Документация к генератору кс-грамматики

Константинова Полина, Павлов Игорь

31 декабря 2022 г.

Содержание

1 Описание грамматики

1.1 Грамматика

Программа генерирует КС-грамматики, описывающиеся грамматикой, приведённой ниже:

```
<grammar> ::= <rule> <rule-sep> <grammar-tail>
<grammar-tail> ::= <rule> <rule-sep> <grammar-tail>
<grammar-tail> ::=

<rule> ::= <nonterm> <arrow> <rule-right>

<rule-right> ::= <production> <rule-tail>
<rule-tail> ::= <production-sep> <production> <rule-tail>
<rule-tail> ::=

<production> ::= <epsilon>
<production> ::= <term> <production-tail>
<production> ::= <nonterm> <production-tail>

<production-tail> ::= <term> <production-tail>
<production-tail> ::= <nonterm> <production-tail>
<production-tail> ::=

<term> ::= <term-regex>
<nonterm> ::= <nterm-start> <nterm-regex> <nterm-end>
```

Токены, задаваемые пользователем:

```
<rule-sep>
<arrow>
<production-sep>
<nterm-start>
<nterm-end>
<term-regex>
<nterm-regex>
```

1.2 Свойства грамматики

При разумном выборе пользователем токенов грамматика детерминированная и `LL(1)`. Грамматика `LR(1)`, так как префикс-свойство не выполняется.

2 Генератор

2.1 Возможности

Данная программа позволяет генерировать КС-грамматики с пользовательским синтаксисом. Есть возможность генерировать только достижимые нетерминалы, только генерирующие нетерминалы и нетерминалы, не порождающие пустые слова.

Также есть возможность проверки грамматики на `LL(1)`. Если пользовательский синтаксис будет нарушать `LL(1)`, выведется ошибка, которая укажет, какие токены надо заменить.

Грамматика генерируется не в виде строки, а в виде структуры. Поэтому есть возможность применять алгоритмы анализа КС-грамматик.

2.2 Установка и запуск

Для запуска генератора вам понадобится язык программирования python3 и установщик пакетов pip. Далее склонируйте репозиторий и установите зависимости:

```
git clone https://github.com/pollykon/FLT_lab5
cd FLT_lab5
pip install -r requirements.txt
```

Для работы генератора необходимо создать конфигурационный файл и указать в нем параметры генерации. Для запуска генератора используйте команду:

```
python generator.py < path/to/config.yaml
```

Программа выведет сгенерированную грамматику, а также два флага: являются ли все правила достижимыми и все нетерминалы порождающими.

Чтобы включить проверку получаемой грамматики на LL(1) свойство, используйте ключ `--check-ll1`:

```
python generator.py --check-ll1 < path/to/config.yaml
```

2.3 Структура проекта

```
FLT_lab5/
- .gitignore
- additional/
    - attribute_grammar.pdf
    - lab_5_dop.pdf
- cfg.py
- configs/
    - default.yaml
    - test.yaml
- generator.py
- README.md
- requirements.txt
- utils.py
- validator.py
```

В репозитории в директории `configs` лежат конфигурационные файлы в формате `yaml`. В `validator.py` находится код валидации параметров конфига и проверки получаемой грамматики на LL(1) свойство. В модуле `cfg.py` объявлен класс, описывающий КС-грамматику. Основной модуль - `generator.py`, в нем находится код генератора. Генератор генерирует грамматику в виде экземпляра класса `CFG`.

2.4 Настраиваемые параметры

В репозитории в директории `configs` лежат файлы с пользовательским синтаксисом. Параметры указываются в файле формата `yaml`. Если какие-то параметры пользователь не указал, они загружаются из файла `default.yaml`.

`grammar`:

`nonterminals_range` - (list) диапазон количества нетерминалов. Значение по умолчанию: `[1, 6]`;

`rules_for_nonterminal_range` - (list) диапазон количества правил для нетерминала. Значение по умолчанию: `[1, 3]`;

`only_non_empty_nonterminals` - (boolean) флаг для генерации нетерминалов, не порождающих пустое слово (при значении `True`). Значение по умолчанию: `False`;

`unreachable_nonterminals` - (boolean) флаг для генерации недостижимых нетерминалов (при значении `True`). Значение по умолчанию: `False`;

`only_generating_nonterminals` - (boolean) флаг для генерации только порождающих нетерминалов (при значении `True`). Значение по умолчанию: `False`;

`start_nonterminal` - (string) стартовый нетерминал. Значение по умолчанию: `S`;

`production_separator` - (string) разделитель для продукций. Значение по умолчанию: |;

`arrow` - (string) разделитель для нетерминала и его правила. Значение по умолчанию: `->`;

`rule_separator` - (string) разделитель для правил. Значение по умолчанию `;`;

`epsilon`:

`value` - (string) символ, обозначающий пустое слово. Значение по умолчанию: `eps`;

`chance` - (float) вероятность добавления пустого слова к правилу. Значение по умолчанию: `0.35`;

`production`:

`max_symbols` - (int) максимальное количество элементов продукции. Значение по умолчанию: `5`; `terminals`:

`regex` - (string) регулярное выражение для задания значений терминалов. Значение по умолчанию: `[a-z0-9]`;

`max_length` - (int) максимальная длина терминала. Значение по умолчанию: `4`;

`nonterminals`:

`nonterminal_start` - (string) разделитель для отличия нетерминала от терминала (в начале нетерминала). Значение по умолчанию: `[`;

`nonterminal_end` - (string) разделитель для отличия нетерминала от терминала (в конце нетерминала). Значение по умолчанию: `]`;

`regex` - (string) регулярное выражение для задания значений нетерминалов. Значение по умолчанию: `[A-Z]`;

`max_length` - (int) максимальная длина нетерминала. Значение по умолчанию: `4`;

3 Примеры работы

3.1 Стандартное использование

Запустим генератор, используя конфигурационный файл по-умолчанию:

```
$ python generator.py < configs/default.yaml
[H] -> [P]6g[H] | [J][O][H];
[O] -> [O]k4[H][H] | [S]n;
[P] -> [S][S]5[J] | [J] | epsilon;
[S] -> [S] | [P];
[J] -> [O][O][S][J];

is generating: False
all reachable: True
```

Попробуем изменить параметр `production_separator` в конфигурационном файле на `;` и включить проверку на LL(1):

```
$ python generator.py --check-ll1 < configs/default.yaml
Traceback (most recent call last):
  File ...
  ...
validator.NotLL1Grammar: For LL1 Grammar rule_separator
should be different from production_separator
```

3.2 Продвинутое использование

Объявим класс генератора `Generator` и попробуем сгенерировать грамматику. `Generator` инициализируется конфигом, который можно считать, используя библиотеку `pyyaml`.

```
from generator import Generator
from cfg import CFG

gen = Generator(config)
cfg = gen.generate_grammar()
```

`cfg` - экземпляр класса `CFG`. Этот класс можно расширить, добавив методы для анализа и обработки КС-грамматик. Работа с классом `CFG` абстрагирована от пользовательского синтаксиса, при этом `print(cfg)` выведет грамматику в пользовательском синтаксисе.