КМ 1 Лабораторный практикум 5

Компьютерная математика
БГУ, ММФ, 1 курс, КМ
доц. Малевич А.Э.,
доц. Щеглова Н.Л.
2021-10-06

Рекурсия

Первым делом – сохраняем Рабочий Документ!!!

Создайте новый документ, запишите название работы, атрибуты исполнителя, дату. Секционируйте документ с учетом количества разделов в выполняемом задании.

Сохраните документ в памяти компьютера, на котором Вы работаете, в папке Вашего курса, Вашей группы. Имя документа должно содержать идентификатор работы и Вашу фамилию (КМ1ЛБ05xxxxxxx.nb).

1. Построение дерева выражения

`Задание 2` Напишите функцию, которая отображает заданное выражение, его полную форму, в виде дерева. Используйте рекурсию.

`Выполнение задания 2`

Постоим функцию `ExprAsTree[expr]`, которая вычисляет графический образ выражения `expr` в виде дерева.

`Задание 2.1` Сформулируйте основные соглашения при отображении выражения. Опишите процесс построения дерева выражения, используя рекурсию.

`Выполнение задания 2.1`

Определим соглашения, которыми будем руководствоваться при отображении заданного выражения `expr`.

Согласно рекурсивному определению, выражение – это либо атомарное выражение, либо неатомарное выражение.

Понятие атомарного выражения, или атомарного объекта является базой рекурсии. Определение атомарного выражения см. [1, с. 55].

Выражение, не являющееся атомарным, состоит из головы и кортежа аргументов. И голова, и каждый аргумент – это, в свою очередь, выражение, атомарное либо нет. Так мы описываем один шаг рекурсивного определения выражения.

Договоримся, что мы отображаем выражение `expr` в виде планарного связного ациклического графа – дерева, с указанным корневым узлом.

Если отображаемое выражение – атомарный объект, то в корневом узле располагаем атом.

Если заданное выражение не является атомарным, то оно имеет вид $\texttt{head[arg}_1, \ldots, \texttt{arg}_n\texttt{]}$. Тогда в корневом узле располагаем голову `head` выражения. Далее смежными ребрами, инцидентными вершине-голове,

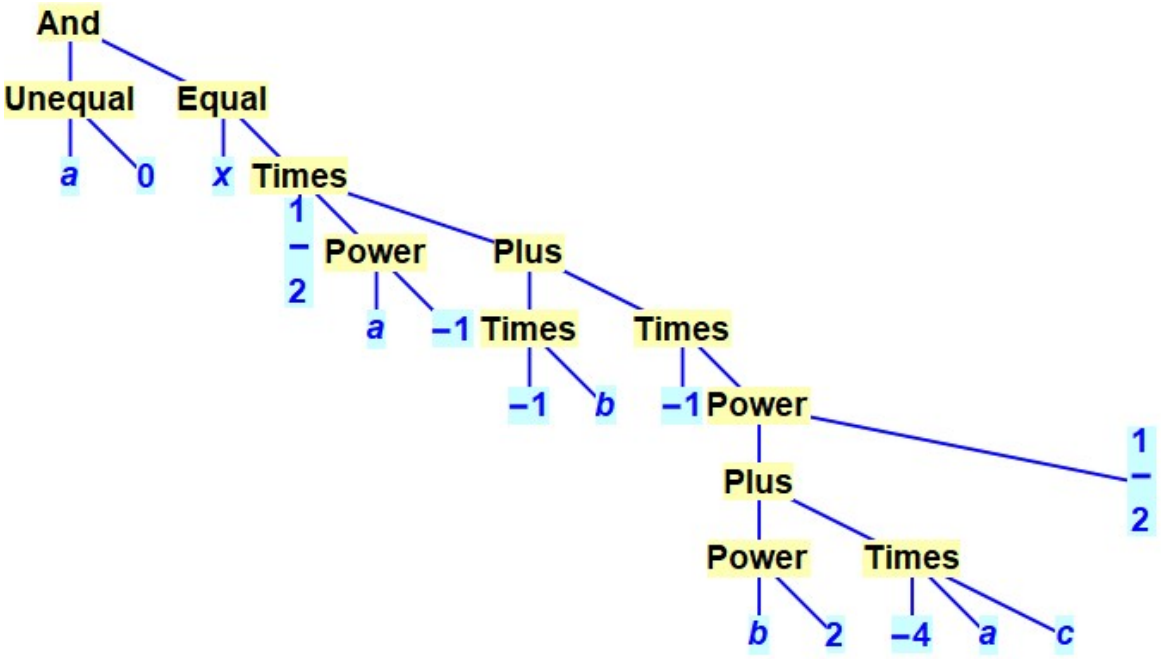
соединяем голову и корневые узлы поддеревьев-аргументов arg_i, $i = 1, \ldots, n$. Аргументы являются либо атомарными, либо не атомарными выражениями – таким образом, мы переходим к следующему шагу рекурсии.

Следуя соглашениям, описанным выше, выражение в виде дерева технически будем отображать сверху вниз, слева направо.

Например, выражение вида

$$a \neq 0 \;\&\&\; \left(x == \frac{-b - \sqrt{b^2 - 4\,a\,c}}{2\,a} \right)$$

по нашим соглашениям отображается следующим образом:



Если выражение не является атомарным, то на определенном уровне (по вертикали), в определенном месте (по строке) располагается голова выражения, глубже (на уровень ниже) располагаются аргументы-подвыражения, каждый из которых – выражение.

Если же выражение является атомарным, то на определенном уровне, в определенном месте отображается этот атом-выражение.

Задание 2.2 Спроектируйте глобальные правила ExprAsTree построения дерева выражения рекурсивным способом и опишите связи между этими правилами.

Выполнение задания 2.2

Спроектируем внешнюю функцию, функцию пользователя, которая строит и отображает заданное выражение expr в виде дерева.

Для имени функции используем символ ExprAsTree, не зарезервированный системой. Обязательным аргументом функции ExprAsTree является заданное выражение expr. Позволим также пользователю «наводить красоту»: второй, необязательный аргумент

options функции ExprAsTree даст возможность пользователю управлять режимами отображения графических объектов.

Итак, функция пользователя ExprAsTree – это глобальное правило (SetDelayed), левая часть которого представлена образцом ExprAsTree[expr_, options___Rule].
Возможность отсутствия второго аргумента функции ExprAsTree мы указали посредством образца (Pattern) BlankNullSequence, или тремя символами подчеркивания без пробелов.

Как будет работать функция пользователя ExprAsTree[expr, options], что будет делать?

Она передаст управление вычислением внутренней, технической функции ExprAsTree. Та обязана позаботиться о двух важных моментах: о построении графических объектов и об их отображении.

А именно, для построения графических объектов, узлов и ребер графа, фунция пользователя вызовет техническую функцию от двух обязательных аргументов ExprAsTree[what, were]. После того, как техническая функция подготовит графические объекты, функция пользователя ExprAsTree[expr, options] использует встроенную функцию Graphics, указывая режимы отображения options.

Таким образом, внешняя функция пользователя ExprAsTree[expr, options] имеет вид

```
ExprAsTree[expr_,
  options___Rule] :=
 Graphics[
  ExprAsTree[expr, {0, 0}],
  options]
```

Спроектируем далее порядок работы внутренней, технической функции ExprAsTree[what, were].

По нашему замыслу, эта техническая функция готовит графические объекты для отображения: узлы и ребра графа. При этом в каждом узле располагается либо атомарный объект, либо голова неатомарного выражения. Таким образом, случай обработки атомарного выражения отличен от случая работы с неатомарным выражением. Отличия в обработке приводят к написанию различных глобальных правил (SetDelayed).

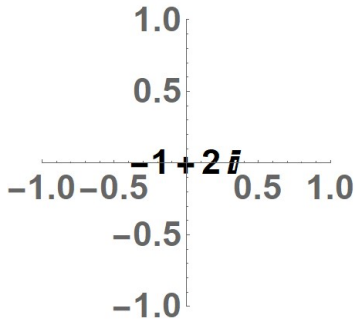
Правила ассоциируются с одним и тем же символом ExprAsTree, но аргументы этой функции описываются различными образцами: ExprAsTree[atom_?AtomQ,coord:{_,_}] и ExprAsTree[expr_,{x_,y_}].

Работа с атомарным выражением – это база рекурсии. Если выражение `what` атомарно, то техническая функция `ExprAsTree[what, were]` в корневом узле, расположенном в точке с координатами `were`, отображает атомарное выражение `what`. Функция строит графический объект, используя графический примитив `Text`.

```
ExprAsTree[atom_?AtomQ,
  coord:{_, _}] := Text[atom, coord]
```

Проверим работу введенных правил, укажем при этом режим отображения числовых осей, это поможет нам ориентироваться на плоскости. Для наглядности можно также использовать функции **`Trace`** или **`TracePrint`**.

```
ExprAsTree[-1 + 2 i, Axes → True,
 BaseStyle →
  {Medium, FontWeight → Bold}]
```

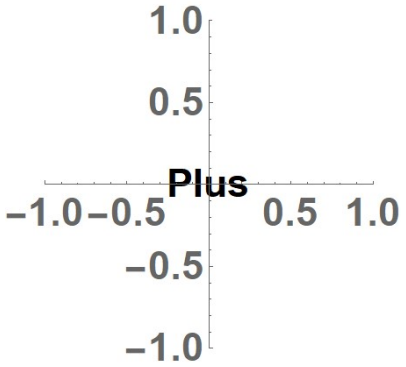


Таким образом, для случая отображения атомарного выражения искомая функция построена.

В случае, когда заданное выражение `expr` не является атомарным, оно имеет вид `head[subexpr1, subexpr2, ..., subexprn]`. Тогда в корневом узле следует отобразить голову выражения. Кроме того, из корневого узла должны выходить ребра (соединительные линии), соединяющие голову `head` и аргументы-подвыражения `subexpr1, subexpr2, ..., subexprn`, располагающиеся на уровень ниже.

```
ExprAsTree[expr_, {x_, y_}] :=
 {(*... Соединительные линии...*)
  Text[Head[expr], {x, y}]
  (*... Поддеревья,
  каждое в своей позиции ...*)}
```

```
ExprAsTree[a + 2 i, Axes → True,
 BaseStyle →
  {Large, FontWeight → Bold}]
```



Следовательно, для неатомарного выражения `expr` каждый его аргумент-подвыражение отображается в виде поддерева (то есть опять дерева!), имеющего свой корневой узел и т.д. Налицо рекурсия: техническая функция `ExprAsTree[what, were]`, чтобы построить графические объекты для отображения дерева, использует саму себя, техническую функцию `ExprAsTree[what, were]`, для построения графических объектов поддеревьев.

Как расположить графические объекты в прямоугольной декартовой системе координат?

Пусть корневой узел выражения `expr` имеет координаты `(x, y)`. Ребра, ему инцидентные, соединяют этот корневой узел с корневыми узлами поддеревьев `subexpr1, subexpr2, ..., subexprn`. Таким образом, для каждого неатомарного выражения `expr` нам следует знать координаты корневых узлов его поддеревьев (и только их). Эти координаты нам нужны, чтобы готовить графические объекты для отображения голов или атомов. Поэтому удобно иметь отдельную функцию, вычисляющую координаты корневых узлов поддеревьев или листьев, расположенных на уровень ниже, относительно заданной координаты `(x, y)` корневого узла дерева выражения `expr`.

`Задание 2.3` Постройте функцию `NodesCoord`, которая по заданной координате `(x, y)` корневого узла дерева выражения `expr` вычисляет координаты корневых узлов поддеревьев или листьев, соответствующих аргументам выражения `expr`.

`Выполнение задания 2.3`

Рассмотрим не атомарное выражение вида

`head[subexpr1, subexpr2, ..., subexprn].`

По соглашению, голова `head` располагается в корневом узле дерева. Пусть корневой узел имеет координаты `(x, y)`. Далее нам следует рассчитать координаты корневого узла каждого поддерева $\texttt{subexpr}_i$.

По соглашению, координаты поддеревьев располагаются в строке, ниже корневого узла `head`. Поддерево `subexpr1` имеет координаты корневого узла `{x, y-1}` (там будет отображаться либо атом `subexpr1`, либо голова аргумента `subexpr1`), поддерево `subexpr2` – координаты $\{x+w_1, y-1\}$,

поддерево subexpr3 – координаты {x+w_1+w_2, y-1}, ..., поддерево subexprn – координаты {x+w_1+w_2+...+w_{n-1}, y-1}, где w_i – ширина поддерева $subexpr_i$, i = 1, …, n.

Таким образом, каждый корневой узел поддерева $subexpr_i$ имеет одну и ту же ординату y-1, а его абсцисса зависит от «ширин» предыдущих аргументов-поддеревьев. Тогда выражение

{x+#, y-1}& /@ {0,w_1,w_1+w_2,w_1+w_2+w_3, ...,w_1+..+w_{n-1}}

сформирует список координат корневых узлов поддеревьев $subexpr_i$, i = 1, …, n.

Построим выражение, вычисляющее «накопительный» список ширин поддеревьев {0,w_1,w_1+w_2,w_1+w_2+w_3, ...,w_1+..+w_{n-1}}, или абсциссы корневых узлов поддеревьев.

Ширина w_i поддерева $subexpr_i$ совпадает с количеством атомарных объектов выражения $subexpr_i$, i = 1, …, n. В частности, ее вычисляет рекурсивная функция Width, построенная в [1, с. 66] (разобраться, задать вопросы, выучить!).

```
ClearAll[Width];
Width[_?AtomQ] = 1;
Width[expr_] :=
 Plus @@ Width /@ List @@ (expr)
```

Построим выражение, которое вычисляет координаты каждого из корневых узлов поддеревьев относительно корневого узла дерева с заданными координатами (x, y).

Для наглядности рассмотрим конкретное выражение, например

$$expr = \frac{1}{a^5 + b} - x^9 + 57 - v$$

Вычислим ширину каждого подвыражения, получим список

```
Width /@ List @@ (expr)
{1, 4, 2, 3}
```

Соответсвенно, вычислим отступы при определении абсциссы корневого узла для каждого поддерева, исключим ширину последнего поддерева

```
FoldList[Plus, 0, Width /@ List @@ (expr)] // Most
```

Таким образом, выражение, вычисляющее список координат корневых узлов поддеревьев относительно корневого узла дерева с координатами `(x, y)`, имеет вид

```
{x+#, y-1}& /@
FoldList[Plus, 0, Width /@ List @@ (expr)] // Most
```

Напишите функцию, глобальное правило `NodesCoord`, которая по заданной координате `(x, y)` корневого узла дерева неатомарного выражения `expr` вычисляет координаты корневых узлов поддеревьев, соответствующих аргументам выражения `expr`. Используйте рассуждения, приведенные выше.

`Задание 2.4` Постройте техническую функцию `ExprAsTree[what, were]`, которая готовит графические объекты-узлы дерева, каждый в своей позиции.

`Выполнение задания 2.4`

Для подготовки графических объектов используем графический примитив `Text[what, were]`. Чтобы ответить на вопрос, ЧТО и ГДЕ отображать, формируем список вида

```
{список аргументов выражения expr,
    список координат соответствующих корневых узлов}.
```

Далее, используя функцию **`MapThread`** и примитив **`Text`**, можно подготовить к отображению аргументы, каждый в своей позиции

```
MapThread[Text[#1, #2] &,
 {(*список аргументов,
  список координат соответствующих корневых узлов*)}
]
```

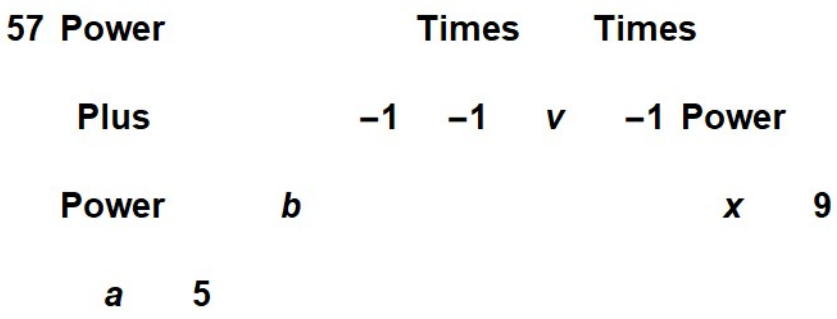
$$\left\{\text{Text}[57, \{x, -1+y\}], \text{Text}\left[\frac{1}{a^5+b}, \{1+x, -1+y\}\right],\right.$$
$$\left.\text{Text}[-v, \{5+x, -1+y\}], \text{Text}\left[-x^9, \{7+x, -1+y\}\right]\right\}$$

Модифицируем правило (техническую функцию), которое мы построили ранее для отображения неатомарного выражения. При этом будем учитывать, что в случае, когда аргумент выражения `expr` не атом, следует отображать лишь голову, а далее рекурсивно работать с аргументами, соответствующими этой голове. Поэтому реализуем рекурсивный вызов функции `ExprAsTree[what, were]`, его укажем вместо **`Text`** в аргументах функции **`MapThread`**

```
ExprAsTree[expr_, {x_, y_}] :=
  {(*... Соединительные линии...*)
   Text[Head[expr], {x, y}],
   MapThread[ExprAsTree[#1, #2] &,
    {(*список аргументов,
     список координат соответствующих корневых узлов*)}]};
```

Проверим, как теперь работает функция `ExprAsTree`. Вызываем функцию пользователя

```
ExprAsTree[expr, BaseStyle → {Large, FontWeight → Bold}]
```



`Задание 2.5` Модифицируйте функцию `ExprAsTree` таким образом, чтобы она отображала также соединительные линии, ребра графа. При визуализации используйте возможности цвета.

`Выполнение задания 2.5`

Этот пункт задания выполните самостоятельно.

`Задание 2.6` Модифицируйте функцию `ExprAsTree` таким образом, чтобы при наведении указателя Мыши на корневой узел всплывала подсказка (`Tooltip`), содержащая текст подвыражения, соответствующего указываемому узлу.

`Выполнение задания 2.6`

Выполните самостоятельно.

`Задание 2.7` Перепишите функцию `ExprAsTree` таким образом, чтобы

1) голова выражения находилась по центру выражения;
2) учесть, что голова выражения также является выражением, не всегда атомарным.

2. Рекурсия. Числа Фибоначчи и функция Аккермана.

`Задание 1.1` Напишите функцию, которая вычисляет последовательность чисел Фибоначчи. Используйте рекурсивный способ задания, запоминая при этом найденные значения функции.

`Задание 1.2` Напишите функцию, которая вычисляет значения функции Аккермана.

Ackerman(0, n) = n + 1;

Ackerman(m, 0) = Ackerman(m – 1, 1);

Ackerman(m, n) =

Ackerman(m – 1, Ackerman (m, n – 1))

Используя рекурсивный способ задания, запоминайте при этом найденные значения функции Аккермана (см. `Help, Functions That Remember Values They Have Found`). Постройте график функции Аккермана, используйте `ListPlot3D`.

3. Закрепление навыков и умений

Оформите выполненную работу: форматируйте документ, напишите краткие комментарии, дополните их локальными выводами.

Запишите все встроенные функции *Mathematica*, которые Вы использовали при выполнении практикума. Выучите назначение этих функций, закрепите навыки работы с ними, используя «живые примеры» в документах системы справки.

Литература

1. Голубева Л.Л., Малевич А.Э., Щеглова Н.Л. Компьютерная математика. Символьный пакет *Mathematica*. Лаб. практикум в 2 ч. Ч 1. - Минск: БГУ, 2012. – 235 с.