

1 Вариант 3. Логические формулы в стиле Python

Логические формулы. Используются операции `and`, `or`, `xor`, `not`. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

В качестве операндов выступают переменные с именем из одной буквы. Используйте один терминал для всех переменных. Для каждой логической операции должен быть заведен один терминал (не три 'a', 'n', 'd' для `and`).

Пример: `(a and b) or not (c xor (a or not b))`

2 Разработка грамматики

Грамматика находится в файле `resources/grammar.yaml`. Данный файл должен иметь следующие обязательные поля:

- *terminals* - массив всевозможных терминалов в формате `[name]{{name: [name], option: [option]}}`
Где *name* - имя уникальное имя терминала. N.B. запрещено в имени использовать символы `['$]` и в качестве имени использовать `eps`
Где *option* - некоторые опции в формате `[WHITESPACELESS|VARIABLE]`
WHITESPACELESS - возможное отсутствие пробельных символов вокруг данного нетерминала
VARIABLE - текущий нетерминал является переменной (принимает значения `[a-z]`)
- *non-terminals* - массив всевозможных нетерминалов в формате `[name]`
Где *name* - имя уникальное имя нетерминала. N.B. запрещено в имени использовать символы `['$]`
- *start-non-terminal* - необязательный параметр. Присваивается имя стартового для разбора нетерминала.
By default: `S`
- *rules* - массив правил грамматики в формате `[non-terminal] -> [terminal|non-terminal|eps] +`
Где *non-terminal* - имя нетерминала
Где *terminal* - имя терминала
Где *eps* - терминал пустой строки

Грамматика для данного задания:

```
terminals:
  - xor
  - or
  - and
  - not
  - {name: (, option: WHITESPACELESS}
  - {name: ), option: WHITESPACELESS}
  - {name: var, option: VARIABLE}
non-terminals:
  - S
  - XOR
  - OR
  - AND
  - NOT
  - BRACKETS
start-non-terminal: S
rules:
```

```

- S -> XOR
- XOR -> OR xor XOR
- XOR -> OR
- OR -> AND or OR
- OR -> AND
- AND -> NOT and AND
- AND -> NOT
- NOT -> not NOT
- NOT -> BRACKETS
- BRACKETS -> ( S )
- BRACKETS -> var

```

Смысл текущих нетерминалов:

- S - стартовый нетерминал, который раскрывается в анализируемое выражение
- XOR - нетерминал, раскрывающийся в выражение вида $\alpha \text{ xor } \beta$ или *OR*
- OR - нетерминал, раскрывающийся в выражение вида $\alpha \text{ or } \beta$ или *AND*
- AND - нетерминал, раскрывающийся в выражение вида $\alpha \text{ and } \beta$ или *NOT*
- NOT - нетерминал, раскрывающийся в выражение вида *not* α или *BRACKETS*
- BRACKETS - нетерминал, раскрывающийся в выражение вида (α) или $[a-z]$

Удаление правого ветвления и непосредственной левой рекурсии происходит в классе *grammar/Grammar.java*. В следствие получаются следующие правила:

```

S -> XOR
XOR'r -> xor XOR
XOR'r -> eps
OR'r -> or OR
OR'r -> eps
AND'r -> and AND
AND'r -> eps
NOT -> not NOT
NOT -> BRACKETS
BRACKETS -> ( S )
BRACKETS -> var
XOR -> OR XOR'r
OR -> AND OR'r
AND -> NOT AND'r

```

Смысл новых нетерминалов:

- S - стартовый нетерминал, который раскрывается в анализируемое выражение
- XOR'r - нетерминал, наращивающий выражение вида $(\text{xor } \alpha)^*$
- OR'r - нетерминал, наращивающий выражение вида $(\text{or})^*$
- AND'r - нетерминал, наращивающий выражение вида $(\text{and } \alpha)^*$
- NOT - нетерминал, раскрывающийся в выражение вида *not* α или *BRACKETS*
- BRACKETS - нетерминал, раскрывающийся в выражение вида (α) или $[a-z]$
- XOR - нетерминал, раскрывающийся в выражение вида $(\alpha \text{ xor } \beta)^+$ или *OR*
- OR - нетерминал, раскрывающийся в выражение вида $(\alpha \text{ or } \beta)^+$ или *AND*
- AND - нетерминал, раскрывающийся в выражение вида $(\alpha \text{ and } \beta)^+$ или *NOT*

3 Лексический анализатор

Лексический анализатор находится в классе *lexic/LexicalAnalyzer.java*. Данный анализатор по переданной строке позволяет вызывать метод `nextToken()`, который пытается взять следующий токен в данной строке типа *grammar/objects/terminals/Terminal.java* из списка, предоставленном в пункте 2 и специального терминала *EOS*, обозначающий конец строки, и метод `getToken()` который возвращает текущий распаршенный токен.

4 Синтаксический анализатор

В классе *syntax/SyntaxAnalyzer.java* находятся методы предподсчета и получения функций *FIRST* и *FOLLOW*.

FIRST от нетерминалов равны:

```
S [not, (, var]
XOR [not, (, var]
OR [not, (, var]
AND [not, (, var]
NOT [not, (, var]
BRACKETS [(, var]
XOR'r [xor, eps]
OR'r [or, eps]
AND'r [and, eps]
```

FOLLOW от нетерминалов равны:

```
S [$, )]
XOR [$, )]
OR [xor, $, )]
AND [or, xor, $, )]
NOT [and, or, xor, $, )]
BRACKETS [and, or, xor, $, )]
XOR'r [$, )]
OR'r [xor, $, )]
AND'r [or, xor, $, )]
```

Сам же парсер выражения находится в классе *syntax/ExpressionParser.java*.

Данный парсер подбирает для текущего токена и нетерминала правило $A \rightarrow \alpha$, для которого наш токен присутствует в множестве $(FIRST(\alpha) \setminus eps) \cup (FOLLOW(A) \text{ if } eps \in FIRST(\alpha))$.

Затем мы перебираем все оъекты (терминалы и нетерминалы) из α . Если текений объект терминал, запускаем рекурсивно данную функцию парсинга. Если - нетерминал, проверяем, что текущий токен совпадает с ожидаемым и переходим на следующий.

В итоге парсер возвращает дерево разбора в виде объекта класса *syntax/ExpressionParser.Node*.

5 Визуализация дерева разбора

Визуализация дерева разбора происходит при помощи библиотеки GraphViz в классе *visualizer/GraphVisualizer*. Визуализированное дерево разбора можно найти по пути *graph/parsedTreeGraph.png*

6 Тесты

Тесты находятся по пути *test/java/**

Тесты представляют из себя:

- SimpleUnitTests - простые unit тесты, предназначенные, чтобы протестировать базовые возможности парсера

- LexicalAnalyzerTests - тесты на корректность лексического анализатора
 - *lexicalAnalyzerTest₁* проверяет, что строки, не являющиеся терминалами отклоняются
 - *lexicalAnalyzerTest₂* проверяет, что сконкатинированные операторы и операнды отклоняются
- SyntaxAnalyzerTests - тесты на корректность синтаксического анализатора
 - *lexicalAnalyzerTest₁* проверяет, что пустое выражение отклоняются
 - *lexicalAnalyzerTest₂₋₈* проверяет, корректные терминалы, поставленные в неправильном порядке (не удовлетворяет изначальным правилам), отклоняются
- RandomTests - тесты на корректность с рандомно сгенерированными выражениями