

# 1 Знакомство с языком Лисп. Абстракция вычислений

**Введение.** Абстракция — умышленное игнорирование частных свойств и связей и выделение основных. В языках программирования различают абстракцию вычислений и абстракцию данных. Циклы, процедуры, продолжения (continuations), операторы недетерминизма, исключения, потоки (threads) — вот доступные во многих языках программирования механизмы абстракции вычислений. Например, цикл `for` в Питоне позволяет абстрагироваться от деталей реализации прохода по контейнеру: не нужна переменная-индекс для прохода по массиву или итератор для прохода по списку или дереву. Процедуры позволяют описывать часть вычислений в виде блока в обособленном месте программы: в другом файле или модуле с тем, чтобы ссылаться на него многократно из других мест программы, использовать как параметр в другой процедуре, сохранять в структурах данных, а в некоторых языках даже модифицировать. Продолжения являются мощным механизмом конструирования более высокоуровневых абстракций вычислений: в Лиспе продолжения могут использоваться для реализации потоков, исключений, а также оператора недетерминизма `amb`. Оператор `amb` позволяет абстрагироваться от порядка перебора путей в графе состояний при поиске в глубину.

Основным механизмом абстракции вычислений в Лиспе и в некоторых других языках являются процедуры. Т.к. большинство диалектов Лиспа требуют реализации оптимизации хвостового вызова (в частности хвостовой рекурсии), многие другие механизмы абстракции вычислений, например циклы, также реализуются через процедуры. Кстати, в некоторых языках, например в Питоне, можно реализовать оптимизацию хвостовой рекурсии на уровне библиотеки встроенными средствами языка. В качестве необязательного задания рекомендуется в этом разобраться.

Целью данной лабораторной работы является знакомство с основным механизмом абстракции вычислений — процедурами — в чисто-функциональных и смешанных языках программирования.

При выполнении лабораторной работы рекомендуется использовать языки программирования, в которых процедуры являются объектами первого класса, т.е. где допустимо сохранение процедур в структурах данных, передача их в качестве параметров, конструирование во время выполнения и передача в качестве возвращаемого значения. Таким образом допускается использование языков Лисп, Питон, Руби, JavaScript и др. В связи с невозможностью конструирования функций во время выполнения, недопустимо использование языков C, C++, Java и подобных.

**Задание.** Реализовать процедуру нахождения неподвижной точки преобразования методом итераций в общем виде, т.е. параметризованную оператором преобразова-

ния, процедуру, вычисляющую преобразование Ньютона, и применить их для реализации метода Ньютона. Нахождение производной также реализовать в виде процедуры (принимаящей функцию и возвращающей функцию — ее производную).

Решить методом Ньютона следующие уравнения:

1.  $e^x + \ln x - 10x = 0, x_0 = 3.$
2.  $3x - 14 + e^x - e^{-x} = 0, x_0 = 1.$
3.  $4 \ln^2 x + 6 \ln x - 5 = 0, x_0 = 1.$
4.  $2x \sin x - \cos x = 0, x_0 = 0.$
5.  $x \operatorname{tg} x - \frac{1}{3} = 0, x_0 = 1.$
6.  $0,25x^3 + x - 1,2502 = 0, x_0 = 0.$
7.  $0,1x^2 - x \ln x = 0, x_0 = 1.$
8.  $3x - 4 \ln x - 5 = 0, x_0 = 2.$
9.  $e^x - e^{-x} - 2 = 0, x_0 = 1.$

## 2 Отложенные вычисления и ленивые списки в Лиспе. Нисходящий разбор по заданной LL(1)-грамматике

**Введение.** Нисходящий синтаксический анализ осуществляется автоматом с магазинной памятью (МП-автоматом). В начале работы МП-автомата в стек помещается стартовый нетерминал. Далее автомат недетерменировано заменяет нетерминалы в стеке на правые части соответствующих продукций, а терминалы снимает с вершины стека вместе с чтением их из входной строки. Строка допускается, если по окончании чтения строки стек пуст. Недетерминизм на практике реализуется сложно, поэтому там, где это возможно, грамматики приводятся к специальной форме, позволяющей от него избавиться. Одна из таких форм — LL(1). Первая «L» означает чтение входной строки слева направо, вторая «L» — получение левого порождения, а «1» — использование на каждом шаге предпросмотра одного символа для принятия решения о действиях автомата.

Грамматика  $G$  называется LL(1)-грамматикой, если для любых двух различных продукций  $A \rightarrow \alpha \mid \beta$  выполняются следующие условия:

1.  $\operatorname{first} \alpha \cap \operatorname{first} \beta = \emptyset.$

2. Если  $\beta$  порождает пустую строку, то  $\text{first } \alpha \cap \text{follow } A = \emptyset$ . Аналогично, если  $\alpha$  порождает пустую строку, то  $\text{first } \beta \cap \text{follow } A = \emptyset$ .

$\text{first } \alpha$  — это множество, содержащее все терминалы, с которых начинаются строки, выводимые из  $\alpha$  и  $\varepsilon$ , если  $\alpha$  порождает пустую строку.

$\text{follow } A$  — множество терминалов, которые могут располагаться непосредственно справа от  $A$  в некоторой сентенциальной форме.

**Задание.** Написать программу, эмулирующую поведение детерминированного МП-автомата, составленного по заданной LL(1)-грамматике. Проверку принадлежности строки языку выполнить в виде поиска допускающего состояния в истории вычислений автомата. Историю вычислений реализовать в виде отложенного списка, то есть не допускается одновременное хранение в памяти всех промежуточных состояний автомата.

Символы, не встречающиеся в левых частях грамматик, и считать терминалами.

1.  $\text{expr} \rightarrow \text{term expr2}$   
 $\text{expr2} \rightarrow + \text{term expr2} \mid \varepsilon$   
 $\text{term} \rightarrow \text{fact term2}$   
 $\text{term2} \rightarrow * \text{fact term2} \mid \varepsilon$   
 $\text{fact} \rightarrow (\text{expr}) \mid a$
2.  $\text{stmt} \rightarrow \text{if } (\text{expr}) \text{ stmt else stmt} \mid \text{while } (\text{expr}) \text{ stmt} \mid \{\text{stmtlist}\}$   
 $\text{stmtlist} \rightarrow \text{stmt stmtlist} \mid \varepsilon$
3.  $\text{stmt} \rightarrow \text{if expr then stmt else stmt}$   
 $\text{expr} \rightarrow \text{term expr2}$   
 $\text{expr2} \rightarrow \text{or term expr2} \mid \text{and term expr2} \mid \varepsilon$   
 $\text{term} \rightarrow a$
4.  $\text{proc} \rightarrow \text{head proclist body}$   
 $\text{head} \rightarrow \text{procedure name } (\text{params})$   
 $\text{proclist} \rightarrow \text{proc proclist2}$   
 $\text{proclist2} \rightarrow ; \text{proc proclist2} \mid \varepsilon$   
 $\text{params} \rightarrow \text{param params2}$   
 $\text{params2} \rightarrow , \text{param params2} \mid \varepsilon$