Methods Of Translation - Lab2

Елфимов Никита Юрьевич M33371

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* - имяуникальное имя терминала. N.B. запрещено в имени использовать символы [’$] и в качестве имени использовать eps

  Где *option* - некоторые опции в формале [WHITESPACELESS|VARIABLE]

  WHITESPACELESS - возможное отсутствие пробельных символов вокруг данного нетерминала

  VARIABLE - текущий нетерминал является переменной (примимает значения [a-z])

- *non-terminals* - массив всевозможных нетерминалов в формате [name]

  Где *name* - имяуникальное имя нетерминала. N.B. запрещено в имени использовать символы [’$]

- *start-non-terminal* - необязательный параметр. Присваевается имя стартового для разбора нетерминала. By default: S

- *rules* - массив правил граматики в формате [non-terminal] -> [terminal|non-terminal|eps]+

  Где *non-terminal* - имя нетерминала

  Где *terminal* - имя терминала

  Где *eps* - терминал пустой строки

Граматика 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 нтерминал, который расскрывается в анализируемое выражение
- XOR - нетерминал, расскрывающийся в выражение вида α *xor* β или *OR*
- OR - нетерминал, расскрывающийся в выражение вида α *or* β или *AND*
- AND - нетерминал, расскрывающийся в выражение вида α *and* β или *NOT*
- NOT - нетерминал, расскрывающийся в выражение вида *not* α или *BRACKETS*
- BRACKETS - нетерминал, расскрывающийся в выражение вида *(α)* или *[a-z]*

Удаление правого ветвления и непосредственной левой рекурсии происходит в клссе *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 нтерминал, который расскрывается в анализируемое выражение
- XOR'r - нетерминал, наращивающий выражение вида *(xor α)**
- OR'r - нетерминал, наращивающий выражение вида *(or)**
- AND'r - нетерминал, наращивающий выражение вида *(and α)**
- NOT - нетерминал, расскрывающийся в выражение вида *not* α или *BRACKETS*
- BRACKETS - нетерминал, расскрывающийся в выражение вида *(α)* или *[a-z]*
- XOR - нетерминал, расскрывающийся в выражение вида *(α xor β)+* или *OR*
- OR - нетерминал, расскрывающийся в выражение вида *(α or β)+* или *AND*
- AND - нетерминал, расскрывающийся в выражение вида *(α and β)+* 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 распаршеный токен.

4 Синтаксический анализатор

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

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 $A \to \alpha$, для которого наш токен присутствует в множестве *(FIRST(*α*) \ eps)* $\cup$*(*$FOLLOW(A)$ *if eps* $\in FIRST(\alpha)$*)*.
Затем мы перебираем все оъекты (терминалы и нетерминалы) из α. Если текещий 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 библеотеки GraphViz в классе *visualizer/GraphVisualizer*.
Визуализированное дерево разбора можно найи 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
  - $lexicalAnalyzerTest_1$ проверяет, что строки, не являющиеся терминалами отклонияются
  - $lexicalAnalyzerTest_2$ проверяет, что сконкатинированные операторы и операнды отклоняюся
- SyntaxAnalyzerTests - тесты на корректность синтаксическоо анализатора
  - $lexicalAnalyzerTest_1$ проверяет, что пустое выражение отклонияются
  - $lexicalAnalyzerTest_{2-8}$ проверяет, коректные терминалы, поставленные в неправельном порядке (не удовлетворяет изначальным правилам), отклоняются
- RandomTests - тесты на корректность с рандомно сгенерированными выражениями