BURT ROSENBERG UNIVERSITY OF MIAMI FALL 1992

Simulation of Infinite Memory

In KMA the following program rewriting function *short* is desired. Given a program P_e of arity j using k variables, $k \geq j$, derive a program $P_{short(e)}$ such that $P_{short(e)} = P_e$ and $P_{short(e)}$ uses only $j + r$ variables, where r is a "constant" depending on j but not k.

The approach in KMA used pairing functions to fold a finite amount of memory into a fixed amount of memory. It is just as easy to devise a scheme which folds an *infinite* amount of memory into a fixed amount provided that only a finite number of variables are ever simultaneously non-zero. For a small conceptual effort to clarify what an infinite memory could mean, the while-program mechanisms are simplified.

A *pairing function* is a computable bijection $\tau : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$. Its inverse is a pair of *projection functions* π_1 and π_2 such that,

$$i = \tau(\pi_1(i), \pi_2(i))$$

for all $i \in \mathbf{N}$. For our construction to succeed, we require $\tau(0, 0) = 0$. The pairing function of KMA is an example of such a pairing function:

$$\tau(i, j) = \frac{(i + j)(i + j + 1)}{2} + i. \tag{1}$$

We will define the symbol $\mathbf{N}^\omega$ as the *direct sum* of an infinite number of copies of the naturals. An element of $\mathbf{N}^\omega$ is an infinite-dimensional vector of naturals, all but finitely many entries being zero. The vector e_i which is zero in all but the i-th coordinate where it is one is an example of an element from $\mathbf{N}^\omega$. So is the vector which is 1 for all odd numbers less than a billion and zero elsewhere.

Theorem 1 *There exists a computable bijection $\tau^* : \mathbf{N}^\omega \to \mathbf{N}$ with a family of projection functions $\pi_i^* : \mathbf{N} \to \mathbf{N}$, $i = 1, 2, \ldots$.*

PROOF: Let τ be a pairing function 1. Then τ^* is,

$$\tau^*(x_1, x_2, \ldots) = \tau(x_1, \tau(x_2, \ldots \tau(x_i, \ldots) \ldots)).$$

Written this way, the function involves infinite recursion and is not computable, in fact, the definition itself is suspect. However, since $\tau(0,0)=0$, at a certain point it is inconsequential to continue the recursion. Define, therefore,

$$\tau^*(x) = \tau^i(x) = \tau(x_1, \tau(x_2, \ldots, \tau(x_i, 0))\ldots)$$

where i is any integer such that x_j is zero for all $j > i$. If i and i' are two integers such that $x_j = 0$ for all $j > \min(i, i')$, then $\tau^i(x) = \tau^{i'}(x)$.

The inverse is defined as,

$$\pi_i^*(x) = \pi_1({\pi_2}^{(i-1)}(x)).$$

That is, apply the projection π_2 iteratively $i-1$ times to the argument, and then project by π_1. Since τ^* is τ^i for some i, the proof that this π^* is the inverse reduces to what has already been proved by KMA for the case of pairing functions $\mathbf{N}^k \to \mathbf{N}$.

The function τ^* is injective. Suppose $\tau^*(x) = \tau^*(y)$. Then for some i and j, $\tau^*(x) = \tau^i(x)$ and $\tau^*(y) = \tau^j(y)$. Letting $k = \max(i,j)$ then,

$$\tau^k(x) = \tau^i(x) = \tau^*(x) = \tau^*(y) = \tau^j(y) = \tau^k(y).$$

In KMA it is shown that τ^k is a bijection, so $x = y$.

The surjectivity a consequence of π^* being total.

□

To encode the variable $X_1, \ldots, X_k$ of a certain k-variable while-program into a single variable, say M, we consider the finite set of variables which the program actually uses as a subset of the infinite set of variables:

$$\ldots, X_{-3}, X_{-2}, X_{-1}, X_0, X_1, X_2, X_3, \ldots.$$

The variables of index zero or less are always zero, as are the variables of indices larger than k.

For a j-ary program P_e begin by encoding the vector,

$$(X_1, X_2, \ldots, X_j, 0, 0, \ldots)$$

into a single number M. Since j is known, it is possible to select the proper finite version of τ^*. The variables,

$$(X_0, X_{-1}, \ldots),$$

being all zero are encoded into the number $M' = 0$. We have the following macro, which takes the "top" number off of M and places it on M'.

```
nextX(M,M') =
   begin
     M' := tau( pi1(M), M' ) ;
     M  := pi2(M)
   end
```

Extraction of the value of a certain X_i from M is then possible by shifting an appropriate number of times and then projecting,

```
Xi := get(M,i) =
   begin
     M'':= M ;
     M' := 0 ;
     i := prev(i) ;
     while i<>0 do nextX(M'',M') ;
     Xi := pi1(M'')
   end
```

Changing the value of a certain X_i in memory M proceeds as follows.

```
set(M,i,Xi) =
     begin
       M' := 0 ;
       k := prev(i) ;
       while k<>0 do nextX(M,M') ;
       M := pi2(M) ;
       M := tau(Xi,M) ;
       k := pred(i) ;
       while k<>0 do nextX(M',M)
     end
```

At the close of the program, we need to correctly place a value in X_1,

```
X1 := pi1(M)
```