Лабораторная 2: ручное построение нисходящих парсеров

Вариант: 8 (Описание лямбда функций в Python)

Этот документ автоматически форматируется в Okular, но на всякий случай я экспортировал его заранее в README.pdf

Задание 1: Грамматика

Для наглядности, терминалы пишутся не так как называются токены (i.e. ")" вместо RPAREN, и "lambda" вместо `LAMBDA_KW`, `var` вместо `VARIABLE`)

Исходная грамматика

```
Declaration -> "lambda" Arglist ":" Expression

Arglist -> eps
Arglist -> Varlist
Varlist -> Varlist "," var
Varlist -> var

El -> El "|" Tl
El -> Tl
Tl -> Tl "&" Expression
Tl -> Expression
Expression -> Expression "+" Term
Expression -> Term
Term -> Term "*" Factor
Term -> Factor
Factor -> (El)
Factor -> var
Factor -> const
```

Описание нетерминалов

```
Нетерминал      Описание
Declaration     Описание лямбда функции в Python
Arglist         Список аргументов функции (может быть пустым)
Varlist         Непустой список comma-separated переменных
El              Левая часть побитовой дизъюнкции
Tl              Левая часть побитовой конъюнкции
Expression      Левое слагаемое в арифметическом выражении
Term            Левый множитель в арифметическом выражении
Factor          Множитель в арифметическом выражении
```

Модифицированная грамматика

В исходной грамматике есть левая рекурсия, избавимся от неё:

```
Declaration -> "lambda" Arglist ":" Expression

Arglist -> Varlist
Arglist -> ""
~ Varlist -> var Varlist'
~ Varlist' -> "," var Varlist'
+ Varlist' -> eps

~ El -> Tl El'
~ El' -> "|" Tl El'
+ El' -> eps
~ Tl -> Expression Tl'
~ Tl' -> "&" Expression Tl'
+ Tl' -> eps
~ Expression -> Term Expression'
~ Expression' -> "+" Term Expression'
+ Expression' -> eps
~ Term -> Factor Term'
~ Term' -> "*" Factor Term'
```

```
+ Term' -> eps
Factor -> (El)
Factor -> var
Factor -> const
```

(~ - измененная строка, + - новая строка, по аналогии с `diff`)

Описание нетерминалов

Запись `-//-` значит “ditto”:

```
Нетерминал      Описание
Declaration     -//-
Arglist         -//-
Varlist         -//-
Varlist'        Продолжение списка из переменных
El              -//-
El'             То, что находится справа от левого операнда дизъюнкции
Tl              -//-
Tl'             То, что находится справа от левого операнда конъюнкции
Expression      -//-
Expression'     То, что находится справа от левого операнда суммы
Term            -//-
Term'           То, что находится справа от левого операнда умножения
Factor          -//-
```

Задание 2: Лексический анализатор

```
Терминал    Токен
"("         LPAREN
")"         RPAREN
$           END
"|"         VBAR
"&"         AMPERSAND
"+"         PLUS
"*"         ASTERISK
var         VARIABLE
const       CONSTANT
"lambda"    LAMBDA_KW
":"         COLON
","         COMMA
```

Задание 3: Синтаксический анализатор

Таблица FIRST и FOLLOW исходной грамматики

```
Нетерминал      FIRST             FOLLOW
Declaration     "lambda"          $
Arglist         var eps           ":"
Varlist         var               ":" ","
El              "(" var const     $ "|" ")"
Tl              "(" var const     $ "&" "|" ")"
Expression      "(" var const     $ "+" "&" "|" ")"
Term            "(" var const     $ "*" "+" "&" "|" ")"
Factor          "(" var const     $ "*" "+" "&" "|" ")"
```

Таблица FIRST и FOLLOW модифицированной грамматики

```
Нетерминал      FIRST            FOLLOW
Declaration     "lambda"         $
Arglist         var eps          ":"
Varlist         var              ":"
Varlist'        "," eps          ":"
El              "(" var const    $ ")"
El'             "|" eps          $ ")"
```

```
Tl              "(" var const    $ "|" ")"
Tl'             "&" eps          $ "|" ")"
Expression      "(" var const    $ "&" "|" ")"
Expression'     "+" eps          $ "&" "|" ")"
Term            "(" var const    $ "+" "&" "|" ")"
Term'           "*" eps          $ "+" "&" "|" ")"
Factor          "(" var const    $ "*" "+" "&" "|" ")"
```

Как собрать бинарник парсера (принимает на вход выражение и выдаёт дерево разбора для graphviz):

```
make parser
```

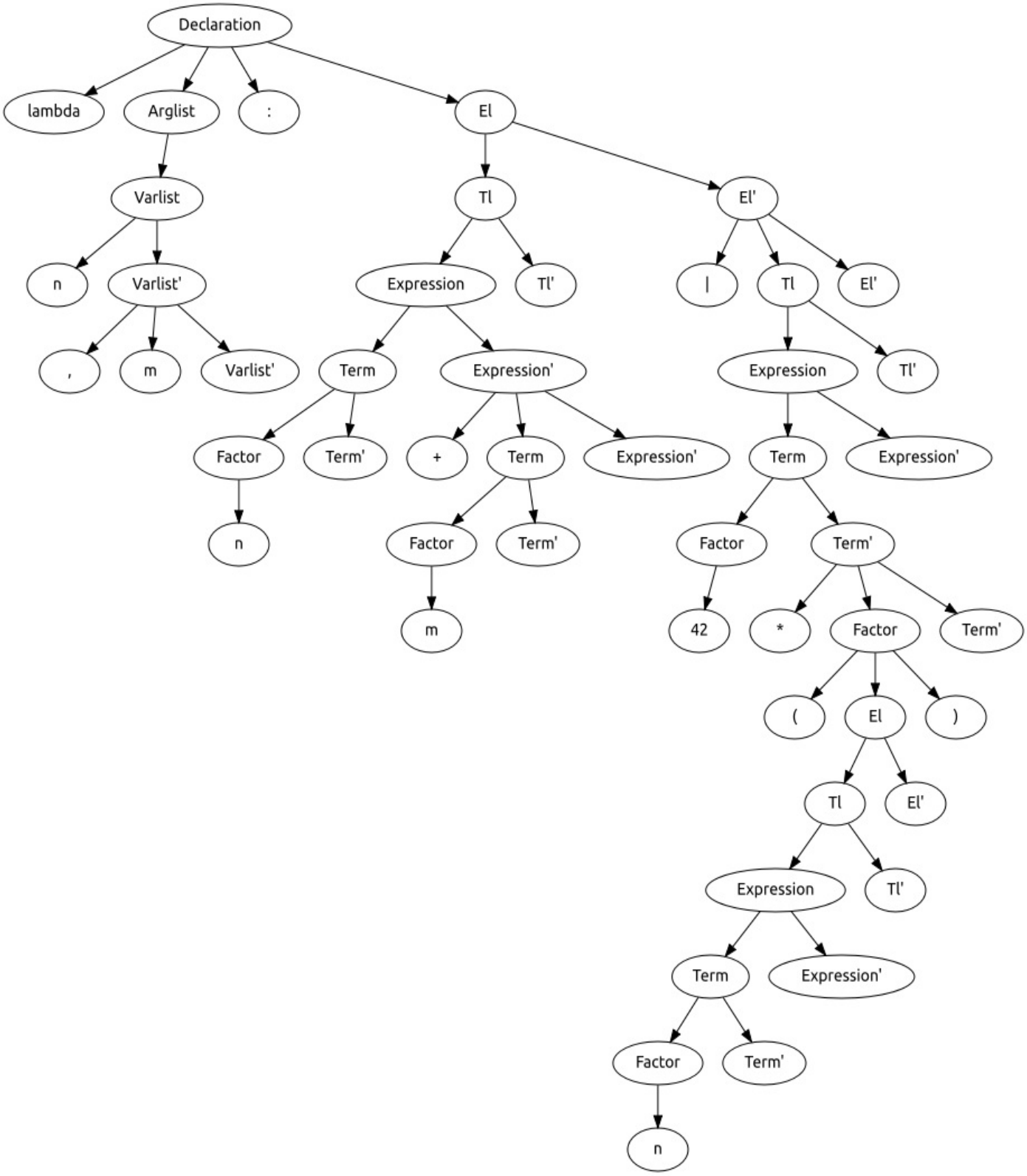
Задание 4: Визуализация дерева разбора

Пример использования:

```
$ ./parser >pic.dot && dot -T svg pic.dot >pic.svg && open pic.svg
lambda n, m: n + m | 42 * (n)
<ctrl+d to indicate end of input>
```

Либо

```
$ make picture
lambda n, m: n + m | 42 * (n)
<ctrl+d to indicate end of input>
```



Задание 5: Тесты

Автоматические тесты

`make test` - скажет если что-то не так. Гоняет программу на наборе валидных (`./examples`) и невалидных (`./invalid-examples`) строк.

Валидные строки

Программа должна завершиться без ошибки и вывести AST для graphviz:

Вход	Вывод
`lambda: n`	OK
`lambda  :    n`	OK

```
lambda x: x*x                          OK
lambda x   : x  *x                     OK
lambda a, b:   a + b * (c)             OK
lambda x,y,z: (((x*x + y*y + z*z)))    OK
lambda x,lambdaa: x + lambdaa          OK
lambda n, m: n + m | 42 * (n)          OK
```

(Полный список тестов лежит в `./examples`)

Невалидные строки

Программа должна завершиться с ошибкой и вывести соответствующее сообщение:

```
Вход                                 Вывод программы (stderr)
lambda                               Expected argument list, got `` at position 7
lambda:                              Expected beginning of expression, got `` at
                                     position 8
lambda n : :                         Expected beginning of expression, got `:` at
                                     position 12
: n                                  Expected lambda declaration, got `:` at
                                     position 1
;;; invalid token chars              There are no tokens that start with `;` (at
                                     pos 1)
a bunch of tokens                    Expected lambda declaration, got `a` at position 1
lambda varlist without commas: x     Expected continuation of variable list, got
                                     `without` at position 16

lambda x, y: (x & 102&) | (y & 32)   Expected Expression, got `)` at position 23
```

(полный список тестов лежит в `./invalid-examples`)