Сравнение возможностей генераторов парсеров при разборе неоднозначных контекстно-свободных грамматик

Над проектом пыхтели:
Буянтуев Александр
Игнатов Николай
Федотова Евгения

Предисловие

Неоднозначные грамматики - это грамматики, для которых существует более чем одно дерево разбора; то есть при попытке разобрать выражение языка программа сталкивается с ситуацией, когда корректны несколько переходов. Так, грамматика S->SS|SSS|a - неоднозначна -- например, строка aaa может быть обработана по-разному.

Цели

Относительно разбора неоднозначных грамматик парсеры делятся на 2 категории: те, что при встрече с неоднозначностью выбрасывают исключение и завершаются, и те, что четко знают, в какую ветку разбора следует идти. Наша задача - сравнить работу парсеров второго вида на примере слов выбранной неоднозначной грамматики и сопоставить умения генераторов парсеров строить деревья вывода.

ANTLR v4

ANTLR (Another Tool for Language Recognition) - генератор парсеров, который позволяет преобразовать контекстно-свободную грамматику в программу на одном из основных языков программирования. Изначально был написан на Java. Удобен своей кроссплатформенностью и наличием своих сред разработки/плагинов для известных. ANTRL v4 использует Adaptive LL(*)-парсер, более ранние версии использовали LL(*)

ANTLR v4

Для работы с ANTLR достаточно просто описать грамматику в файле с расширением .g4, остальные необходимые файлы ANTLR сгенерирует сам. При необходимости, пользователь может изменить Visitor под свои нужды - например, чтобы считать свои функции во время парсинга.

ANTLR использует Adaptive LL(*) - он сочетает в себе простоту и эффективность LL-парсеров, а механизм принятия решений схож с GLR-парсером. ALL(*) в теории работает за O(n^4)

Yacc

Yacc ("Yet Another Compiler Compiler") -- широко известный генератор парсеров, разбирающий контекстно-свободные грамматики, описанные в форме последовательного определения одних синтаксических конструкций через другие.

В своей основе использует LALR-парсер;

превращает спецификации в программу на C;

задействует модуль PLY.

Yacc

Парсер, созданный Yacc-ом, состоит из машины конечных состояний со стеком. Текущее состояние - всегда на вершине стека. Последовательно считываются входные токены.

Yacc в обязательном порядке требует наличие лексера, часто в роли генераторов лексических анализаторов выступают lex или flex.

В лексере в словаре tokens задается набор токенов, затем описывающихся регулярными выражениями в функциях, названия которых имеют префикс 't_'.

В парсерах в функциях, имеющих в имени префикс 'p_', описаны правила грамматики, заданные в форме Бэкуса-Нура -- то есть несколькими конструкциями вида:
<expr> : <seq 1> | <seq 2> | ... | <seq n>

Yacc

Особенность Yacc-а заключается в удобном обращении к элементам описываемого выражения:

```
def p_expr_plus(p):
    'expr : expr PLUS term'
    p[0] = p[1] + p[3]
```

Доступ происходит по индексу слова в строке, в данном примере вернувшееся из рекурсии значение ячеек p[1] и p[3] записывается в expr p[0].

Yacc отлично выявляет неоднозначности при построении алгоритма разбора. Исходно все правила делятся на 2 типа: свертки и переносы. Когда в разборе выражения возникает конфликт тип1-тип2, генератор делает выбор в соответствии с зафиксированными правилами разрешения неоднозначностей.

Yacc

Еще **о преимуществах**:

- высокая скорость работы
- возможность устанавливать различные флаги компиляции
- понятный вывод ошибок -- подробный отчет в файле parser.out + генерация модулем PLY таблицы parsetable.py

Главный **недостаток**:

- отсутсвие автоматического построения дерева разбора

Итого, Yacc стоит использовать в случае необходимости быстрого разбора выражения, удобного обращения к синтаксическим конструкциям и если пользователю не требуется наглядное представление синтаксического дерева.

Lark

Lark - библиотека для Python, позволяющая работать с несколькими парсерами при разборе грамматик (Earley, Lalr(1), CYK) и несколькими лексерами. Мы же рассмотрим Earley

Lark (Earley)

Earley - это так называемый chart parser (анализатор диаграмм). Такие парсеры используют подход динамического программирования и отлично подходят для неоднозначных грамматик. Earley умеет разбирать все контекстно-свободные грамматики. Асимптотика работы в худшем случае (для неоднозначных грамматик) - O(|G|n^3), где n - длина разбираемой строчки, G - размер грамматики. Lark поддерживает построение дерева вывода для грамматик (строит одно из возможных деревьев, полученных после работы Earley). Разрешение неоднозначностей держится на сортировке правил от более приоритетных к менее приоритетным. На практике довольно хорошее время работы для неоднозначных грамматик (100 букв a для грамматики S->SS|SSS|a разбираются за 4 секунды). Но для других грамматик есть парсеры и побыстрее...

Lark (Lalr(1))

Не умеет работать с неоднозначностями:

lark.exceptions.GrammarError: Reduce/Reduce collision in Terminal(u'LETTER') between the following rules:
- <expr : expr expr expr>
- <expr : expr expr>

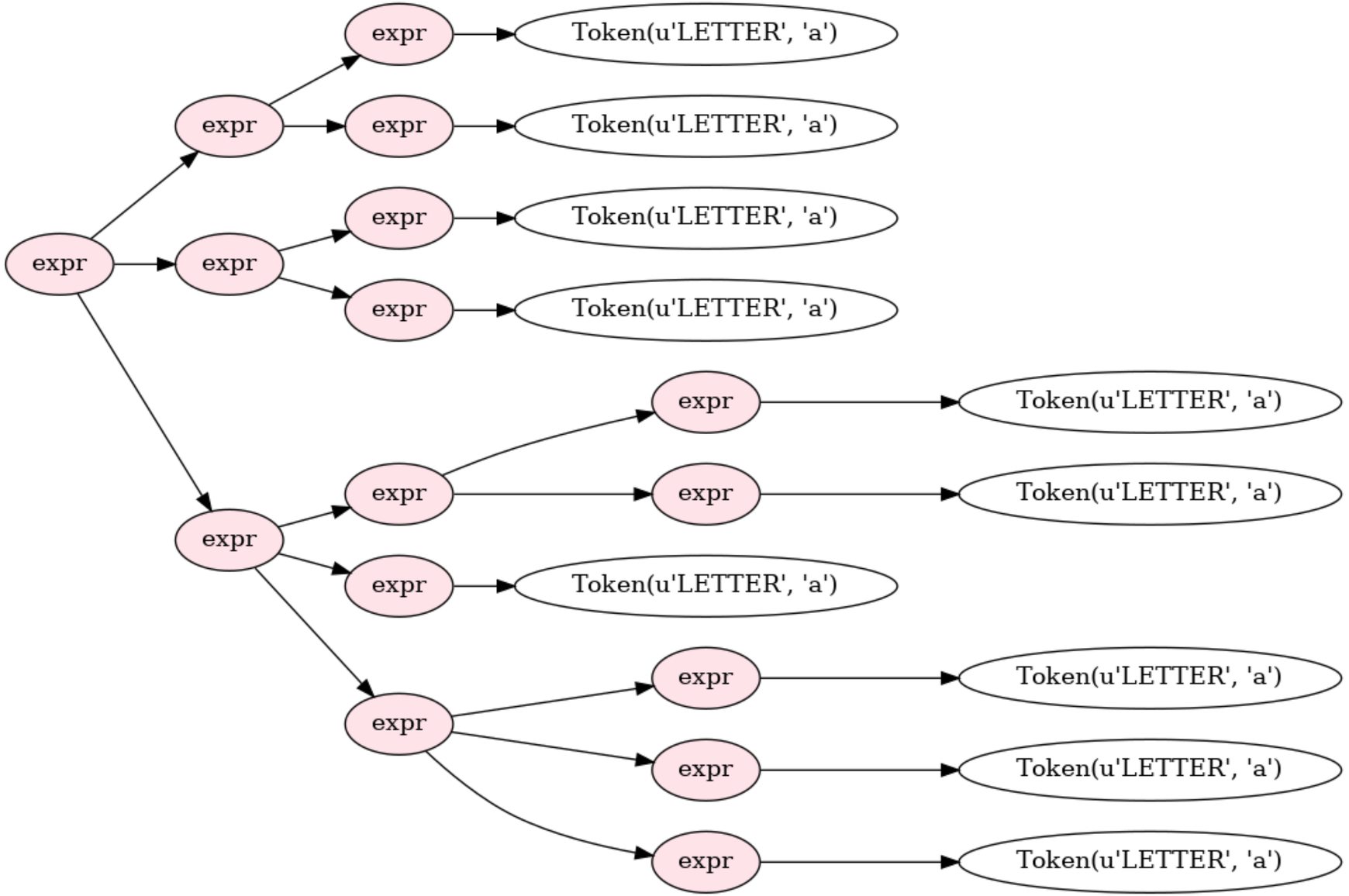
При этом на более простых грамматиках (S->aS|a) очень эффективен:

number of a: 10
 Earley parser --> 0.00121402740479 seconds
Lalr parser --> 0.00028395652771 seconds

number of a: 100
 Earley parser --> 0.0651278495789 seconds
Lalr parser --> 0.00073504447937 seconds

number of a: 1000
 Earley parser --> 24.207887888 seconds
Lalr parser --> 0.00907111167908 seconds

Дерево вывода для lark (Earley), строчки aaaaaaaaaa и грамматики S->SS|SSS|a



Parglare

Parglare - генератор парсеров, использующий LR и GLR алгоритмы (в зависимости от грамматики).

GLR-парсер (Genergalized LR) - расширенный алгоритм LR-парсера, предназначенный для работы с неоднозначными грамматиками. Принцип работы схож с LR-алгоритмом, но GLR обрабатывает все возможные трактовки входной последовательности при помощи поиска в ширину. В худшем случае алгоритм имеет сложность O(n^3)

Parglare

Для работы с Parglare достаточно описать грамматику в .pg файле и написать простейший парсер на языке Python.

Преимущества

- GLR-алгоритм
- Нет необходимости писать lexer
- Встроенная визуализация деревьев
- Подробный Debug режим

Недостатки

- Отсутствие парсинга без построения дерева разбора
- Сложность при работе с некоторыми встроенными функциями

Parglare помогает проверить свою грамматику на неоднозначности и подробно изучить все ее недостатки

Результаты сравнения генераторов парсеров

Время работы парсеров на грамматике S->SS|SSS|a

(в качестве парсера для Lark взят Earley)

Размер входных данных	ANTLR4	Lark	Parglare	Yacc
1	0.003	0.0001	0.00035	0.00008
10	0.263	0.0077	0.014	0.00014
20	33.994	0.0418	0.264	0.00025
30	TL	0.1155	1.911	0.00034
40	TL	0.2598	9.906	0.00038
50	TL	0.4973	18.399	0.00047
60	TL	0.8868	60.883	0.00056
70	TL	1.3198	162.57	0.00065
80	TL	2.0565	182.499	0.00074
90	TL	2.8626	272.041	0.00139
100	TL	4.1458	563.266	0.00178

Встроенное графическое построение деревьев

Генератор	Графическое построение деревьев
ANTLR4	Дерево, только при сборке под Java
Lark	Дерево
Parglare	Дерево (LR) + Лес (GLR)
Yacc	Нет

Умение разрешать неоднозначности

Генератор	Умение разрешать неоднозначности
ANTLR4	Неясно. Четко фиксированных данных и правил нет.
Lark	Зависит от выбора парсера. `Larl (1)` не умеет; `Earley` хорошо и быстро разбирает неоднозначности; `CYK` тоже справляется, но медленнее.
Parglare	Строит все возможные варианты разбора строки.
Yacc	Отлично справляется с неоднозначностями. Правила выбора ветки разбора зафиксированы на уровне языка.

Какой генератор выбрать пользователю

При выборе генератора парсеров пользователю стоит ориентироваться на ситуацию и свои цели. Как видно из анализа двух рассмотренных выше таблиц, генератор `Yacc` не строит деревья разбора, соответственно, это отражается на времени работы программы -- `Yacc` отрабатывает быстрее других генераторов. `ANTLR4` на строке длиной больше 20-ти символов уже работает слишком долго, однако строит подробные деревья разбора; `Parglare` ведет себя похожим образом, но умеет не слишком долго обрабатывать строки длиной уже до 90 символов. Если хочется и относительно небольшое время работы, и наглядное представление парсинга -- наверное, предпочтение следует отдать генератору `Lark`.

Спасибо за внимание :)