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 an elective 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 have with declarative languages. Given its elective 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 libraries written in different languages, etc.

With the main goal of increasing students motivation and course scope we have developed the IDEFIX Project [17] since course 2000/2001, which is focused on the development of an Internet based platform for teaching multiparadigm programming languages.

The course devotes 45 hours to theoretical presentations and 15 hours to programming assignments. In this paper, we just describe the programming assignment scheme, which is based on the use of XML vocabularies that include bidimensional graphics in SVG and virtual worlds in X3D. These assignments follow an incremental presentation scheme using the taxonomy of cognitive goals [15]:
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 so they learn by imitation a basic understanding of program construction. Finally, we ask them to create a new program by themselves.

The paper begins with a brief overview of these XML vocabularies. In the next section we present the first basic exercise consisting of the development a XML generation library. In section 3 we present the next exercise which allows to generate graphical function representations using higher order functions. 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. Section 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 [5] 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 generates an empty element e with attributes as.
- gen e es generates element e with subelements es
- genAs e as es generates element e with attributes as and subelements es

The above functions could be implemented as:

```haskell
empty :: String → [(String, String)] → String
empty e as = "<" ++ e ++ putAtts as ++ "/">

gen :: String → String → String
gen e c = genAs e [] c

genAs :: String → [(String, String)] → String → String
genAs e as c = "<" ++ e ++ putAtts as ++ ">" ++
c ++
  "<" ++ e ++ "">

putAtts = concat . map f
  where f (v, n) = " " ++ v ++ "," ++ n ++ "\n"
```

2
One of the specific XML vocabularies that will be used is SVG [3] 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 [2], which can be considered as a new VRML adapted to XML plus some new features.

3 Higher order functions: Graphical Representations

The second assignment 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:

```haskell
plotF :: (Double -> Double) -> String -> IO ()
plotF f fn = writeFile fn (plot f)

plot f = gen "svg" (plotPs ps)
where
  plotPs = concat . map mkline
  mkline (x, x') = line (p x) (p x')
  ps = zip ls (tail ls)
  ls = [0..sizeX]
  p x = (x0+x, sizeY-f x)

line (x, y) (x', y') = 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 delayed it in previous years, a lot of students found strange its 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. In figure 1 we show the mean of $(x \rightarrow 10 \ast \sin x)$ and $(x \rightarrow 10 + \sqrt{x})$.

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

```haskell
mean f g = plotF (\x -> (f x + g x)/2)
```
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 of binary trees. Usually, the students had serious difficulties to understand these recursive definitions and are usually not motivated by these exercises [11, 20]. In order to increase their motivation we use the concept of quadtrees [19], 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:

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

An example of a quadtree could be:

```haskell
exQT = D r g g (D r g g r )
where r = B (RGB 2 5 5 0 0 )
g = B (RGB 0 2 5 5 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.

```haskell
type Point = (Int, Int)
type Dim = (Point, Int)
```
Fig. 2. Example of quadtree

wQt : QT → FileName → IO ()
wQt q fn = writeFile fn (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 → Color → 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 to three dimensions. 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:

\[
\text{exOT} \colon \text{OT} \\
\text{exOT} = \text{D e s s e r e e g} \\
\text{where} \quad e = \text{Empty} \\
\quad s = \text{Sphere} \ (\text{RGB} \ 0 \ 0 \ 255) \\
\quad r = \text{Cube} \ (\text{RGB} \ 255 \ 0 \ 0) \\
\quad g = \text{Cube} \ (\text{RGB} \ 0 \ 255 \ 0)
\]

We present the students the function \( wOT : \text{OT} \rightarrow \text{FileName} \rightarrow \text{IO} () \) which stores a representation of an octree in a X3D file.

Figure 3 shows a screen capture of the \( \text{exOT} \) 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. The students can define infinite octrees like the following:

\[
\text{inf} \colon \text{OT} \\
\text{inf} = \text{D i n f e s e e e r i n f} \\
\text{where} \quad e = \text{Empty} \\
\quad s = \text{Sphere} \ (\text{RGB} \ 0 \ 0 \ 255) \\
\quad r = \text{Cube} \ (\text{RGB} \ 255 \ 0 \ 0)
\]

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.
Fig. 4. Example of infinite octree

Fig. 5. Repetition of infinite octree

\texttt{repeat} \colon \texttt{OT} \to \texttt{OT}

\texttt{repeat} \ x = \texttt{D} \ x \ x \ x \ x \ x \ x \ x

When applying the \texttt{repeat} function to the octree \texttt{inf} we obtain the octree that appears in figure 5.

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 quadrant, it is possible to define a generalization of quadtrees that may contain values of different types. The new declaration could be:

\texttt{data QT} \ a = \texttt{B} \ a

\quad | \ D \ (QT} \ a) \ (QT} \ a) \ (QT} \ a) \ (QT} \ a)
The quadtrees presented in previous sections would be values of type QT Color. This 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:

\[
\text{qth} :: \text{QT Float} \\
\text{qth} = \text{let } x = \text{D (B 1 0) (B 2 0) (B 0) (B 1 0)} \\
\text{in D x x x x}
\]

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.

\[
\text{foldQT} :: (b \rightarrow b \rightarrow b \rightarrow b \rightarrow b) \rightarrow \\
(a \rightarrow b) \rightarrow \text{QT a} \rightarrow b
\]

\[
\text{foldQT f g (B x)} = g x
\]

\[
\text{foldQT f g (D a b c d)} = f (\text{foldQT f g a}) (\text{foldQT f g b}) \\
(\text{foldQT f g c}) (\text{foldQT f g d})
\]

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:

\[
\text{values} :: \text{QT a} \rightarrow \text{[a]}
\]

\[
\text{values} = \text{foldQT (\a b c d \rightarrow a ++ b ++ c ++ d)} \\
(\x \rightarrow \text{[x]})
\]

The depth of a quadtree can also be defined as:

\[
\text{depth} :: \text{QT a} \rightarrow \text{Int}
\]

\[
\text{depth} = \text{foldQT (\a b c d \rightarrow 1 + maximum [a, b, c, d])} \\
(\_ \rightarrow 1)
\]

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 [4].
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 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 with 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 \( \text{col}(Xs,Q1,Q2) \) which is true when \( Q2 \) is the quadtree formed by filling the quadtree \( Q1 \) with colours from the list \( Xs \).

\[
\text{col}(Xs, b(\_), b(X)) ;
\]

where \( \text{elem}(X,Xs) \) 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.

? \text{- 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 \( \text{noColor} \).

\[
\text{noColor}(b(\_)).
\]

\[
\text{up}(b(X), 1(X)).
\]

\[
\text{down}(b(X), 1(X)).
\]
9 Constraints

Declarative languages offer an ideal framework for constraint programming. Although an in-depth presentation of the field is out of our 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 [7].

\[
\text{colC} (M,N,b(X),b(Y)) :- \text{fd_domain} (X,M,N).
\]

\[
\text{colC} (M,N,d(A,_,C,_) ,d(E,F,G,H)) :-
\]

\[
\text{colC} (M,N,A,E) , \text{colC} (M,N,B,F) .
\]
The new version of noColor is almost the same changing calls to predicate \( \leq \) by calls to predicate \#\( \leq \). However, the constraint programming implementation solves the problem for \( N=3 \) in 0.01s while the naive solution does it in 74.5s.

### 10 Related Work

The lack of popularity of declarative languages [21] has motivated the search for practical and attractive applications which highlight the main features of these languages. As an example, P. Hudak's textbook [12] 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 [6, 8, 10, 22]. Recently, C. Okasaki [18] has taken as a starting point the Haskell representation of a quadtree to define an efficient implementation of square matrices using nested datatypes. The consistency of his representation is maintained thanks to the Haskell type system.

In the imperative field there have been several works that emphasize the use of quadtrees as good examples for programming assignments [14, 13, 16]. 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 [9].

### 11 Conclusions

In this paper, we propose a programming assignment scheme for 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 [17], among the future lines of research we are considering the development of virtual communities like ActiveWorlds [1] where the students can visit a community, create their own worlds and chat with other students.
References