Proceedings of the 23rd ACM SIGCSE Symposium,
SIGCSE 1992, Kansas City, MO., March 5-6, 1992.

Introducing Functional Programming in Discrete Mathematics

Roger L. Wainwright
Department of Mathematical and Computer Sciences
The University of Tulsa
600 South College Avenue
Tulsa, Oklahoma 74104-3189
rogerw@tusun2.mcs.utulsa.edu

ABSTRACT

Programming assignments in my discrete mathematics course have changed recently due to an influx of non-computer science students with little or no programming experience. Programming problems are now assigned in a simple to learn, easy to write, mathematical-like functional programming language that requires no previous programming experience. In theory, all students begin on the same basis. Exposure to the concepts of functional programming is an essential part of computer science and mathematics curricula. For most students this is the only exposure to functional programming. Functional programming and discrete mathematics are a natural combination. One week of lectures and perhaps a small monetary investment is all that is required. An instructor totally unfamiliar with functional programming can easily learn enough in a week or so to present a simple introduction to the topic. Introducing functional programming concepts in discrete mathematics was very successful. Students found the exposure to functional programming to be an insight they had never experienced before and enthusiastically recommended an introduction to functional programming be a permanent part of the course.

1. INTRODUCTION

I have taught Discrete Mathematics off and on over the past 15 years. The course has evolved over the years, but for the most part has been taught as a "traditional" computer science discrete mathematics course. The course is taught at the second semester sophomore level for computer science majors. The prerequisites include a freshman two semester programming sequence and Calculus I. Programming assignments have always been part of the course. Usually three programs are assigned; the last one is rather complicated involving some detailed aspect of graph theory. The students almost exclusively choose to program in C.

A recent change in the Mathematics degree requirements has had an impact on the student population in the discrete mathematics course. Discrete mathematics is currently cross listed as a mathematics course. Mathematics and mathematics education majors are currently allowed to take discrete mathematics as a mathematics elective. As a result, the student population in the discrete mathematics course changed recently from almost 100% computer science majors to 70% computer science majors and 30% mathematics majors.

The Problem: mathematics students enter the course with little or no prior computing background. Mathematics students who minor in computer science have the necessary background, but mathematics majors who minor in engineering or operations research, or education at our university are not required to take the prerequisite programming courses. There is insufficient staff to justify two separate courses one for each major, and there are not enough students to justify two sections.

Solution I: Teach the course without any programming assignments. This is the simplest solution. It is also an acceptable solution. Many discrete mathematics courses are taught with no programming assignments even when all of the students are computer science

majors. Programming is certainly not an integral part of the course, and a rigorous discrete mathematics course can be taught without any programming assignments.

Solution II: Assign programming problems in a simple to learn, easy to write, mathematical-like functional programming language that requires no previous programming experience. The concepts of functional programming are easy and natural for mathematics and computer science students. Exposure to functional programming is an essential part of a computer science curriculum, and a discrete mathematics course is an excellent opportunity to present an introduction to functional programming. For most students this is the only exposure to functional programming. Functional programming techniques are helpful as a specification tool for algorithms, theorems, and other concepts. At this point none of the students (computer science or mathematics) have studied functional languages before. In theory, all students begin on the same basis. The mathematics students should not be at any disadvantage having little or no prior programming experience.

2. FUNCTIONAL LANGUAGES

This overview of functional languages is intended for the reader with little or no exposure to functional programming. In a functional programming language, programs consist entirely of functions. Functional programming languages emphasize describing the results of a computation and focus less on how to perform the computation. This relieves the programmer of the burden of describing the flow of control. Functional programs contain no assignment statements, in the sense that once a variable is given a value it can never be changed. Functional programs contain no side-effects of any kind. Thus expressions can be evaluated at any time, in any order. Variables can be replaced by their values at any time. This allows the program to be modularized into small simple modules. Functional programming languages have many advantages. Programs are written in a simple succinct mathematical-like notation. Consequently, functional programs tend to be easier to read, write, verify and modify than programs written in a procedural language. Using a functional (declarative) programming language, the programmer approaches a problem at a higher level of abstraction. Modern functional languages support pattern matching, user-defined data types, abstract data types, polymorphic types, higher order functions, infinite data structures by lazy evaluation, currying functions, and other features. Burton and Yang[2] and Hughs[3] give excellent overviews and illustrate the benefits of functional programming languages.

In my course, I introduced the students to the functional language, Miranda. Miranda is a trademark of Research Software Ltd. Although Miranda supports several data types, for simplicity, only the data type, list, is used. A list of the first 5 integers in Miranda is represented as [1,2,3,4,5] or [1..5]. Lists are concatenated using ++ (ie., [1,2] ++ [3,4,5] is [1..5]), and an element is "cons" to a list using : (ie., 1:[2,4] is [1,2,4]). Consider the following quicksort function (supplied by Miranda) that sorts a list of numbers. Notice the succinct mathematical-like notation specifying what is to be done and not how to do it.

```
>qsort []     = []
>qsort (a:x) = qsort[b | b<-x; b<=a] ++ [a] ++
>                    qsort[b | b<-x; b>a]
```

The function is written recursively. The base case is listed first and indicates that qsort of an empty list returns an empty list. The notation (a:x) in Miranda is used to represent a list where *a* is the first element, and *x* represents all of the remaining elements of the list, if any. Thus qsort of a non-empty list results in three lists concatenated together. The first list, for example, is the result of qsort on the list of elements *b* such that *b* comes from *x* and b<=a. Notice the "pivot" is the first element in the list, *a*. Furthermore, the details of the "partitioning" algorithm are not specified, only that a partitioning must be done. This function covers all possible arguments: an empty list and a non-empty list. Miranda will match the actual argument to each qsort function to determine which of the two qsort functions to invoke. Consider the following function for calculating all primes.

```
>primes     = sieve[2..]
>sieve(p:x) = p:sieve[n | n<-x; n mod p ~=0]
```

The function primes has no arguments and invokes the sieve function with an infinite list! This is an excellent example of lazy evaluation, which means that an expression is evaluated as it is needed. Hence the values in [2..] are generated as needed. Furthermore, as each

value for *n* comes from the infinite list *x*, another sieve function can be invoked. Furthermore, when part of the result represented by *p* is determined, it is displayed to the screen.

The reader is encouraged to study the following functions to discover how succinct and easy to read they are, even with no prior exposure to Miranda or functional programming. Guards (conditions) appear after a comma at the end of a statement. A statement is performed only when the guard is true. Note x!n is the same as a subscript, and represents the *nth* element of x using 0 origin. Furthermore, #x represents the length of list, x.

```
>|| Sort a list of numbers using Mergesort
>msort []  = []
>msort [a] = [a]
>msort x
>  = merg (msort first_half) (msort second_half)
>    where
>    first_half    = [x!n | n <- [0..mid] ]
>    second_half = [x!n | n <- [mid1..max_index] ]
>    max_index   = #x - 1
>    mid             = max_index div 2
>    mid1            = mid + 1

>|| Merge two sorted lists into one sorted list.
>merg :: [*]->[*]->[*]  || built-in function
>merg [] y = y
>merg (a:x) [] = a:x
>merg (a:x) (b:y) = a:merg x (b:y), if a<=b
>                 = b:merg (a:x) y, otherwise

>|| Given a set as a list of numbers determine all subsets
>subset []     = [[]]
>subset (x:xs) = [[x] ++ y | y<- ys] ++ ys
>               where ys = subset xs
```

Miranda is rich with built-in functions and has many features not presented here. A more detailed study of Miranda is provided by David Turner[5,6], the author of Miranda. In addition, Bird and Wadler[1] have written an excellent textbook on functional programming. Examples in this textbook are written is a Miranda-like notation.

3. RESOURCES REQUIRED

There are several pure functional programming languages that are available. I choose Miranda because of prior experience with the language. Miranda runs on a Unix platform and is available to academic institutions for a modest fee from Research Software Ltd; inquires can be made to mira-request@ukc.ac.uk. The Miranda subsystem was placed on one of our existing Unix-based systems, with sufficient access over the network to easily accommodate all students in the class. Thus, no additional hardware was required.

The discrete mathematics class met three times a week for 50 minutes each. I devoted three 50 minute class periods (the equivalent of one week) to Miranda. Miranda was presented during the first lecture of each week for three consecutive weeks. This gave students time between lectures to use Miranda and try out some of the examples. Miranda was taught primarily by example. Each lecture was spent going over a two to three page handout of simple Miranda functions written by the instructor or examples provided by the Miranda subsystem. Suggested functions were given for students to try during the week. All handouts are working Miranda scripts and were also made available electronically for the students to copy to their account and work with. No other Miranda handouts were necessary since Miranda comes with a 92 page on-line manual. The students were not expected to read the manual, but use it for help when needed. In addition, a one-time five hour lab period was set aside to assist students who were not familiar with Unix. Students could come and go as their schedule permitted. Most students who came stayed 45 minutes to an hour. Instruction was given, along with handouts on simple Unix and vi commands. The Miranda subsystem was also explained and sample exercises were given. Instruction was primarily one on one.

Any instructor can teach an introduction to functional programming at the level presented in this course. Even an instructor totally unfamiliar with functional programming can easily learn enough in a week or so to present a simple introduction to the topic. To aid in this process, all Miranda programs, handouts, assignments and solutions that I have written are available upon request via electronic mail to any interested instructor.

4. PROGRAMMING ASSIGNMENTS

Programming Assignment I is given in Appendix I as a Miranda script. Students electronically copied the assignment to their account and wrote the necessary functions. Note statements beginning with > are executable statements. All other statements are comments. Comments are also indicated by ||. Students mailed their completed programs electronically to the instructor where it was subjected to a test script. The assignment was given after all of the Miranda lectures were completed. Since Miranda was started early in the course, plenty of time was available to do the assignment. Students were given six weeks to complete the assignment, approximately three times longer than they needed. This allowed extra time for any student who needed to catch up on Unix, vi, or was having difficulty in understanding the concepts of functional programming.

Hvorecky [4] poses an interesting mathematical problem which is the subject of Programming Assignment II. *Given a natural number N. Split N to its digits and compute the sum of their squares. Take the result and split it to its digits and compute the sum of their squares again. Repeat the procedure until you get 1 or 4.* It turns out that after some time the sequence degenerates to either 1, 1, 1, ... or 4, 16, 37, 58, 89, 145, 42, 20, 4, ... A variation to this problem is to find the smallest N generating the sequence containing L numbers before 1 or 4, (for $L <= 15$). A Miranda solution to this problem is given in Appendix II. Notice from the solution that this problem is more challenging, using more of the power of Miranda and some of its built-in functions. Students should compare this solution with a program they might write in C to do the same thing. This problem was not assigned, but was given to students who were interested in pursuing functional programming in more depth. The problem, however, is an interesting exercise for a discrete mathematics class to study.

5. RESULTS

A survey was taken among the students to get their views on introducing functional programming in the discrete mathematics course. The survey was given after the assignment was graded and returned. Responses were anonymous. Results are presented in Table I. The computer majors were divided into two groups in the survey. CS represents majors in computer science, a CSAB accredited degree program. CIS majors (Computer Information Systems) take the same computer courses as CS majors, but less mathematics and more business courses. The third group were mathematics majors. There were 13 CS students, 9 CIS students and 10 Math students responding to the survey.

The CIS and Math students for the most part were unfamiliar with Unix and vi and thought a little more instruction should have been given. In the future I intend to offer additional outside labs; perhaps two or three instead of one. Everyone thought enough time was given to do the assignment, and the assignment was at the right level of difficulty. Generally, all students found exposure to Miranda to be a fun experience, and helpful in understanding the concepts of functional programming. Finally, the general opinion by all students endorsed the idea of continuing to introduce functional programming in the discrete mathematics course. CS students found the exposure to functional programming to be an insight they had never experienced before and enthusiastically recommended an introduction to functional programming be a permanent part of the course.

6. CONCLUSIONS

I believe the concepts of functional programming are very important and should be a part of any computer science or mathematics degree program. At our university, mathematics students are not exposed at all to functional programming except in this course. All CS and CIS students are required to take a sophomore level language concepts course, but functional languages are not emphasized, and in some cases not presented. (I do not consider Lisp a suitable language for learning the concepts of functional programming). Combining an introduction to functional programming with discrete mathematics guarantees some exposure to all majors. These concepts are used in later courses. For example, I have occasionally used Miranda as a specification language to describe an algorithm or outline a proof in graduate and undergraduate data structures and algorithms courses. Functional programming can be a handy tool for mathematics majors in future courses as well. I believe that functional programming should be introduced in high school mathematics and computer courses along with the traditional procedural languages. Expos-

ing mathematics education students to functional programming is the primary way to accomplish this.

The introduction to functional programming in discrete mathematics is not intended to replace a semester course on the topic. However, many computer science departments do not offer a semester course on functional programming. Indeed, in many cases, the topic is totally absent from the discipline. Discrete mathematics provides an excellent opportunity to introduce the topic. Functional programming and discrete mathematics are a natural combination. Sets, relations, mathematical induction, counting methods, permutations and combinations, logic, and some aspects of graph theory are naturally expressed using a functional language. The students indicated this was an excellent experience. Introducing concepts of functional programming arose from an initial need to assign programs to students who have had little or no prior programming experience. Introducing functional programming concepts in discrete mathematics was very successful in practice and I am extremely pleased with the results. In fact, I believe this is an excellent idea even if all students are computer science majors and have significant programming experience. I believe exposure to the concepts of functional programming is an essential part of computer science and mathematics degree programs. One week of lectures and a small monetary investment is all that is required. I encourage professors to introduce functional programming in their discrete mathematics courses, or other appropriate courses if students have no other exposure to the topic.

REFERENCES

[1] Bird, R. and Wadler, P. *Introduction to Functional Programming*, Prentice Hall, 1988.

[2] Burton, F. Warren and Yang, Hsi-Kai. "Manipulating Multilinked Data Structures in a Pure Functional Language", *Software Practice and Experience*, vol. 20 (11), pp. 1167-1185, November, 1990.

[3] Hughes, J. "Why Functional Programming Matters", *The Computer Journal*, vol. 32, no. 2, pp. 98-107, 1989.

[4] Hvorecky, Jozef. "On a Connection Between Programming and Mathematics", *SIGCSE BULLETIN*, vol. 22, no. 4, pp. 53-54, December, 1990.

[5] Turner, David. "An Overview of Miranda", *SIGPLAN Notices*, pp. 158-166, December, 1986.

[6] Turner, David. "Miranda: a Non-strict Functional Language with Polymorphic Types", *Functional Programming Languages and Computer Architectures*, Springer-Verlag, Lecture Notes in Computer Science, vol. 201, 1985.

Appendix I
Programming Assignment I

>|| pgm1.m

DIRECTIONS: Complete the following program to implement SETS as ordered (sorted) lists with no repeated values. You are to write the following set operations as functions in Miranda. You MUST use the same function names that I have given below. Write Miranda functions called remove_dup, mem, add_elem, del_elem, subset, psubset, diff, sym_diff, union, and inter. EDIT THIS FILE AND IMPLEMENT THE FOLLOWING FUNCTIONS:

```
remove_dup :: set -> set
        This function is given an ordered list (perhaps with
        duplicates) and removes all duplicates
mem :: num -> set -> bool
        This function determines if an element is in a set
add_elem :: num -> set -> set
        This function adds the element to a given set
        (be sure to check for a duplicate)
del_elem :: num -> set -> set
        This function removes the element from a set
subset :: set -> set -> bool
        This function determines if the first set is a subset
        of the second
psubset :: set -> set -> bool
        This function determines if the first set is a proper
        subset of the second
diff :: set -> set -> set
        This function determines the set difference between
        two sets the first set - the second set.
sym_diff :: set -> set -> set
        This function determines the symmetric difference
        between two sets
union :: set -> set -> set
        This function determines the union of two sets
```

```
inter :: set -> set -> set
        This function determines the intersection of two sets

>set == [num]  || a set is a list of num
```

Appendix II
Miranda Solution to Programming Assignment II

```
>|| split_num.m
>|| Miranda solution to the problem posed in [4]

>digits :: num -> [num]
>digits 0 = []
>digits x = digits (x div 10) ++ [x mod 10]

>sqr :: num -> num
>sqr x = x*x

>|| ssd calculates the sum of the squares of all of the
>||    digits of an integer.
>ssd :: num -> num
>ssd x = sum(map sqr (digits x))

>|| iterate is a Miranda function.  iterate f x returns the
>|| infinite list [x, f x, f(f x), ... ]
>|| iterate f x = x:iterate f(f x)
>it_ssd :: num -> [num]
>it_ssd x = iterate ssd x

>not1_4 :: num -> bool
>not1_4 x = ~((x=4)\/(x=1))

>|| takewhile is a Miranda function.  takewhile applied to a
>|| predicate and a list, takes elements from the front of the
>|| list while the predicate is satisfied.
>|| takewhile f [] = []
>|| takewhile f (a:x) = a:takewhile f x, if f a
>||                    = [], otherwise
>result :: num -> [num]
>result x = takewhile not1_4 (it_ssd x)
>|| note result does not include the trailing 1 or 4 in the list

>len :: num -> num
>len x = # result x

>|| small a will determine the smallest natural number, n,
>|| such that the length of the generated sequence for n
>|| (ie., result n) is a.
>small :: num -> num
>small a = 1 + #(takewhile p (map len [1..]))
>       where p x = ~(x=a)

>table :: num -> [num]
>table limit = map small [1..limit]
>|| table 15 is [2,11,23,19,7,29,16,5,8,9,3,36,6,88,269]
>|| This is identical to the solution presented in [4]
```

Table I

Functional Language Survey
in Discrete Mathematics

Key: 1. Strongly Agree
2. Somewhat Agree
3. No Opinon
4. Somewhat Disagree
5. Strongly Disagree

Survey Questions	Student Population		
	CS	CIS	MATH
1. I was familiar with the Unix system before entering this class.	1.9	4.1	4.4
2. I was familiar with the vi editor before entering this class.	2.3	4.2	4.6
3. Enough classroom instruction was given in Miranda in order to do the assignment.	2.2	3.2	2.9
4. Enough time was give to do the Miranda assignment.	1.0	1.3	1.3
5. The Miranda Assignment was at the right level of difficulty.	1.5	2.0	2.5
6. I found programming in Miranda to be fun after I got into it.	1.2	2.1	2.3
7. The Miranda Assignment was very helpful in learning more about functional languages.	1.3	2.2	2.4
8. I recommend that Miranda be continued as part of the course the next time it is taught.	1.3	2.4	2.4