FUNCTIONAL PEARLS
Monadic Parsing in Haskell

Graham Hutton
University of Nottingham

Erik Meijer
University of Utrecht

1 Introduction

This paper is a tutorial on defining recursive descent parsers in Haskell. In the spirit of *one-stop shopping*, the paper combines material from three areas into a single source. The three areas are functional parsers (Burge, 1975; Wadler, 1985; Hutton, 1992; Fokker, 1995), the use of monads to structure functional programs (Wadler, 1990; Wadler, 1992a; Wadler, 1992b), and the use of special syntax for monadic programs in Haskell (Jones, 1995; Peterson *et al.* , 1996). More specifically, the paper shows how to define monadic parsers using `do` notation in Haskell.

Of course, recursive descent parsers defined by hand lack the efficiency of bottom-up parsers generated by machine (Aho *et al.*, 1986; Mogensen, 1993; Gill & Marlow, 1995). However, for many research applications, a simple recursive descent parser is perfectly sufficient. Moreover, while parser generators typically offer a fixed set of combinators for describing grammars, the method described here is completely extensible: parsers are first-class values, and we have the full power of Haskell available to define new combinators for special applications. The method is also an excellent illustration of the elegance of functional programming.

The paper is targeted at the level of a good undergraduate student who is familiar with Haskell, and has completed a grammars and parsing course. Some knowledge of functional parsers would be useful, but no experience with monads is assumed. A Haskell library derived from the paper is available on the web from:

```
http://www.cs.nott.ac.uk/Department/Staff/gmh/bib.html#pearl
```

2 A type for parsers

We begin by defining a type for parsers:

```
newtype Parser a = Parser (String -> [(a,String)])
```

That is, a parser is a function that takes a string of characters as its argument, and returns a list of results. The convention is that the empty list of results denotes failure of a parser, and that non-empty lists denote success. In the case of success, each result is a pair whose first component is a value of type `a` produced by parsing