An Editor for Helping Novices to Learn Standard ML

Jon Whittle\textsuperscript{1} and Alan Bundy\textsuperscript{1} and Helen Lowe\textsuperscript{2*}

\textsuperscript{1} Dept. of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, UK.

\textsuperscript{2} Dept. of Computer Studies, Napier University
Craiglockhart, 219 Colinton Road, Edinburgh
EH14 1DJ, UK.

Email: jonathw@dai.ed.ac.uk

Abstract. This paper describes a novel editor intended as an aid in the learning of the functional programming language Standard ML. A common technique used by novices is programming by analogy whereby students refer to similar programs that they have written before or have seen in the course literature and use these programs as a basis to write a new program. We present a novel editor for ML which supports programming by analogy by providing a collection of editing commands that transform old programs into new ones. Each command makes changes to an isolated part of the program. These changes are propagated to the rest of the program using analogical techniques. We observed a group of novice ML students to determine the most common programming errors in learning ML and restrict our editor such that it is impossible to commit these errors. In this way, students encounter fewer bugs and so their rate of learning increases. Our editor, CYNTHIA, has been implemented and is due to be tested on students of ML from September, 1997.

Keywords: Programming Language Learning, Learning Environments, Analogy

1 Introduction

Functional programming languages such as LISP, ML and Haskell are increasingly being used in academe and industry. Many universities now teach functional languages as a key part of their software engineering programme. However, the teaching of such languages presents problems. Functional languages involve abstract concepts such as recursion which are difficult to learn ([1]). Many experiments have been carried out that suggest that students overcome these difficulties by using analogy in the early stages of programming [16, 18]. Given a program to write, novices refer to similar programs they have written before or seen in the course literature. They then use the old program as a basis to construct the new one. We have conducted our own informal experiment with a group of 30

* The first author was supported by an EPSRC studentship. The second and third authors are supported by EPSRC grant GL/L/11724
novice ML students which involved observations of the students over the course of a semester and in-depth interviews with two of the students. This provided additional evidence of programming by analogy [19].

ML is a typed, functional language incorporating extensive use of pattern matching and recursion. We have implemented a program editor, CYNT/HIA, for Standard ML that supports programming by analogy. Programs are constructed in CYNT/HIA by transforming an existing program from an available library. The user is provided with a collection of editing commands. Each command makes an isolated change to the current program, such as adding an extra argument to a function definition. The effects of this change are then propagated automatically throughout the rest of the program. By applying a sequence of editing commands, previously constructed programs can be easily transformed into new ones. In addition, programs produced using CYNT/HIA are guaranteed free of certain kinds of bugs.

To illustrate the idea, consider the task of writing a function, **count**, to count the number of nodes in a binary tree, where the definition of the datatype **tree** is given in ML as:

```ml
datatype tree = leaf of int | node of tree * tree;
```

Suppose the user recognises that a function, **length**, to count the number of items in an integer list, is similar to the desired function. He can then use **length** as a starting point. Below we give the definition of **length** preceded by its type:

```ml
'a list -> int
fun length nil = 0
  | length (x::xs) = 1 + (length xs);
```

Note that `a list` is the polymorphic list type. We show how **length** could be edited into **count**. This example is taken from [20].

1. The user may indicate any occurrence of **length** and invoke the **RENAME** command to change **length** to **count**. CYNT/HIA then changes all other occurrences of **length** to **count**:

```ml
'a list -> int
fun count nil = 0
  | count (x::xs) = 1 + (count xs);
```

2. We want to count nodes in a tree so we need to change the type of the parameter. Suppose the user indicates **nil** and invokes **CHANGE TYPE** to change the type to **tree**.

---

3. **int** is the built-in datatype integers.
4. Throughout this document, I refer to the user by the pronoun 'he' although the user may be male or female.
5. :: is the ML list operator cons