Compact Linear Types

John Skaller

August 4, 2019

1 Introduction

Let us observe that the number of pennies in 3 pounds, 10 shillings, and 8 pennies, is given by:

$$8 + 10 \times 20 + 3 \times 20 \times 12$$

because there are 12 pennies in a shilling, and 20 shillings in a pound. Generalising this idea we have:

$$a = x_0 + \sum_{i=1}^{n-1} (x_i \prod_{j=0}^{i-1} c_j)$$

which is a general finite positional number system, with j ranging from 0 to $n-2$ representing an n position system, and c_j representing the number of digits in the j'th position.

Decoding is given by

$$x_i = (a \operatorname{rmd} \prod_{j=i}^{n-2} c_j) \operatorname{div} \prod_{j=0}^{i-1} c_j$$

2 Application to types

Let the symbol decimal integer n represent the sum of n units, where the unit type is the type of an empty tuple. Call the type void, the sum of no units, and observe

```
typedef void = 0;
typedef unit = 1;
typedef bool = 2;
```

Call any such type a unitsum. Now observe that the positional number system described in section 1 may be used to encode and decode values of a finite

product of unit sums by adding a limit on the number of digits in the higest term.

The formula for the x_i are then projections of the product which is formed by the formula for the a. The set of integers calculated by the constructor then has size

$$N = \prod_{j=0}^{n-1} c_j$$

and the set is compact and consists of the values 0 through $N-1$ and is therefore also linear.

The system is also familiar as the decomposition of cyclic groups and exhibits the isomorphism

$$Z_N \cong \mathbb{Z}_{j_0} \times \mathbb{Z}_{j_1} \dots \mathbb{Z}_{j_{n-1}}$$

3 Application to arrays

A fundamental problem in computing is that a loop through an array of arrays requires two loops:

```
var a = ((1,2,3),(4,5,6));
var sum = 0;
for i in 0..<2
  for j in 0..<3 perform
     sum += (a.i).j;
println$ sum;
```

and for an array of rank k we need k loops. There is a two step solution to this problem. We desire to make the transformation of types

$$\texttt{int}^{3^2} \longrightarrow \texttt{int}^{3\times 2}$$

This has the effect of changing the data functor to a single array, indexed by a tuple:

```
var a = ((1,2,3),(4,5,6));
var b = a :>> (int^(2 * 3));
var sum = 0;
for i in 0..<2
  for j in 0..<3 perform
     sum += b.(i:>>2,j:>>3);
println$ sum;
```

Although we still need two indices and two loops, we have formed a new data functor by composition of two array functors.

A second isomorphism solves our problem:

$$\texttt{int}^{2\times 3} \longrightarrow \texttt{int}^{6}$$

with

```
var a = ((1,2,3),(4,5,6));
var b = a :>> int^(2*3);
var c = b :>> int^6;
var sum = 0;
for i in 0..<6 perform
  sum += c.i;
println$ sum;
```

performing a compact linear encoding.

4 Extension to Arrays

If the base type of an array is itself compact linear, then the whole array is also compact linear. For example

```
var x : bool ^ 4 = true, false, true, true;
println$ x :>> int;
```

Bit arrays drop out of compact linear types as a special case.

5 Extension to sums

We consider first the case of two arrays of the same base type. If the first has length n and the second length m then an index of type $n+m$ is interpreted as follows: first chose which array to index, then use the argument of the selected case as the index. By concatenating the arrays, we see the index is isomorphic to an array of length $n+m$.

This suggests a sum of compact linear types is also compact linear.

The size of a compact linear type is given inductively by the rule:

- the size of a unitsum n is n
- the size of a product of compact linear types is the product of the sizes

- the size of an exponential of compact linear types is the exponential of the sizes
- the size of a sum of compact linear types is the sum of the sizes

The encoding formula is:

$$a = y_i \sum_{j=0}^{i-1} z_j$$

where z_j is the size of the j'th case. The y_i are readily identified as the characteristic injections of the sum type.

The decoding formula is in two parts: we must first calculate the correct case index, it is the unique j such that

$$z_j > a \geq z_{j-1}$$

Then the argument is given by

$$y_j = a - \sum_{i=0}^{j-1} z_i$$