

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/2005-03/0114.html>

From: John Creighton (*JohnCreighton__at_hotmail.com*)

Date: 03/11/05

Date: 10 Mar 2005 20:24:02 -0800

Richard Fateman <fateman@cs.berkeley.edu> wrote in message
news:<422F2A1F.1000007@cs.berkeley.edu>...

> *The simple fact that you want to introduce is, I think, that programming*
> *systems exist which allow the programmer to use symbols such as x,y,z ,*
> *and expressions such as $x+y$, $\cos(z)$, etc in a way similar to conventional*
> *numeric values.*

I may do as you suggest here for my proposal. However, in my thesis I
would like to cover the topic slightly more toughly. Here is my latest
draft on the section:

1.1.3 Some Notes on: Computer Algebra, Symbolic Programming and Functional Programming

Computer Algebra systems (CAS) is a modern synonym for a symbolic math
package (e.g. Maple) It is easy to confuse the phrarse symbolic math
package with the phrase symbolic programming. Symbolic programming
languages fall under two categories, functional (e.g. Haskell) and
logical (e.g. PROLOG). Lisp was the first functional programming
language and the second programming language. Lisp has since been
expanded to include many programming paradigms and is now known as
Common Lisp (CLisp). Although, Common Lisp is no longer a pure
functional language some functional programming language lisp dialects
have been created (e.g. Scheme).

Computer algebra systems were originally conceived a century prior to
there first use in the 1970's. The first CAS (e.g. MACSYMA, REDUCE) we
under use in the 1970's on Lisp based mainframe systems at such
institutions as MIT. These systems required megabytes of ram which at
the time was beyond the power of personal computers. MAPLE was
conceived in the 1980's at Waterloo University. The goal was to stress
space efficiency as well as time efficiency. To do this a system
design language of the (BCPL) family was the obvious tool. Initially
Maple was implemented in B on a Honewell but shortly switched to C
since C was the more widely available implementation language.

Although Maple was implemented in a procedural programming language, the MAPLE syntax has both elements of procedural and functional programming. 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. As in all languages data types are built up of elemental units called primitives. However, in some third generation languages (e.g. Maple) the primitives are not part of the language. This is analogous to assembler and machine code falling outside the scope of a second generation language. Maple uses a numeric data type which can be of arbitrary precision. The software must decide how to reduce the higher level computation down to machine level operations on primitive data.

In a pure functional (e.g. Haskell) languages such as Haskell [39] assignment is not allowed in order to eliminate side effects. Alternatively monads are used to solve problems such as exception handling which is difficult to do without assignment. This is not necessary in Maple because Maple is not a pure functional language. In Maple assignment can be done as in a procedural style:

```
x:=2  
x:=x+1
```

Alternatively chains of assignment can be set up:

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

In which case x will evaluate to 3. The assignment chain is not altered by evaluation. It is only altered when one of the variables in the chain is reassigned. So for instance the chain could be altered by assigning the first variable in the chain the value at the end of the chain as follows:

```
x:=eval(x)
```

In the above example if the single quotes were left off the assignment change would only be the same if y was previously unassigned. However, if the single quotes were left off and y was previously assigned then x would be assigned the value of y.

Variables can represent many kinds of data structures. A data structure that can be evaluated is known as an expression. An expression could be formed from a single variable x, a single function at a given domain f(x) or through the composition of several expressions f(g(x)).

Another data structure is a list. Linguistically a list is an ordered collection of items. In computer science a list usually refers to a data structure that is easy to add and remove stuff from but must be accessed in a sequential fashion. In a purely functional language a list can be defined by the composition of functions in a recursive manner. For instance the list function could take two arguments where one argument is the first element of the list and the next argument is

the rest of the list. Since, a list can be represented as a composition of functions; functions can be evaluated and; expressions are something that can be evaluated then: a list could be thought of as an expression. Conversely an expression could be represented with a composition of lists by letting the first element of the list be a function name or reference and the subsequent elements be the arguments where the arguments can also be lists of the same format.

When the arguments of functions are compound expressions it becomes natural to represent the overall expression as a tree. This representation is known as the operation tree. Given that in an operation tree the outer function is the easiest to access, there are certain efficiency advantages to evaluating the outer function first. For instance if the outer function involves the multiplication of an expression by zero it may not be necessary to know the exact nature of the expression which is being multiplied by zero to determine the result. The evaluation of the outer function (e.g. multiplication) first and only evaluating the arguments if called by the inner functions if called by the outer function is referred to as lazy evaluation. A function programming language which does not use lazy evaluation is known as a strict functional language.

The symbolic package for MATLAB performs symbolic operations by storing symbolic expressions as objects. MATLAB manipulates expressions by: passing the expression to the Maple kernel and then storing the resulting string in a field of a symbolic object [7]. The Method used in MATLAB is not the most efficient way to perform symbolic operations but was chosen given the limitation of MATLAB in dealing with variable references and pointers. For the proposed research the efficiency limitations are not usually a serious concern since the symbolic computations must only be done in the design phase. However, there is a size limitation on the size of a symbolic variables that can be used within the symbolic package.