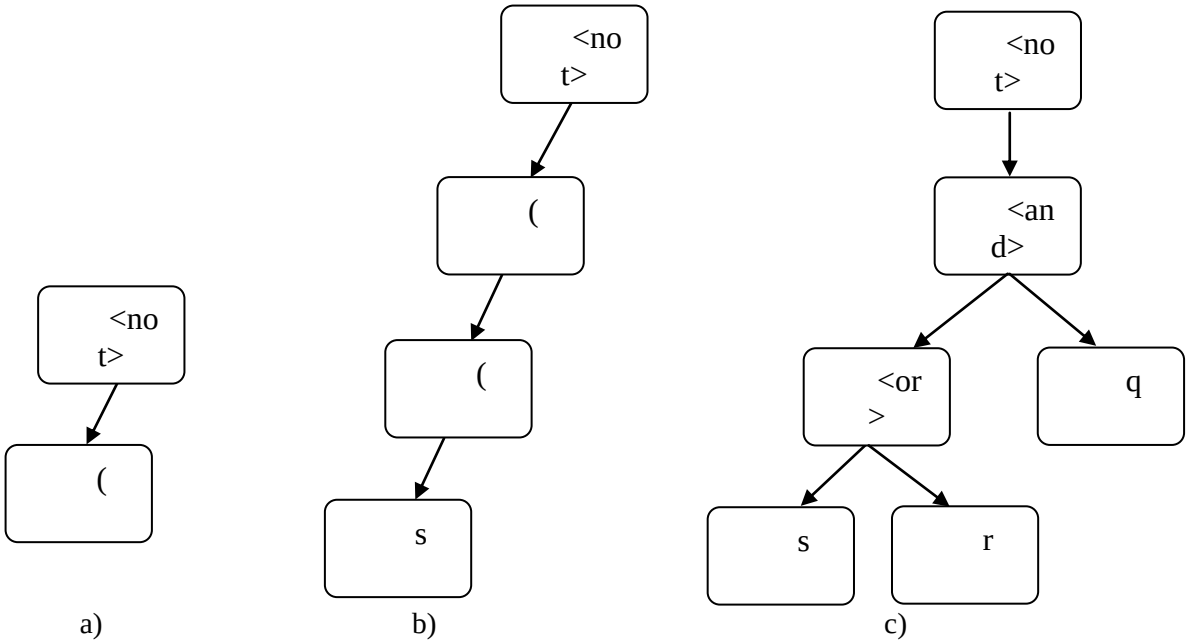

Рисунок 2 – Дерева разбора выражения Алгебры А

Для преобразования выражений алгебры А в другие языки разработано приложение, описанное в [6]. Приложение написано на языке C#. Для разбора грамматики использована библиотека Spart. Данная библиотека позволяет составлять грамматику и правила, которые в процессе могут быть легко изменены. Также библиотека позволяет следить за ходом операции, когда строка проходит через созданную грамматику.

Заключение

Создано приложение по преобразования выражений из Алгебры А в другие языки запросов. Не все преобразования возможны из-за специфики языка. Для того чтобы осуществить преобразование из Алгебры А в другие языки было построено абстрактное синтаксическое дерево. Данное дерево является удобной структурой для последующего формирования из одного языка в другой. Абстрактное синтаксическое дерево используется может использоваться не только для Алгебры А но и для любых других языков. Незаменимым элементом являлся фреймворк Irony, который предоставляет большое количество возможностей по парсингу выражений.

Библиография

1 С. Кузнецов. *Основы баз данных*, Москва, Бином: 2-е изд, 2007.

2 Х. Дарвен, К. Дейт. *Основы будущих систем баз данных*. Янус-К: Третий манифест , 2004.

3 *Writing a calculator in C# using Irony*. Jakob T. Andersen, URL: http://blog.miraclespain.com/archive/2009/Oct-07.html

4 *Irony - .NET Language Implementation Kit*. URL: http://irony.codeplex.com.

5 *Abstract Syntax Trees.* URL: http://www.cse.ohio-state.edu/software/2231/web-sw2/extras/slides/21.Abstract-Syntax-Trees.pdf

6 Саранчук Дориан, Опря Дмитрий, Войков Александр. *Преобразование выражений Алгебры А в различные языки запросов*. Conferinţa tehnico-ştiinţifică a colaboratorilor, doctoranzilor şi studenţilor, Universitatea Tehnică a Moldovei, 22 octombrie 2013, Chişinău, Republica Moldova.