Polytypic Data Compression

Johan Jeuring

1 Introduction

A large part of the data sent over the internet is (semi-) structured, examples are databases, html files, and javaScript programs. And a recent future-scenario about the internet predicted that future computer users will not buy their programs (editors, compilers, spreadsheets, etc.) anymore, but instead fetch their programs from the internet. Thus a considerable amount of internet traffic consists of files that possess structure, and it will pay to compress these structured files. Structure-specific compression methods give much better compression results than conventional compression methods such as the Unix compress utility [11, 10, 1]. For example, Unix compress typically requires four bits per byte of Pascal program code, whereas Cameron [2] reports compression results of one bit per byte Pascal program code.

The basic idea of the structure-specific compression methods is simple: parse the input file into a structured value, separate structure from contents, flatten the structure into a string, judiciously choosing representations for constructors, and compress the resulting string and the contents with a conventional compression method. For example, suppose we have the following datatype for trees:

```
data Tree a  =  Leaf a | Bin (Tree a) (Tree a)
```

and a file contains the following tree `t`

```
Bin (Bin (Leaf "bar") (Leaf "car")) (Bin (Leaf "far") (Leaf "war"))
```

Separating structure from contents gives:

```
Bin (Bin (Leaf ()) (Leaf ()) (Bin (Leaf ()) (Leaf ()))
```

and a list containing the four words `"bar"`, `"car"`, `"far"` and `"war"`. Assigning `0` to `Bin` and `1` to `Leaf`, the above structure can be represented by

```
0011011
```

This list of zeroes and ones equals 108 when read as a binary number, and hence this list can be represented by the 108'th ASCII character 'l'. So the tree `t` can be represented by the list of words `[l,bar,car,far,war]`. The tree `t` is stored in 68 bytes, and its compressed counterpart requires 19 bytes. This list can be further compressed using a conventional compression method.

Most authors of papers on program code compression [3, 6, 9, 2, 5, 8, 4, 7] observe that this method works for arbitrary structured objects, but all results are based on compressing Pascal programs. To compress JavaScript programs we will have to write a new compression program. It is desirable to have a polytypic data compression program.

2 The problem

Your assignment is to write the polytypic programs for data compression and decompression, using the polytypic programming system Polyp. The polytypic data compression program should:

- parse the input string into a value of a datatype;
- separate structure from contents;
- flatten the structure to a string using as little space as possible;
- and compress the resulting string and the contents of the value using a conventional compression method, such as Unix compress.

The polytypic decompression program should be the inverse of the data compression program.

An important part of the polytypic data compression program is a polytypic program for parsing. A polytypic function for parsing takes a function that prints values of type `Mu f a` and returns a function that parses a string into a value of type `Mu f a`.

```
parse :: (Mu f a -> Print) -> String -> Mu f a
```

where `Print` is some suitable datatype that can be used to pretty print values.

References

[1] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, 1990.

[2] Robert D. Cameron. Source encoding using syntactic information source models. *IEEE Transactions on Information Theory*, 34(4):843–850, 1988.

[3] J.F. Contla. Compact coding of syntactically correct source programs. *Software–Practice and Experience*, 15(7):625–636, 1985.

[4] Rod M. Davies and Ian H. Witten. Compressing computer programs. Technical Report 93/7, Department of ComputerScience, University of Waikato, Hamilton, New Zealand, 1993.

[5] J. Katajainen and E. Mäkinen. Tree compression and optimization with applications. *International Journal of Foundations of Computer Science*, 1(4):425–447, 1990.

[6] J. Katajainen, M. Penttonen, and J. Teuhola. Syntax-directed compression of program files. *Software–Practice and Experience*, 16(3):269–276, 1986.

[7] Craig G. Nevill-Manning and Witten Ian H. Compressing semi-structured text using hierarchical phrase identification. In *Proceedings Data Compression Conference, DCC '96*, 1996.

[8] Gary Promhouse and Mike Bennett. Semantic data compression. In J. Storer and J.H. Reif, editors, *Proceedings Data Compression Conference, DCC '91*, pages 323–331, 1991.

[9] R.G. Stone. On the choice of grammar and parser for the compact analytical encoding of programs. *The Computer Journal*, 29(4):307–314, 1986.

[10] T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.

[11] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.