Stream Fusion on Haskell Unicode Strings

Thomas Harper[1]

Oxford University Computing Laboratory
Oxford, United Kingdom

Abstract. Prior papers have presented a fusion framework called stream fusion for removing intermediate data structures from both lists and arrays in Haskell. Stream fusion is unique in using an explicit datatype to accomplish fusion. We demonstrate how this can be exploited in the creation of a new Haskell string representation *Text*, which achieves better performance and data density than *String*. *Text* uses streams not only to accomplish fusion, but also as a way to abstract away from various underlying representations. This allows the same set of combinators to manipulate Unicode text that is stored in a variety of ways.

1 Introduction

Lists are the primary workhorse data structure of functional programming. In the programming language Haskell[1], strings are represented using the built-in list type. This allows programmers to use standard polymorphic list combinators to build complex string manipulation functions in the same way that they would manipulate other lists. Haskell programmers often take advantage of this by composing list functions to form "pipelines" for transforming lists (and strings). For example:

$$return \cdot words \cdot map\ toUpper \cdot filter\ isAlpha =\!\!\ll readFile\ f$$

This program reads in a file, filters out the non-alphabetic characters, converts the remaining characters to uppercase, and then tokenises them. It exemplifies Haskell's ability to create concise yet powerful programs through the composition of modular functions. It is also, however, extremely inefficient. Haskell's *String*s are much larger than, for example, their C counterparts. To address these inefficiencies, there is an alternative to *String* in the Haskell core libraries called *ByteString*. *ByteString* addresses *String*'s inefficiencies and achieves greater performance, but it does so at the cost of support for non-ASCII characters. We are introducing a new data type, *Text*, to fill in the gap left between these two approaches. *Text* addresses the performance issues associated with *String* while maintaining Unicode support.

The *Text* type is an array-based string representation that is faster and more compact than *String*. Its API is based on Haskell's list library, which means that it can be used as a drop-in replacement for *String*. Figure 1 shows the speed-up achieved by using *Text*s to run the example program from above. The main contribution of this paper is a faster, more compact string representation