Доступ к частям выражения

В компьютерной алгебре попытки добиться желаемого результата путём выполнения преобразований над всем выражением редко бывают успешными. Чаще всего подобный подход приводит к лавинообразному росту объёма промежуточных результатов и заканчивается полным исчерпанием имеющихся вычислительных ресурсов.

Более рациональный подход состоит в точечном воздействии на интересующее нас выражение. Для этого необходимо уметь выделять требуемые фрагменты выражения и трансформировать только их, не затрагивая остальные части.

Для доступа к частям выражения система Maple предоставляет низкоуровневую функцию *op*. Название этой функции является англоязычной аббревиатурой слова operand. С её помощью можно извлечь из выражения необходимые нам операнды.

В простейшем случае необходимо указать порядковый номер требуемого операнда и само выражение, из которого следует произвести извлечение. Преположим, что задано выражение

```
> expr:=sin(Pi/8)^4+cos(3*Pi/8)^4+sin(5*Pi/8)^4+cos(7*Pi/8)^4;
```

$$expr := \sin\left(\frac{1}{8}\,\pi\right)^4 + \cos\left(\frac{3}{8}\,\pi\right)^4 + \sin\left(\frac{3}{8}\,\pi\right)^4 + \cos\left(\frac{1}{8}\,\pi\right)^4$$

Выделим второе слагаемое, указав его номер. Отсчёт слагаемых ведётся слева направо.

```
> op(2,expr);
```

$$\cos\left(\frac{3}{8}\,\pi\right)^4$$

Если операнда с заданным номером не существует, то получим сообщение об ошибке.

```
> op(5,expr);
Error, improper op or subscript selector
```

Если первый аргумент *op* является целым отрицательным числом, то это является указанием на то, что отсчёт будет идти справа налево. Типичное рассуждение будет звучать так: "возьмём последнее слагаемое".

```
> op(-1,expr);
```

$$\cos\left(\frac{1}{8}\,\pi\right)^4$$

При использовании команды *op* полезно знать сколько вообще операндов содержится в выражении. Для этого предназначена функция *nops*. Её название расшифровывается как number of operands. Единственным аргументом *nops* является интересующее нас выражение.

Убедимся в том, что переменная *expr* содержит четыре слагаемых.

```
> nops(expr);
```

$$4$$

Для некоторых структурированных типов данных с помощью команды *nops* можно узнать количество компонент, содержащихся в переменной указанного типа. Предположим, что нам задан список

```
> L:=[1,4,9,16,25,36,49];
```

$$L := [1, 4, 9, 16, 25, 36, 49]$$

Простейший способ определить количество элементов в списке состоит в вызове команды *nops*.

```
> nops(L);
```

$$7$$

Этот же приём срабатывает и для объектов типа множество. Пусть, например,

```
> S:={1,8,27,64,125};
```

$$S := \{1, 8, 27, 64, 125\}$$

Тогда количество элементов во множестве определяется аналогично предыдущему

случаю.

```
> nops(S);
```

$$5$$

Однако с аналогиями не следует заходить слишком далеко. Преобразуем список в вектор.

```
> vL:=convert(L,vector);
```

$$vL := \left[\begin{array}{ccccccc} 1 & 4 & 9 & 16 & 25 & 36 & 49 \end{array}\right]$$

Как мы видим, количество компонент вектора невозможно узнать с помощью команды *nops*.

```
> nops(vL);
```

$$1$$

Параметры индексированных переменных следует уточнять, применяя команду *numelems*. Эта команда была введена в Maple 15 для того, чтобы корректно возвращать характеристики объектов, которые могут храниться в памяти системы в разреженном виде.

```
> numelems(vL);
```

$$7$$

Её не следует применять для обычных выражений.

```
> numelems(expr);
```

```
Error, invalid input: numelems expects its 1st argument, t, to be of type indexable, but received sin((1/8)*Pi)^4+cos((3/8)*Pi)^4+sin((3/8)*Pi)^4+cos((1/8)*Pi)^4
```

Несмотря на то, что строка состоит из литер,

```
> str:="Hello world!";
```

$$str := \text{"Hello world!"}$$

в Maple она рассматривается как неделимый объект, то есть имеет тип *atomic*. К такого рода объектам функция *nops* неприменима.

```
> type(str,atomic);
  nops(str);
```

$$true$$

$$1$$

В то же время команда *numelems* знает как находить число символов в строке.

```
> numelems(str);
```

$$12$$

Для строк более естественной может показаться команда *length*, поскольку в других языках именно так называется функция, вычисляющая длину строки.

```
> length(str);
```

$$12$$

В соответствии с документацией, если первый аргумент *i* является целым отрицательным числом, то вызов *op(i,expr)* равнозначен вызову *op(nops(expr)+i+1,expr)*. Таким образом, для извлечения предпоследнего слагаемого необходимо указать

```
> op(-2,expr);
```

$$\sin\left(\frac{3}{8}\,\pi\right)^4$$

Из вышеуказанного замечания следует, что отрицательные значения первого аргумента по модулю не должны превышать количества слагаемых. Иначе мы опять получим сообщение об ошибке.

```
> op(-6,expr);
```

```
Error, improper op or subscript selector
```

Весьма полезной особенностью команды *op* является возможность извлечения сразу нескольких операндов. Для этого её первый параметр должен представлять собой диапазон вида *i..j*. Результатом выполнения такой команды будет последовательность операндов заданного выражения, начиная с *i*-го и кончая *j*-ым.

```
> op(2..3,expr);
```

$$\cos\left(\frac{3}{8}\pi\right)^4, \sin\left(\frac{3}{8}\pi\right)^4$$

Если нижняя граница диапазона окажется больше верхней, то получаем пустую последовательность, то есть *NULL*.

```
> op(3..2,expr);
```

Границы диапазона могут быть и отрицательными целыми числами

```
> op(-3..-2,expr);
```

$$\cos\left(\frac{3}{8}\pi\right)^4, \sin\left(\frac{3}{8}\pi\right)^4$$

однако нижняя граница в любом случае не должна превышать верхнюю. Невыполнение этого условия приведёт к пустой последовательности.

```
> op(-2..-3,expr);
```

С учётом правил пересчёта отрицательных аргументов в положительные, корректной оказывается и такая форма обращения к функции, когда одна из границ диапазона является положительной величиной, а другая отрицательной.

```
> op(2..-2,expr);
```

$$\cos\left(\frac{3}{8}\pi\right)^4, \sin\left(\frac{3}{8}\pi\right)^4$$

Команду *op* можно вызвать только с одним аргументом, которым является выражение. При этом произойдёт разложение выражения на операнды. Формально такой вызов с одним аргументом идентичен вызову *op(1..nops(expr),expr)*.

```
> op(expr);
```

$$\sin\left(\frac{1}{8}\pi\right)^4, \cos\left(\frac{3}{8}\pi\right)^4, \sin\left(\frac{3}{8}\pi\right)^4, \cos\left(\frac{1}{8}\pi\right)^4$$

Подобные вызовы оказываются весьма полезными при модификации списков, которые являются немутируемыми объектами. Например, для того, чтобы добавить ещё один элемент в конец списка следует выполнить команду

```
> L:=[op(L),64];
```

$$L := [1, 4, 9, 16, 25, 36, 49, 64]$$

Первым аргументом *op* может быть список. Элементы этого списка интерпретируются как подоперанды исходного выражения на возрастающих уровнях вложенности.

Если нас интересует аргумент тригонометрической функции в третьем слагаемом после автоупрощения, то следует выполнить команду

```
> op([3,1,1],expr);
```

$$\frac{3}{8}\pi$$

Формально вызов $op([a_1, a_2, a_3], expr)$ эквивалентен вызову $op(a_3,op(a_2, op(a_1, expr)))$, но выполняется быстрее.

Когда выражением оказывается вызов некоторой функции, то команда *op* перечисляет её аргументы. Пусть *f* является функцией с тремя аргументами.

```
> expr1:=f(x^2,y-2*x,x^3+g(x,y));
```

$$expr1 := f\left(x^2, y - 2x, x^3 + g(x, y)\right)$$

Тогда список аргументов извлекается путём вызова

```
> op(expr1);
```

$$x^2, y - 2x, x^3 + g(x, y)$$

Следовательно, для нахождения второго аргумента функции *g* можно выполнить любую из команд

```
> op(-1,op(-1,op(-1,expr1)));op([-1,-1,-1],expr1);
```

$$y$$
$$y$$

Если выражение, поступающее на вход команды *op*, представляет собой один единственный объект, то в качестве первого параметра можно указать операнд с

нулевым индексом. Такая форма вызова позволяет получить информацию об этом объекте.

В том случае, когда объект является обращением к функции, можно узнать имя функции. Возвращаясь к предыдущему примеру, получаем

```
> op(0,expr1);
```

$$f$$

Для неинициализированных переменных, а также для объектов, подчиняющихся особым правилам вычислений, такими как массивы, вектора, матрицы и другие аналогичные структуры, вызов команды *op* с указанием в качестве первого параметра операнда с нулевым номером выдаст результат *symbol*.

```
> op(0,ZZ);op(0,vL);
```

$$symbol$$

$$symbol$$

Совершенно неочевидным является понятие операнда для выражений, являющихся рядами. В качестве примера рассмотрим разложение в степенной ряд функции tg(x) в точке $x = \pi$.

```
> expr2:=series(tan(x),x=Pi);
```

$$expr2 := x - \pi + \frac{1}{3}\,(x-\pi)^3 + \frac{2}{15}\,(x-\pi)^5 + \mathrm{O}\left((x-\pi)^7\right)$$

На первый взгляд кажется, что усечённый ряд содержит 5 слагаемых. Однако фактическое количество операндов больше.

```
> nops(expr2);
```

$$8$$

Посмотрим, что они собой представляют.

```
> op(expr2);
```

$$1, 1, \frac{1}{3}, 3, \frac{2}{15}, 5, \mathrm{O}(1), 7$$

Итак, операнды с чётными номерами представляют собой степени переменной, по которой ведётся разложение. Нечётные операнды являются числовыми коэффициентами данного степенного ряда. Чтобы выделить, например, коэффициент при $(x-\pi)^3$, следует выполнить команду

```
> op(3,expr2);
```

$$\frac{1}{3}$$

Сама переменная величина как раз и является нулевым операндом степенного ряда.

```
> op(0,expr2);
```

$$x - \pi$$

В большинстве других случаев, если выражением оказывается одиночный объект, то выполнение команды *op* с нулевым первым аргументом позволяет узнать тип этого объекта. Например,

```
> op(0,L);
```

$$list$$

Поскольку все элементы списка L целые, то

```
> op(0,L[2]);op(0,L[3]);
```

$$Integer$$

$$Integer$$

Более сложные структуры данных не являются исключением. В частности, с точки зрения Maple графики являются низкоуровневыми объектами *PLOT* или *PLOT3D*.

```
> P:=plot(convert(expr2,polynom),x=0..2*Pi):
  op(0,P);
```

$$PLOT$$

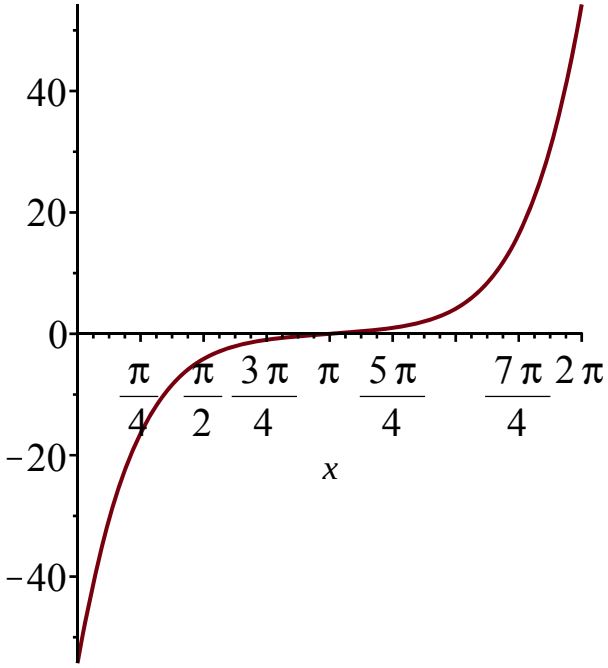
С точки зрения синтаксиса *PLOT* и *PLOT3D* являются функциями

```
> whattype(P);
```

$$function$$

и стандартная процедура форматированной печати (prettyprinter) знает, как выводить на экран результаты работы этих функций.

```
> P;
```



Ранее уже упоминалось о возможности использования диапазона в качестве первого аргумента *op*. Данная возможность допускает естественное обобщение на случай, когда номер операнда является списком. Последний элемент этого списка может быть диапазоном.

```
> op([-1,-1,1..2],expr1);
```

$$x, y$$

Формально, вызов *op([a1, a2, a3*$_1$*..a3*$_2$*], expr)* эквивалентен выполнению команды *op(a3*$_1$*..a3*$_2$*, op(a2, op(a1, e)))*, но при этом допускает более эффективную реализацию.

При использовании диапазонов в команде *op* система Maple позаимствовала ключевые идеи о вырезках и сечениях массивов из языка Fortran. По этой причине допустимой является следующая команда

```
> op([-1,-1,..],expr1);
```

$$x, y$$

Такая возможность существовала и в некоторых прошлых версиях системы, однако была недокументированной.