

## Re: EE Student, Edit, Proposal Masters, Help (concepts of functional programming, symbolic programming and MATLAB)

**Source:** <http://sci.tech-archive.net/Archive/sci.math.symbolic/2005-03/0089.html>

---

**From:** John Creighton ([JohnCreighton\\_\\_at\\_hotmail.com](mailto:JohnCreighton__at_hotmail.com))

**Date:** 03/08/05

Date: 7 Mar 2005 21:21:56 -0800

In article <c5e803c2.0503061419.2fee6687@posting.google.com>,

John Creighton <[JohnCreighton\\_\\_at\\_hotmail.com](mailto:JohnCreighton__at_hotmail.com)> wrote:

>I am an EE student and in part of my thesis I use MAPLE to  
>symbolically derive the RIDF for a Kalman filter. This worked but as  
>part of my proposal I discuss symbolic math packages in the context  
of  
>functional programming. Since, I am not widely read in the area I  
>>would appreciate it if someone could look over what I read to make  
>sure everything I say is at least somewhat true. I am currently  
>reading more references to strengthen this aspect of my proposal. All  
>help is appreciated.  
>1.1.3 A few comments on Symbolic and Functional Programming  
>Symbolic programming languages fall under two categories, functional  
>(e.g. lisp, Maple) and logical (e.g. PROLOG). In functional  
>programming languages each program is a function similar to a  
>mathematical function. A program is made up of a composition of  
>functions [9]. The arguments to the functions can include the data  
>types, lists, expressions, primitives and variables. All data types  
>are built up of elemental units called primitives. In Maple an  
>algebraic variables contains a pointer.

"Nearly everything has pointers. The low-level internal  
representation  
of data is not ordinarily of interest to the user."

Perhaps from an abstract mathematical perspective the low level  
interpretation is not important except when the low level behavior has  
a direct consequence on the abstract high level behavior. In the case  
of MAPLE the pointers are not completely encapsulated. Consider the  
example given in

Rofer Kraft, Programming In Maple

<http://adept.maplesoft.com/powertools/programming/html/2.01and2.02.html>

"Now make x a name for y , y a name for z , and give z a numeric

value, in that order.

```
> x := y;
```

```
> y := z;
```

```
> z := 3;
```

```
      x:=y
```

```
      y:=z
```

```
      z:=3
```

What does x evaluate to now?

```
> x;
```

```
      3
```

Change the value of z .

```
> z := 5;
```

```
      X:=5
```

Check the value of x .

```
> x;
```

```
      5
```

In this example, x points to y , y points to z , and at first z pointed to 3, so x (and y ) evaluated to 3. Then z was pointed at 5, so x (and y ) evaluated to 5. This is full evaluation. Maple keeps following the trail of assignments until it gets to a "dead end", either a numeric value or an unassigned name. "

> *The pointer will point to null*

>*until a value is substituted for the variable.*

"You don't substitute a value for the variable, you assign a value to the variable."

Okay, I see my mistake here. Although you can substitute a value for a variable in an expression usingm "subs" or "algsb" the pointer stored in the variable for which an expression or variable is being substituted will no longer have anything to do with the resulting expression. In contrast if you assign a variable a value which is in the expression then the value the variable points to is linked to the expression though the variable.

> *At that point the*

>*variable points to the value subsisted, which could be another*

>*variable, an entire expression or a primitive. If the value*

>*substituted is a primitive, then a numeric value can be obtained by*

>*evaluating the variable.*

"I don't know what you mean by a "primitive". That word is not part of the usual description of Maple. An object of type "numeric" is an integer, a fraction or a float."

Come to think of it I agree. My statement is not quite right. In MAPLE a numeric variable can contain an indeterminate number of bytes resulting in numbers which use more bytes than a native machine type.

Thus in Maple a numeric variable is not a primitive data type.

> *A pointer abstracts the manipulation of data*  
> *in a manner similar to the way variables in mathematics abstract the*  
> *manipulation of numbers. That is the rules of algebra can be used to*  
> *manipulate the variables without having complete knowledge of the*  
> *underlying structure.*

"I don't know what you're getting at here."

I'll rethink this. Not to sound too obvious but the elements manipulated are often a mapping to or an obstruction of the actual object which are of primary concern in the computation. Some times these mappings are given such names as homomorphism and isomorphism.

> *Expressions describe how data is evaluated mathematically in terms of*  
> *data (e.g. variables) and functions (e.g plus). Expressions can be*  
> *represented with strings but are usually represented as a list.*

"No, expressions are not lists. A list is a particular type of expression."

Do you have a reference for this? I wonder if the definition of a list is agreed upon in the context of computer science. In MATLAB a list separates function input arguments when calling a function and function output arguments when the function returns its values. A list has the property that each element is of indeterminate size. A cell array in MATLAB is also a data structure which maps an n tuple of integers to elements of indeterminate size. The natural way to represent such a data structure is with an array of pointers and when the array is one dimensional it has the same properties of a list. Perhaps it would be best to see how list is defined in the language lisp given lisp was the second programming language and stands for list processing.

By the very sound of its name I would of took the word expression to be something which can be evaluated. Continuing with my reasoning I would think that a list would be no more a type of expression then a numeric would be an expression and a number a set.

> *The*  
> *MATLAB equivalent of a list is a one dimensional cell array. In an*  
> *expression, the first element of the list is the function or*  
> *operation. The subsequent elements are the arguments. Arguments can*  
> *be*  
> *any data structure including expressions. When the arguments of the*  
> *functions in an expression are expressions it becomes natural to*  
> *represent the expression as a tree. This representation is known as*  
> *the operation tree. Unlike the flow of procedural programming*  
> *languages, in functional programming language the outer function is*  
> *evaluated first and the inner function is only evaluated if called by*  
> *the outer function. This is referred to as lazy evaluation.*

"Maple does not use lazy evaluation. When an expression is evaluated, it is ordinarily evaluated from the inside out: by default, a function evaluates its arguments first. There are some exceptions."

I do know that you can explore the operation tree with the functions:

op  
type  
whattype

It would seem to me that since expressions are stored in an operation tree that to get to the innermost argument you must start from the outside and work inward. Perhaps expressions are by default automatically evaluated by maple before being passed to a function. And perhaps there are ways to prevent this. Given a simply command like collect or expand it would seem like a waste of computer cycles to reevaluate the expression before the function which simplifies the expression is carried out. I am interested to here more on this do you have any good references? Anyway, I still think a natural way to represent an operation tree is with an array of pointers where the first pointer points to the operation and subsequent pointers point to the elements. Given an expression can be represented by an operation tree you must see how I presumed that expressions were represented by lists.

Returning to the concept of lazy evaluation lets look at the example given in Roger Craft Programming in Maple, levels of evaluation <http://www.imada.sdu.dk/~hjm/MM91/Copy%20of%20Programming/0.TableOfContents1.html>

### "2.3 Levels of evaluation

Let us go back to the first example of full evaluation defined earlier.

```
> x:='x': y:='y': z:='z':  
> x:=y;  
> y:=z;  
> z:=3;
```

x=3

Now if we ask Maple to evaluate x , Maple uses full evaluation and returns 3.

```
> x;
```

3

But for Maple to get from x to 3 it had to go through the intermediate values of y and z . There is a name for these intermediate values and there is a way to access them also. The name

for these intermediate values is levels of evaluation and we can control the level of evaluation using a special form of the eval command. For example, here is how we get one level of evaluation of x , which returns the name y .

```
> eval( x, 1 );
```

y

Here is how we get two levels of evaluation of x , which returns the name z .

```
> eval( x, 2 );
```

z

Here is how we get three levels of evaluation of x , which returns the value 3.

```
> eval( x, 3 );
```

3

We can ask for four or higher levels of evaluation, but 3 always evaluates to 3.

```
> eval( x, 4 );
```

3

"

Anyway, to me in this example it looks like the evaluation begun from the outside and worked inwards. Perhaps this is not general case.

Thanks for all you your help. Given a few things I said seem to be in error it seemed worthwhile to post for help in clarifying these concepts. All feedback and suggested references are greatly appertained.