Graphical Representations and Infinite Virtual Worlds in a Logic and Functional Programming Course

J. E. Labra Gayo

Department of Computer Science, University of Oviedo
C/ Calvo Sotelo S/N, 3307,
Oviedo, Spain
labra@lsi.uniovi.es

Abstract. The assignment scheme of our *Logic and Functional Programming* course has adopted the generation of graphical representations, quadtrees and octrees to teach the different features of this kind of languages: higher order functions, lazy evaluation, polymorphism, logical variables, constraint satisfaction, etc. The use of standard XML vocabularies: SVG for graphics and X3D for virtual worlds enables to have numerous visualization tools. We consider that the incorporation of these exercises facilitates teaching and improves the motivation of students.

1 Introduction

The *Logic and Functional Programming* course is given as a free choice course of 60 hours to third year undergraduate students in Computer Science Engineering at the University of Oviedo. The rest of programming courses is based on imperative languages like Pascal, C, C++ and Java. In this way, our course is the first, and sometimes the only, contact that our students will have with declarative languages. Given its free choice nature, the course survival depends on the number of registered students. Apart from the intrinsic difficulty of programming courses, the choice of students is conditional on several aspects related with this kind of languages: poor programming environments, lack of graphical and intuitive debugging and tracing systems, lack of libraries or difficulties to link with existing libraries written in different languages, etc.

With the main goal of increasing students motivation and course scope we have developed the IDEFIX Project [15] since course 2000/2001, which is mainly focused on the development of an Internet based platform for teaching multiparadigm programming languages.

The course assessment is mainly based on programming assignments. These assignments follow an incremental presentation scheme using the taxonomy of cognitive goals [13]: firstly, we present a correct program that the students have to compile and execute. secondly, we ask them to make some modifications to the program in order that they learn by imitation a basic understanding of program construction. Finally, we ask them to create a new program by themselves.

In this paper, we describe the assignment scheme which is mainly based on the use XML vocabularies that include bidimensional graphics in SVG and virtual worlds in X3D. The papers begins with a brief overview of these XML vocabularies. In the next section we present the first basic exercise consisting on the development of graphical representations. In section 4 we study the quadtree recursive structure. In section 5 we present octrees and in section 6 we present the manipulation of infinite virtual worlds using lazy evaluation. In section 7 we describe the creation of polymorphic datatypes. Sección 8 describes the use of logic programming features (logical variables and non-determinism) and in the next section we solve the quadtree coloring problem using constraints.

Notation. Along the paper, we use Haskell and Prolog syntax. We suppose that the reader is familiar with these languages.

2 XML Vocabularies

XML [3] is becoming a standard for Internet information exchange and representation. One of the first assignments is to develop a small library for XML generation. This library will allow the students to have a common base for the rest of exercises and as it only requires string manipulation, it can serve as a simple familiarization with declarative languages.

The library must contain the following functions:

- empty e as = empty element e with attributes as
- gen e es = element e with subelements es
- genAs e as es = element e with attributes as and subelements es

One of the specific XML vocabularies that will be used is SVG (Scalable Vector Graphics) which is becoming the main standard for Internet graphics. Also, VRML (Virtual Reality Modelling Language) can be considered the main standard to represent Internet virtual worlds. However, as this standard precedes XML and does not follow its syntax, since year 2000 the Web3D consortium has developed X3D, which can be considered as a new VRML adapted to XML plus some new features.

3 Higher order functions: Graphical Representations

The second assignment that we ask the students is the development of graphical representations of functions. We present them the function plotF which stores in a SVG file the graphical representation of a function:

```
plotF :: (Double → Double) → String → IO ()
plotF f fn = writeFile fn (plot f)

plot f = gen "svg" (plotPs ps)
 where
  plotPs        = concat . map mkline
```

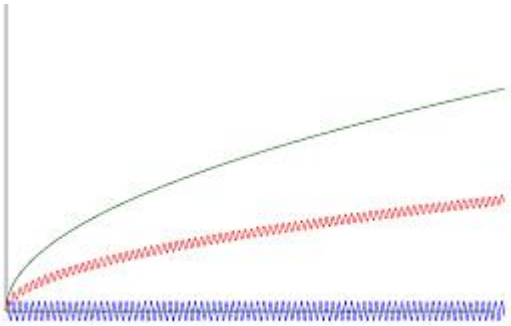

Fig. 1. Graphical representation of 2 functions and their mean

```
  mkline (x,x')= line (p x) (p x') c
  ps           = zip ls (tail ls)
  ls           = [0..sizeX]
  p x          = (x0+x, sizeY - f x)

line (x,y) (x',y') c = empty "line"
             [("x1",show x) ,("y1",show y),
              ("x2",show x'),("y2",show y')]

sizeX = 500; sizeY = 500; x0 = 10
```

Notice that plotF does Input/Output actions. Traditionally, most of the courses that teach purely declarative languages try to delay the presentation of the IO system. We consider that a more incremental presentation of this system avoids future misunderstandings. Since when it was posponed a lot of students found strange that introduction in a language that they were told to be pure. The source code of the plotF function also serves as an example that uses the recursive combinators zip, map, foldr, etc. that characterize Haskell.

In this assignment, the proposed exercises use the concept of higher order functions, which is a key concept of functional programming languages. For example, we ask the student to build a function plotMean that writes in a file the graphical representation of two functions and their mean. For example, in figure 1 we show the mean of (\x →10 ∗ **sin** x) and (\x →10 + **sqrt** x).

The code to represent the mean function is as simple as:

```
mean f g = plotF (\x → (f x + g x)/2)
```

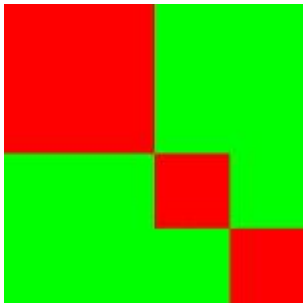

Fig. 2. Example of quadtree

4 Recursive Datatypes: Quadtrees

The next didactic unit is centered on the definition of recursive datatypes and their manipulation. The traditional assignments used the predefined datatype of lists and defined a new datatype fo binary trees. Usually, the students had serious difficulties to understand these recursive definitions and are usually not motivated by these exercises [9, 18]. In order to increase their motivation we will use the concept of quadtrees [17], which are recursive structures that can represent pictures and have numerous practical applications. In a quadtree, pictures are represented by a unique color or by the division of the picture in four quadrants which are quadtrees. The generalization of quadtrees to three dimensions are called octrees, as they divide the cube in eight parts. These structures are widely used in computer science because they allow to optimize the internal representation of scenes and tridimensional databases for geographical information systems. In Haskell, a quadtree can be represented using the following datatype:

```
data Color = RGB Int Int Int
data QT = B Color
        | D QT QT QT QT
```

An example of a quadtree could be:

```
exQT = D r g g (D r g g r)
  where r = B (RGB 255 0 0)
        g = B (RGB 0 255 0)
```

The above example is represented in figure 2.

Quadtree visualization is made using SVG files which are generated using the functions empty, gen and genAs implemented by students for XML file generation.

```
type Point = (Int, Int)
type Dim   = (Point, Int)
```

```
shqt :: QT → IO ()
shqt q = writeFile "qtree.svg"
           (gen "svg" (sh ((0,0),500) q))

sh :: Dim → QT → String
sh ((x,y),d) (B c) = rect (x,y) (x+d+1,y+d+1) c
sh ((x,y),d) (D ul ur dl dr) =
  let d2 = d `div` 2
  in if d <= 0 then ""
     else sh ((x, y), d2)             ul ++
          sh ((x + d2, y), d2)        ur ++
          sh ((x, y + d2), d2)        dl ++
          sh ((x + d2, y + d2), d2) dr

rect :: Point → Point → Point → String
rect (x,y) (x',y') (RGB r g b) =
 empty "rect"
     [("x",show x),("y",show y),
      ("height",show (abs (x - x'))),
      ("width", show (abs (y - y'))),
      ("fill", "rgb("++show r++","++
                       show g++","++
                       show b++",")]
```

5 Octrees and Virtual Worlds

An octree is a generalization of a quadtree two tridimensional representations: when each face of a cube is divided in four quadrants, eight cubes are obtained. The Haskell representation of octrees could be:

```
data OT = Empty
        | Cube Color
        | Sphere Color
        | D OT OT OT OT OT OT OT OT
```

The above representation declares that an octree can be empty, or it can be a cube, or a sphere or a division of eight octrees. An example of an octree could be:

```
exOT :: OT
exOT = D e s s e r e e g
 where e = Empty
       s = Sphere (RGB 0 0 255)
       r = Cube   (RGB 255 0 0)
       g = Cube   (RGB 0 255 0)
```

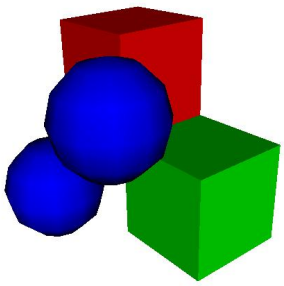

Fig. 3. Example of octree

The present the students the function wOT ::OT →FileName →**IO** () which takes two arguments, an octree and the name of a file, an stores a representation of that octree in a X3D file.

Figure 3 shows a screen capture of the exOT octree. ALthough in this paper, we present a printed version, the system generates a 3D virtual model that the students can navigate through.

6 Lazy Evaluation and Infinite Worlds

A nice feature of Haskell is lazy evaluation, which allows the programmer to declare and work with infinite data structures. In this case, we let the students to define infinite octrees like the following:

```
inf :: OT
inf = D inf e s e e e r inf
 where e = Empty
       s = Sphere (RGB 0 0 255)
       r = Cubo   (RGB 255 0 0)
```

Figure 4 presents the visualization of the previous octree.

It is also possible to define functions that manipulate infinite octrees. For example, the following function takes an octree as argument and generates a new octree repeating it.

```
repeat :: OT → OT
repeat x = D x x x x x x x
```

When applying the **repeat** function to the octree inf we obtain the octree that appears in figure 5.

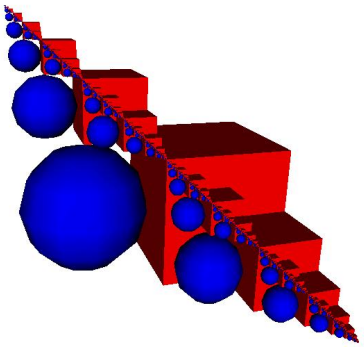

Fig. 4. Example of infinite octree

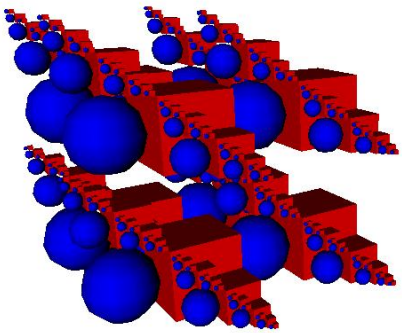

Fig. 5. Repetition of infinite octree

7 Polymorphism: Height Quadtrees

The Haskell type system allows the programmer to declare polymorphic datatypes like lists. Although traditional quadtrees contain color information in each quadtrant, it is possible to define a generalization of quadtrees that may contain values of different types. The new declaration could be:

```
data QT a = B a
          | D (QT a) (QT a) (QT a) (QT a)
```

Traditional quadtrees would be values of type QT Color. The above generalization enables the definition of other kinds of quadtrees, like those that contain height information (a value of type **Float**). Those quadtrees can be useful to represent terrain information. For example, the following quadtree:

```
qth :: QT Float
```

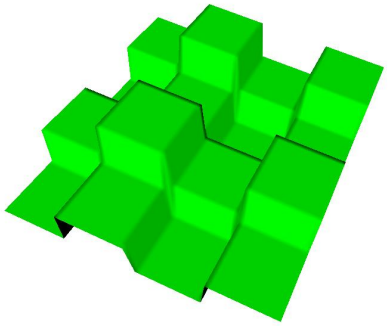

Fig. 6. Heights quadtree

```
qth = let x = D (B 10) (B 20) (B 0) (B 10)
      in D x x x x
```

is represented in figure 6.

Haskell promotes the use of generic functions like the predefined **foldr** function for lists. A similar function can also be defined for quadtrees.

```
foldQT :: (b → b → b → b → b) →
          (a → b) → QT a → b
foldQT f g (B x) = g x
foldQT f g (D a b c d) = f (foldQT f g a) (foldQT f b)
                           (foldQT f g c) (foldQT f d)
```

It is possible to define numerous functions using the foldQT function. For example, to calculate the list of values of a quadtree, we can define:

```
values :: QT a → [a]
values = foldQT (\a b c d → a ++ b ++ c ++ d)
                (\x → [x])
```

The depth of a quadtree can also be defined as:

```
depth :: QT a → Int
depth = foldQT (\a b c d → 1 + maximum [a,b,c,d])
               (\_ → 1)
```

The foldQT belongs to a set of functions that traverse and transform a recursive data structure into a value. These functions are also called catamorphisms and are one of the research topics of generic programming [2].

8 Non-determinism: Coloring Quadtrees

One of the difficulties of our course is the introduction of two paradigms, logic and functional in a short period of time. We are currently employing two different

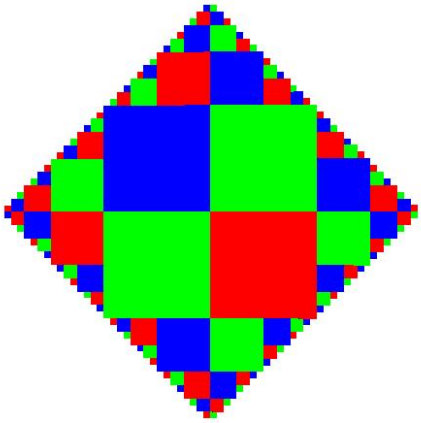

Fig. 7. Solution of the quadtree coloring problem

programming languages, Haskell and Prolog, although we would like to use a logic-functional language like Curry in the future. In order to facilitate the shift between paradigms and languages, we ask the students similar assignments in both parts of the course. So they can observe the main differences. In this way, the first assignments in the logic programming part also work in quadtrees and octrees. However, an important feature of logic programming is the use of logical variables and predicates that admit several solutions obtained by backtracking.

As an example, it is possible to define the predicate col(Xs,Q1,Q2) which is true when Q2 is the quadtree formed by filling the quadtree Q1 with colours from the list Xs.

```
col(Xs,b(_),b(X)):-elem(X,Xs).
col(Xs,d(A,B,C,D),d(E,F,G,H)):-
    col(Xs,A,E),col(Xs,B,F), col(Xs,C,G),col(Xs,D,H).
```

where **elem**(X,Xs) is a predefined predicate that is satisfied when X belongs to the list Xs. Notice that when we ask to fill a quadtree with several colours, we can obtain multiple solutions.

```
?-col([0,1],d(b(_),b(_),b(_),b(_)),V).
V = d(b(0),b(0),b(0),b(0)) ;
V = d(b(0),b(0),b(0),b(1)) ;
V = d(b(0),b(0),b(1),b(0)) ;
...
```

The traditional problem to color a map in such a way that no two adjacent regions have the same color can be posed using quadtrees. In figure 7 we present a possible solution to color a quadtree representing a rhombus.

A naive solution using logic programming consists of the generation of all the possible quadtrees and the corresponding test using the following predicate noColor.

```
noColor(b(_)).
noColor(d(A,B,C,D)):-
   noColor(A), noColor(B),noColor(C), noColor(D),
   right(A,Ar), left(B,Bl), diff(Ar,Bl),
   right(C,Cr), left(D,Dl), diff(Cr,Dl),
   down(A,Ad), up(C,Cu), diff(Ad,Cu),
   down(B,Bd), up(D,Du), diff(Bd,Du).

up(b(X),l(X)).
up(d(A,B,_,_),f(X,Y)):-up(A,X), up(B,Y).

down(b(X),l(X)).
down(d(_,_,C,D),f(X,Y)):-down(C,X), down(D,Y).

left(b(X),l(X)).
left(d(A,_,C,_),f(X,Y)):-left(A,X), left(C,Y).

right(b(X),l(X)).
right(d(_,B,_,D),f(X,Y)):-right(B,X), right(D,Y).

diff(l(X),l(Y)):-X\=Y.
diff(l(X),f(A,B)):-notElem(X,A), notElem(X,B).
diff(f(A,B),l(X)):-notElem(X,A), notElem(X,B).
diff(f(A,B),f(C,D)):-diff(A,C), diff(B,D).

notElem(X,l(Y)):-X\=Y.
notElem(X,f(A,B)):-notElem(X,A), notElem(X,B).
```

9 Constraints

Declarative languages offer an ideal framework for constraint programming. Although an in-depth presentation of the field is out of the course scope, we considered that a short introduction could be useful to the students, because most of them, will have no more contact with that field in other courses. Following with the quadtrees subject, we can define a solution of the map coloring problem using the logic programming extension with finite domains offered by some implementations. As an example, the following fragment solves the problem using the CLP(FD) syntax implemented in GNU Prolog [5].

```
colC(M,N,b(_),b(X)):-fd_domain(X,M,N).
colC(M,N,d(A,B,C,D),d(E,F,G,H)):-
    col(M,N,A,E),col(M,N,B,F),
```

```
col(M,N,C,G), col(M,N,D,H).
```

The new version of noColor is almost the same changing calls to predicate \= by calls to predicate #\=. However, the constraint programming implementation solves the problem for N=3 in 0.01sg while the naive solution does it in 74.5sg.

10 Related Work

The lack of popularity of declarative languages [19] has motivated the search for practical and attractive applications which highlight the main features of these languages. As an example, P. Hudak's textbook [10] presents an introduction to functional programming using multimedia based examples. That book uses specific libraries to generate and visualize the proposed exercises. Our approach is similar in its goal, but we have adopted standard XML vocabularies to generate the graphical elements. This approach offers several advantages: existence of numerous visualization tools, portability and independence of specific libraries. Furthermore, as a side effect, the students of our course could benefit from the use of these XML technologies in other fields of their future professional activity.

The declarative representations of quadtrees have already been studied in [4, 6, 8, 20]. Recently, C. Okasaki [16] take as a starting point the Haskell representation of a quadtree to define an efficient implementation of square mattrices using nested datatypes. The consistency of his representation is maintained thanks to Haskell type system.

In the imperative field there have been several works that emphasize the use of quadtrees as good examples for programming assignments [12, 11, 14]. However, most of these papers are centered on the image compression application of quadtrees. Several algorithms for quadtree coloring and their application to the schedule of parallel computations are described in [7].

11 Conclusions

In this paper, we propose the programming assignment scheme of our *logic and functional programming* course that intends to favour the graphical visualization of results and to present at the same time the main features of these languages.

Although we have not made a systematic research about the reaction of our students to this scheme, our first impressions are highly positive, with a significant decrease on the abandonment percentage compared to previous courses. Nevertheless, these kind of statements should rigorously be contrasted, checking, for example, that the students that take the new course really solve programming problems better than other students.

The generation of virtual worlds supposed an incentive for our IDEFIX project [15], among the future lines of research we are considering the development of virtual communities like ActiveWorlds [1] where the students can visit a cummunity, create their own worlds and chat with other students.

References

1. Activeworlds web page. http://www.activeworlds.com.
2. Roland Backhouse, Patrik Jansson, Johan Jeuring, and Lambert Meertens. Generic programming - an introduction. In S. Swierstra, P. Henriques, and Jose N. Oliveira, editors, *Advanced Functional Programming*, volume 1608 of *Lecture Notes in Computer Science*. Springer, 1999.
3. Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (1.0). http://www.w3.org/TR/REC-xml, Oct. 2000. 2nd. Edition.
4. F. W. Burton and J. G. Kollias. Functional programming with quadtrees. *IEEE Software*, 6(1):90–97, Jan 1989.
5. P. Codognet and D. Díaz. Compiling constraints in CLP(FD). *Journal of Logic Programming*, 27(3), Jun 1996.
6. S. Edelman and E. Shapiro. Quadtrees in concurrent prolog. In *Proc. International Conference on Parallel Processing*, pages 544–551, 1985.
7. D. Eppstein, M. W. Bern, and B. Hutchings. Algorithms for coloring quadtrees. *Algorithmica*, 32(1), Jan 2002.
8. J. D. Frens and D. S. Wise. Matrix inversion using quadtrees implemented in gofer. Technical Report 433, Computer Science Department, Indiana University, May 1995.
9. J. Good and P. Brna. Novice difficulties with recursion: Do graphical representations hold the solution? In *European Conference on Artificial Intelligence in Education*, Lisboa, Portugal, Oct 1996.
10. P. Hudak. *The Haskell School of Expression: Learning Functional Programming through multimedie*. Cambridge University Press, 2000.
11. R. Jiménez-Peris, S. Khuri, and M. Patiño-Martínez. Adding breadth to cs1 and cs2 courses through visual and interactive programming projects. *ACM SIGCSE Bulletin*, 31(1), Mar 1999.
12. J. B. Fenwick Jr., C. Norris, and J. Wilkes. Scientific experimentation via the mathing game. *SIGCSE Bulletin*, 34(1), Mar 2002. 33th SIGCSE Technical Symposium on Computer Science Education.
13. J. Kaasbøll. Exploring didactic models for programming. In *Norsk Informatikk-Konferanse*, Høgskolen I Agder, 1998.
14. S. Khuri and H. Hsu. Interactive packages for learning image compression algorithms. *ACM SIGCSE Bulletin*, 32(3), Sep 2000.
15. J. E. Labra, J. M. Moreno, A. M. Fernández, and Sagastegui H. A generic e-learning multiparadigm programming language system: IDEFIX project. In *ACM 34th SIGCSE Technical Symposium on Computer Science Education*, Reno, Nevada, USA, Feb 2003.
16. Chris Okasaki. From fast exponentiation to square matrices: An adventure in types. *ACM SIGPLAN Notices*, 34(9):28–35, 1999. International Conference on Functional Programming.
17. Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, Jun 1984.
18. J. Segal. Empirical studies of functional programming learners evaluating recursive functions. *Instructional Science*, (22):385–411, 1995.
19. P. Wadler. Why no one uses functional programming languages? *ACM SIGPLAN Notices*, 33(8):23–27, Aug 1998.
20. D. Wise. Matrix algorithms using quadtrees. Technical Report 357, Computer Science Department, Indiana University, Jun 1992.