

Синтаксические (и лексические) анализаторы в современной разработке

Спикер: Волкова Юлия



Проходит в рамках

phd 12 Positive
Hack Days



Iuliia Volkova

xnuinside

Developer <https://twitter.com/xnuinside> |
<https://www.linkedin.com/in/xnuinside>

Edit profile

🔗 126 followers · 10 following

✉ xnuinside@gmail.com

🔗 <https://medium.com/@xnuinside>

🐦 [@xnuinside](https://twitter.com/xnuinside)




Агенда

- Бизнес-задачи, в которых требуются синтаксические анализаторы
- О том, что это вообще такое
- С чего начать
- Какие библиотеки есть в Python
- Посмотрим на примеры кода

Бизнес-задачи

1) Data-Platforms -> Query - как user input

Last refreshed at 9:21:48 AM ☒ Auto refresh 

SQL Text

```
1 select exp(sum(j))
2 from x join y using (i)
3 where j > 300
4 and i < (
5     select avg(j) from x
6 );
```

Select SQL 

Любезно взято с: <https://docs.snowflake.com/en/user-guide/ui-query-profile>

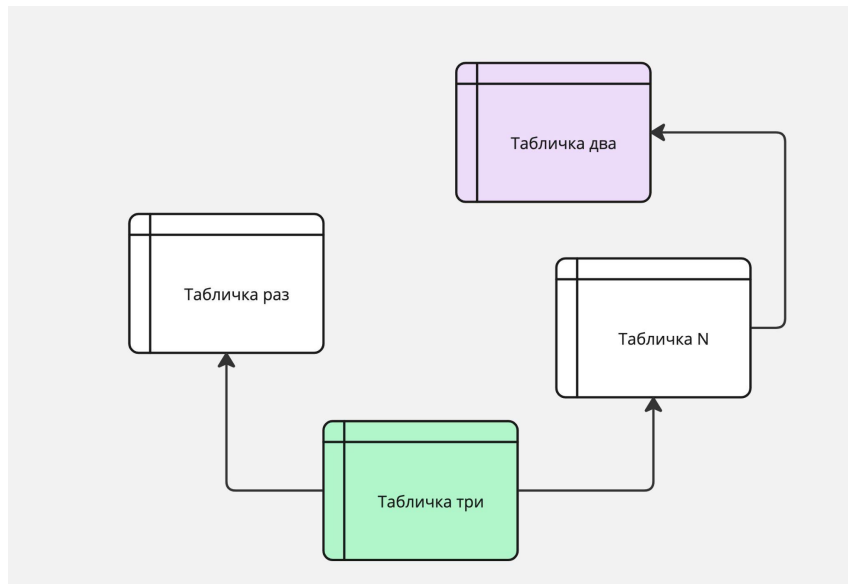
Бизнес-задачи

1) Зачем нам анализировать пользовательский ввод:

- Мы хотим давать подсказки пользователю на основе ввода (например, что ошибка в имени таблицы)
- Мы хотим подсвечивать *syntax errors*
- Мы хотим подсвечивать куски квери на оптимизацию
- Мы хотим подсказывать более оптимальные таблицы для выборки данных
- И т.д.

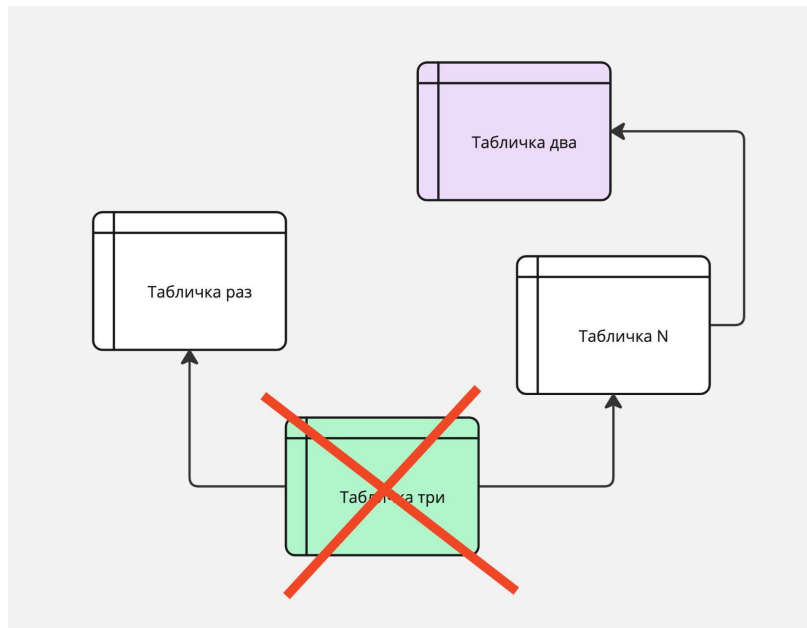
Бизнес-задачи

2) Data Warehouses -> Валидации операций



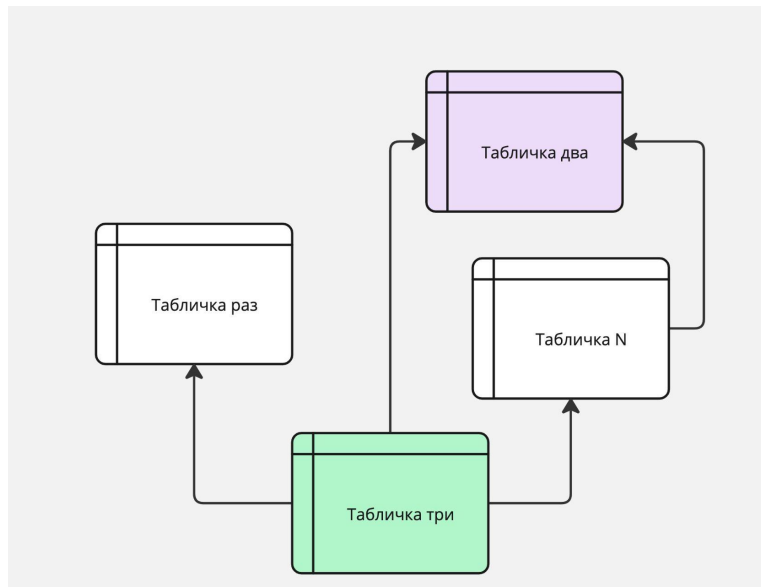
Бизнес-задачи

2) Data Warehouses -> Валидации операций



Бизнес-задачи

2) Data Warehouses -> Валидации операций



Бизнес-задачи

2) Data Warehouses -> Валидации операций:

- Можно ли удалить эту таблицу?
- Можно ли удалить эту колонку?
- Можно ли добавить в зависимость вот эту таблицу (не будет ли у нас рекурсии, которую мы вдруг не умеем разруливать?)
- Можно ли вообще создать эту таблицу или там запрещенные перс данные?
- И т.д.

Бизнес-задачи

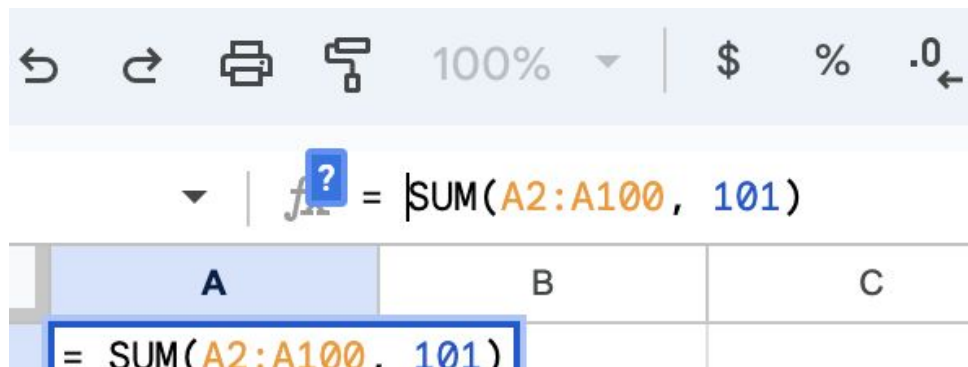
2) Data Warehouses -> Таблица = DDL

```
CREATE TABLE IF NOT EXISTS default.salesorderdetail(  
    SalesOrderID int,  
    ProductID int,  
    OrderQty int,  
    LineTotal decimal  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
COLLECTION ITEMS TERMINATED BY '\\002'  
MAP KEYS TERMINATED BY '\\003'  
STORED AS TEXTFILE
```

Бизнес-задачи

3) Ввод формул, калькуляции через ввод

О, сколько нам экселей дивных дарят бухгалтерские и менеджерские SaaS-решения...



4) Специальные Query-языки

Insert JQL Query

Runs As Admin

JQL Query `project = "SAFe Team A" AND status = "In Progress"`

Limit

Maximum number of issues to insert (leave empty for no limit)

Apply Cancel

Бизнес-задачи

5) Автоматическая обработка любого ввода / любой текстовой информации:

- Специфический формат данных
- Валидация пользовательского ввода
- Транспайлеры из одного формата в другой (чтобы что-то затранспайлить – нам надо это сначала проанализировать/распарсить)
- Линтеры
- Любой ввод, где мы ищем команды/смыслы (например, чат-боты)
- И т.д.

Классическая задача

6) Компиляторы и интерпретаторы

Классическая задача

- *Компиляторы и интерпретаторы*



В чем сложность

Есть задача, начинаем гуглить, а там ...

Терминалы, нетерминалы, LL, Labelled BNF, BNF, EBNF, LR, SLR, токены, грамматики, AST, PEG, lex, yacc, antlr...

“Левая рекурсия: $NT ::= NT x \mid y$, где x, y — произвольные строки терминалов/не терминалов, но y не начинается с NT ”

(с) из статьи “Парсеры, обработка текста. Просто о сложном. CFG, BNF, LL(k), LR(k), PEG и другие страшные слова”

Историческая справка

Зарождение и активное развитие синтаксических анализаторов, формальных языков и грамматик приходится на **1950-е – 1960-е годы**

Ноам Хомский (Noam Chomsky)

“отец” науки о формальных языках и классификации грамматик
выпускает в **1957 году книгу** «Синтаксические структуры», создает
Иерархию Хомского (классификация грамматик формальных языков)

Историческая справка

Создатели **BNF** (*Backus normal form / Backus–Naur form*) «нормальной формы Бэкуса» или «форма Бэкуса – Наура» (в 1964 году Дональд Кнут предложил переименовать расшифровку):

- **Джон Бэкус (John Backus)** – создатель первого языка программирования высокого уровня «**Фортран**» (первая редакция BNF – 1949, Фортран – 1957)
- **Питер Наур (Peter Naur)** – участвовал в создании ALGOL 60

EBNF (Extended Backus–Naur Form) - 1981 год:

- **Никлаус Вирт (Niklaus Wirth)** – создатель *Pascal*, *Modula-2*, *Oberon* - разработал **EBNF**

PEG (Parsing Expression Grammar) - 2004 год:

- *Bryan Ford* (<https://bford.info/>), Lead of Decentralized/Distributed Systems (*DEDIS*) lab at the Swiss Federal Institute of Technology in Lausanne (*EPFL*)

У каждого языка есть:

- *Алфавит (набор символов)*
- *Правила составления выражений (грамматики)*

Синтаксические парсеры:

- 1) *Инструменты, используемые для анализа и разбора текстов, составленных на естественных или формальных языках*
- 2) *Используются для определения структуры языков и связей между элементами*

Грамматика - способ описания языка

```
<expression> ::= <expression> + <term>
| <expression> - <term>
| <term>
<term> ::= <term> * <factor>
| <term> / <factor>
| <factor>
<factor> ::= <primary> ^ <factor>
| <primary>
<primary> ::= <primary>
| <element>
```

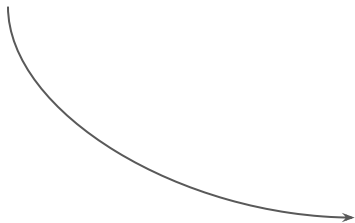
Подробнее -> **Теория формальных языков**

(раздел Теоретической информатики и математики)

Разберем на примере

Задача

Запросы в некую
систему аналитики



```
> clients: 23.05
...
> clients: 2024-12-01
...
> orders: 2017
...
> cats: 2019-10-10
```

Ожидаемый результат

Вот это: "clients: 23.05"



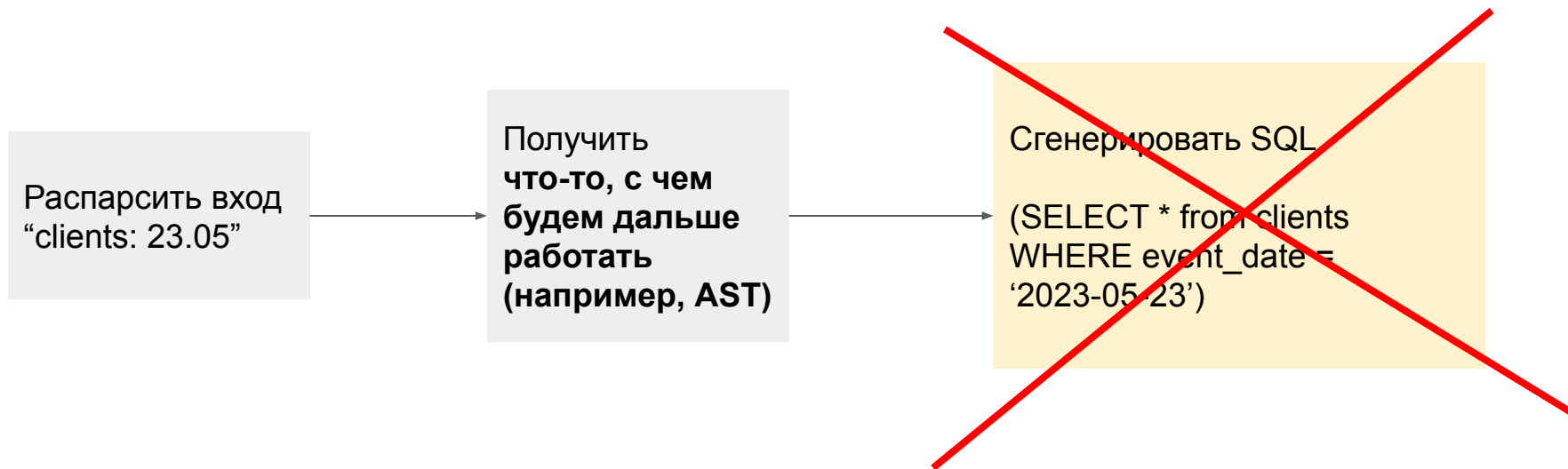
Превратить в вот это:

```
SELECT * from clients WHERE  
event_date = '2023-05-23'
```

Шаги получения



Шаги получения



Смотрим на инпут

```
> clients: 23.05  
...  
> clients: 2024-12-01  
...  
> orders: 2017  
...  
> cats: 2019-10-10
```

→ ***`${key}: ${value}`***

Токены и грамматики

`${key}: ${value}` - выражение раз

А потом сказали, хотим еще и так:

```
> clients: 2024-12-01 / 2025-12-01
...
> cats & orders: 2019-10-10
...
> cats - orders: 2024-12-01 / 2025-12-01
...
```

Токены и грамматики

```
> clients: 2024-12-01 / 2025-12-01
```

■ ■ ■

```
> cats & orders: 2019-10-10
```

■ ■ ■

```
> cats - orders: 2024-12-01 / 2025-12-01
```

$\{key\}$ “-” $\{key\}$: $\{value\}$ / $\{value\}$

$\{key\}$ “-” $\{key\}$: $\{value\}$

$\{key\}$ “&” $\{key\}: \{value\}$

Грамматики

Грамматика в формальных языках – это формальное **описание правил**, которые определяют, **каким образом можно строить выражения в данном языке**.

```
clients: 2024-12-01 / 2025-12-01
```

```
& 2025-12-01: clients
```



Прежде чем начинать писать свой парсер

* все утверждения максимально субъективны

- Готовое решение всегда выгоднее самописного, если его можно допилить
- Чем больше грамматика языка, тем больше размер проблемы, парсить разные диалекты DDL SQL – уже сложно
- Прежде чем выбирать библиотеку для генерации парсера – посмотрите внимательно, как выглядит определение грамматик, трансформеров AST и тд

Прежде чем начинать писать свой парсер

* все утверждения максимально субъективны

- Кто будет после вас поддерживать этот парсер?

Выбирайте максимально понятный инструмент для генерации или оставляйте подробные инструкции

- **AST** – как результат работы парсера, всегда понятнее и проще в дальнейшей работе
- Если вам достаточно пары регулярок – пользуйтесь ими

Прежде чем начинать писать свой парсер

* все утверждения максимально субъективны

- Если вам нужен SQL-парсер для определенного диалекта – гляньте в открытом доступе ANTLR (LL) грамматики, возможно, или вы найдете нужную вам, или что-то будет проще дописать, чем писать с 0 <https://github.com/antlr/grammars-v4>

spass	[build] Fix of #3269 (Remove all mentions of "Antlr4cs") (#3270)	2 months ago
sql	[InformixSql] Add support of drop type/view (#3345)	3 days ago
stacktrace	[build] Fix of #3269 (Remove all mentions of "Antlr4cs") (#3270)	2 months ago
star	[build] Fix of #3269 (Remove all mentions of "Antlr4cs") (#3270)	2 months ago
stellaris	[build] Fix of #3269 (Remove all mentions of "Antlr4cs") (#3270)	2 months ago
stl	[build] Fix of #3269 (Remove all mentions of "Antlr4cs") (#3270)	2 months ago

hellozrh [InformixSql] Add support of drop type/view (#3345)		sql
athena	[build] Fix for Issue 3232 (#3237)	
derby	[build] Fix for Issue 3232 (#3237)	
drill	[Drill] Add new apache drill grammar (#3202)	
hive	[build] Fix for Issue 3232 (#3237)	
informix-sql	[InformixSql] Add support of drop type/view (#3345)	
mariadb	[build] Fix for Issue 3232 (#3237)	
mysql	[MySQL] Support as uid in insert statement (#3345)	
phoenix	[build] Fix for Issue 3232 (#3237)	
plsql	[PLSql] Fix grammar bug with select time 'time str'	
postgresql	[PostgreSQL] Fix 'LEFT/RIGHT' joinType would be re	

Парсер-генераторы и парсеры

Парсер-генераторы –
библиотека для создания
парсеров на основе заданной
грамматики

Парсеры – программный код
для синтаксического анализа
конкретной грамматики

Например

Помогают создавать
парсеры

Парсер-генераторы

- *Lark*
- *Antlr*
- *Ply*
- *Pyarsing*
- *Parsiminious*
- *И т.д.*

Сами парсеры для
конкретных форматов

Парсеры

- *Sqlparse*
- *Sqlglot*
- *Xml*
- *Yaml*
- *Configparser*
- *И т.д.*

Парсер-генераторы в Python

<https://github.com/xnuinside/big-parsers-generators-comparison>

No.	Name	URL	Parsing Algorithms	Grammar Types
1	Lark	https://github.com/lark-parser/lark	Earley, LALR, CYK	EBNF, LALR
2	Parsley	https://parsley.readthedocs.io/en/latest/	PEG	PEG
3	PLY	http://www.dabeaz.com/ply/	Lex, Yacc	LALR(1)
4	PyParsing	https://github.com/pyparsing/pyparsing	Top-down	Top-down
5	Parsimonious	https://github.com/erikrose/parsimonious	PEG	PEG
6	ANTLR4	https://github.com/antlr/antlr4	LL(*)	LL(*)
7	Lrparsing	https://pypi.org/project/lrparsing/	LR(1)	LR(1)
8	SLY	https://sly.readthedocs.io/en/latest/sly.html#sly-overview	Lex, Yacc	LALR(1)
9	pyPEG	https://fdik.org/pyPEG/	PEG	PEG
10	parse	https://github.com/r1chardj0n3s/parse	-	-
11	pyleri	https://github.com/cesbit/pyleri	PEG	PEG



Парсер-генераторы в Python

- Всего 15 активных библиотек парсер-генераторов в Python + **ANTLR** + **еще**
- А еще есть чистый regex (например, **configparser** написан просто на регулярках)

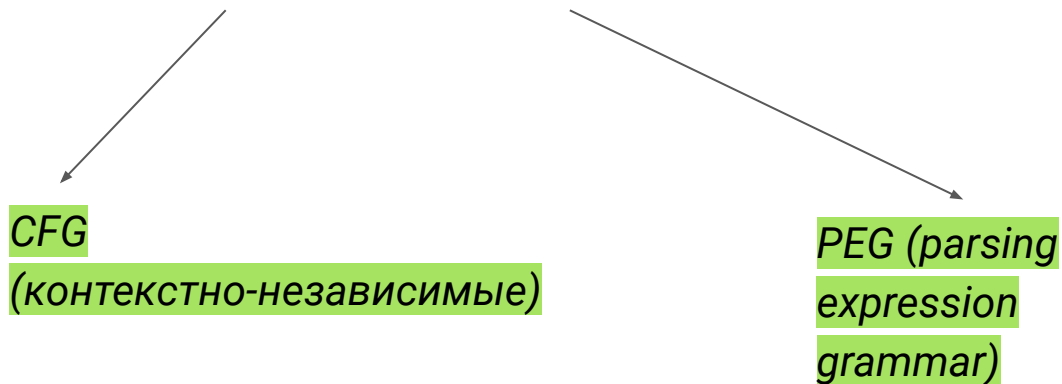
No.	Name	URL	Parsing Algorithms	Grammar Types
1	Lark	https://github.com/lark-parser/lark	Earley, LALR, CYK	EBNF,
2	Parsley	https://parsley.readthedocs.io/en/latest/	PEG	PEG
3	PLY	http://www.dabeaz.com/ply/	Lex, Yacc	LALR(1)
4	PyParsing	https://github.com/pyarsing/pyarsing	Top-down	Top-down
5	Parsimonious	https://github.com/erikrose/parsimonious	PEG	PEG

Напишем Парсер к нашей задаче

Название	url	Грамматика (язык для описания грамматики)	Алгоритмы
Lark	https://github.com/lark-parser/lark	EBNF (Extended Backus-Naur Form)	<ul style="list-style-type: none">- Earley (Эрли, top-down parsing algorithm)- LALR1 (Look-Ahead Left-Right)
Ply	https://github.com/dabeaz/ply	BNF (Backus-Naur Form)	<ul style="list-style-type: none">- LALR1 (Look-Ahead Left-Right)
Parsimonious	https://github.com/erikrose/parsimonious	PEG	<ul style="list-style-type: none">- PEG
Antlr	https://github.com/antlr	LL(k)	<ul style="list-style-type: none">- LL(*)

Очень коротко про грамматики

Основные 2 типа грамматик (на текущий день)



Основная мысль и идея стоящие за PEG

<https://bford.info/pub/lang/peg/>

Parsing Expression Grammars: A Recognition-Based Syntactic Foundation

- CFG изначально для естественных языков со всей их недетерминированностью
- в PEG реализован приоритизированный однозначный выбор
- PEG упрощает синтаксис, фактически устраняя разделение на лексические и иерархические компоненты

Формы описания
контекстно-свободной грамматики
(метаязык)

BNF

EBNF

LL(k)

Наглядно разница между BNF vs EBNF

BNF

```
<expr> ::= '-' <num> | <num>
<num> ::= <digits>
        | <digits> '.' <digits>
<digits> ::= <digit>
            | <digit> <digits>
<digit> ::= '0' | '1' | '2' | '3' |
            '4' | '5' | '6' | '7' | '8' | '9'
```

EBNF

```
<expr> := '-'? <digit>+ ('.' <digit>+)?
<digit> := '0' | '1' | '2' | '3' | '4' |
            '5' | '6' | '7' | '8' | '9'
```

Про PEG и EBNF

<https://docs.python.org/3/reference/grammar.html> - например, грамматика Python это микс между PEG и EBNF

The notation is a mixture of EBNF and PEG. In particular, `&` followed by a symbol, token or parenthesized group indicates a positive lookahead (i.e., is required to match but not consumed), while `!` indicates a negative lookahead (i.e., is required *not* to match). We use the `|` separator to mean PEG's "ordered choice" (written as `/` in traditional PEG grammars). See [PEP 617](#) for more details on the grammar's syntax.

Про PEG и EBNF

EBNF

```
value: STRING -> string
      | FLOAT -> float
      | CALENDAR_DATE -> date
      | INT -> int
      | DATE_RANGE -> date_range
      | expr -> expr
```

- Все нетерминалы равнозначны, есть проблема
- “Shift/Reduce-conflict” (у алгоритмов)

PEG

```
value = date / float / int / date_range
```

Тут очередность поиска

Есть дополнительные expressions, например:

“a - b”

Например:

a = b / “papa” / “babushka”

b = “mama”

Про алгоритмы

Нисходящие (top-down)

- LL(*) (Left-to-right, leftmost derivation / *Recursive descent parser*)
- Earley
- PEG (packrat)
- ...

Восходящие (bottom-up)

- LR parser (Left-to-right, Rightmost derivation in reverse)
- LALR
- CYK (Cocke–Younger–Kasami)
- ...

Знание алгоритма нужно для:

- Понимания как обрабатывается EBNF грамматика в конкретно выбранном вами парсер-генераторе
- Понимания времени/памяти парсера - если это вам важно
- В реальности достаточно понимания как писать грамматику для выбранного парсер-генератора

Напоминаем про задачу

```
> clients: 23.05
...
> clients: 2024-12-01
...
> orders: 2017
...
> cats: 2019-10-10

> clients: 2024-12-01 / 2025-12-01
...
> cats & orders: 2019-10-10
...
> cats - orders: 2024-12-01 / 2025-12-01
```

Как выглядит код для парсер-генератора (на примере Parsimonious)

PEG-грамматика

```
import parsimonious

grammar = """
line = key_expr colon value_expr
key_expr = key (op key)?
value_expr = value (op value)?
key = ~"[a-zA-Z_][a-zA-Z0-9_]*"
value = date / float / int / date_range
...
op = ws? ("&" / "-" / "/" ) ws?
ws = ~"\s*"
emptyline = ws+
"""

parser = parsimonious.Grammar(grammar)
```

Как выглядит код для парсер-генератора (на примере Parsimonious)

PEG-грамматика

```
class LineVisitor(parsimonious.NodeVisitor):

    def visit_line(self, node, visited_children):
        return (visited_children[0], visited_children[2])

    def visit_key(self, node, visited_children):
        return node.text

    def visit_date(self, node, visited_children):
        return node.text

    def visit_date_range(self, node, visited_children):
        return (visited_children[0], '/', visited_children[2])

    def visit_expr(self, node, visited_children):
        return (visited_children[1], visited_children[0], visited_children[2])

    ...
```

Как выглядит код для парсер-генератора (на примере *Parsimonious*)

PEG-грамматика

```
visitor = LineVisitor()

results = []
for line in lines.split('\n'):
    if not line.strip():
        continue
    parsed_data = parser.parse(line.strip())
    key = visitor.visit(parsed_data.children[0])
    value = visitor.visit(parsed_data.children[2])
    results.append((key, value))
```

Что пишем для парсер-генератора

1. Грамматика – правила языка
2. Visitor/Listener/Transformer –
обработчик (что на выходе хотим)

Далее уже вызываем парсер и
пользуемся результатами

- Генерирует парсеры, написанные на разных программирования, в том числе и Python
- Есть огромное количество готовых грамматик в открытом доступе
- Написан на Java
- Прост в использовании: пишем грамматику, вызываем:

```
pip install antlr4-tools (+ установите JVM)
```

```
antlr4 -Dlanguage=Python3 наша_грамматика.g4
```

- Свой синтаксис грамматики LL(k), есть свои нюансы и плюшки

- Например, фрагменты

```
fragment OperatorCharacter
: [*<>=~!@%^&|`?#]
;
// these are the operator characters that don't count towards one ending with + or -
```


- Или “Alternative Labels” - аналогично Labelled BNF

```
grammar T;
stat: 'return' e ';' # Return
    | 'break' ';' # Break
    ;
e : e '*' e # Mult
  | e '+' e # Add
  | INT # Int
  ;
```

Alternative labels do not have to be at the end of the line and there does not have to be a space after the context class definition for each label. For example, here is the listener that ANTLR generates:

```
public interface AListener extends ParseTreeListener {
    void enterReturn(AParser.ReturnContext ctx);
    void exitReturn(AParser.ReturnContext ctx);
    void enterBreak(AParser.BreakContext ctx);
    void exitBreak(AParser.BreakContext ctx);
    void enterMult(AParser.MultContext ctx);
    void exitMult(AParser.MultContext ctx);
    void enterAdd(AParser.AddContext ctx);
    void exitAdd(AParser.AddContext ctx);
    void enterInt(AParser.IntContext ctx);
    void exitInt(AParser.IntContext ctx);
}
```

- Из коробки создает AST
- 2 типа алгоритмов (но код придется переписывать под каждый тип отдельно)
- Можно отделять грамматику от кода – `.lark` расширение
- Есть свои “фишечки” с грамматикой
- Есть встроенные токены, которые можно импортировать и переиспользовать

- Аналогично Лейблингу в ANTLR

```
DATE_RANGE: CALENDAR_DATE " / " CALENDAR_DATE
expr: value "&" value -> and_expr
      | value "-" value -> minus_expr
key_expr: key "&" key -> and_expr
          | key "-" key -> minus_expr
You, 2 weeks ago • rename folders
```

- Наиболее удобный в использовании
(со слов самой либы, но на самом деле грамматика PEG действительно удобнее EBNF) + субъективно подтверждаю
- Следят за перфомансом (свечку над бенчмарком не держала, опять таки со слов либы)

- Один из старейших
- Наиболее доступный для изучения новичкам (очень много статей, документации, примеров)
- Порт Lex & Yacc на Python
- BNF-грамматика: Сложно поддерживать при больших грамматиках

*Давайте сравним грамматики и
почитаем их*

<https://github.com/xnuinside/simple-ddl-parser>

```
def p_type_name(self, p: List) -> None:
    """type_name : type_create id AS
    | type_create id DOT id AS
    | type_create id DOT id
    | type_create id
    """
    p_list = list(p)
    p[0] = {}
    if "." not in p_list:
        p[0]["schema"] = None
        p[0]["type_name"] = p_list[2]
    else:
        p[0]["schema"] = p[2]
        p[0]["type_name"] = p_list[4]
```

- Очень неприятные минусы BNF-грамматики
- Грамматика размазана по функциям-трансформерам

```
def p_type_create(self, p: List) -> None:
    """type_create : CREATE TYPE
    | CREATE OR REPLACE TYPE
    """
    p[0] = None
```

Суммаризируем

- Выбирайте парсер-генератор исходя из удобства написания и поддержания грамматики и трансформеров (в AST)
- В 90% задач, если вы не собираетесь писать свой компилятор - алгоритм влияет только на порядок прочтения грамматики в случае с EBNF
- PEG-грамматика читается всегда однозначно
- PEG не CFG, в Иерархии Хомского его нет (расцвет PEG после 2004 года)
- Для написания парсера вам достаточно понять правила написания грамматики для него для определения токенов и выражений
- Парсеры на практике это не сложно - не надо бояться
- Не обязательно использовать парсер на своем языке программирования

Что еще посмотреть/почитать

- Очень хороший курс статей на русском
<https://eugenezolotarev.ru/programming/creating-language/g-01-bnf>
- <https://coderlessons.com/tutorials/akademicheskii/izuchite-teoriiu-avtomatov/khomskaia-klassi-fikatsiia-grammatik> – про грамматики
- <https://tomasseti.me/parsing-in-python/> – классная большая статья на англ с примерами по парсер-генераторам
- <https://notebook.community/pbreach/pysd/docs/development/Building%20a%20SMILE%20to%20Python%20Translator%20using%20Parsimonious>
- <https://habr.com/ru/articles/348314/> – Просто о сложном. CFG, BNF, LL(k), LR(k), PEG и другие страшные слова
- Поиграть с грамматикой PEG онлайн: <https://ohmjs.org/>

Поставить отзыв доклада



Спасибо за внимание!

Вопросы?



Послесловие: *ChatGPT и парсеры*



python code with lark library to parse lines: ""clients: 23.05
clients: 2024-12-01
clients: 2024-12-01 / 2025-12-01
orders: 2017
cats: 2019-10-10""



Here's an example Python code using the Lark library to parse the given lines:



python

Copy code

```
from lark import Lark, Transformer

# Define the grammar for the key-value pairs
grammar = """
    start: pair+
    pair: WORD ":" value
    value: FLOAT | DATE | DATE_RANGE | INT
    DATE: /\d{4}-\d{2}-\d{2}/
    DATE_RANGE: DATE "/" DATE
    %import common.WORD
    %import common.FLOAT
```

Токены и грамматики

ТОКЕН



`${key}: ${value}`



Выражение
(expression)

Для сравнения

На примере Parsley

ChatGPT


Рабочий код

Доделать слайд

Коммит, где весь код от ChatGPT -

<https://github.com/xnuinside/big-parsers-generators-comparison/tree/f7a8c4adb4f1ca95ab5ef5c3612b2a6b31f21065>

Спасибо за внимание!



Обязательно
поделитесь
своим отзывом
и мнением о
докладе

Что еще посмотреть/почитать

Hypothesis (тестирование парсеров)

<https://ohmjs.org/editor/>

<https://github.com/anentropic/hypothesis-grammar>