A Minimal Stack Machine

Stack Discipline

Recall that this means that an n-ary function f **computed** on a stack s expects its arguments on the stack and on **return** replaces them by its result yielding a new stack t:

$$s = a_1, \ldots, a_n, u \Rightarrow t = f(a_1, \ldots, a_n), u$$

Here the **stack** s is not **coded** on a **Turing tape** but it is a **list** of numbers, where for $1 \leq i \leq n$ we have $a_i = (s)_{i-1}$ for a two place **list indexing function** satisfying:

$$(a, s)_0 = a \qquad (a, s)_{i+1} = (s)_i$$

The element $(s)_0 = H(s)$ is on the **top** of the stack. Note that we have

$$(s)_n = H\,\overbrace{T \cdots T}^{n}(s)$$

Minimal Turing Complete Stack Machine

expects a **program**, which is a **list** of **instructions** which modify a stack of natural numbers. Such a machine is **Turing complete** iff *any numerical function computable on a Turing machine can be computed on the stack machine*
Note that by the **Thesis of Church** we **cannot** compute more functions than those computed by **Turing machines**.
Instructions:
Push: pushes 0 the stack: $s \Rightarrow 0, s$
Pop: removes the top from the stack: $a, s \Rightarrow s$
Incr$_i$: increases the i-th element by 1:
$s = \overbrace{\cdots}^{i}, a, s_1 \Rightarrow t = \overbrace{\cdots}^{i}, a + 1, s_1$
Decr$_i(p)$: if $(s)_i > 0$ decreases the i-th element by 1 and performs p; **Decr**$_i(p)$. It does nothing otherwise.

Bootstrapping of the minimal machine

We can do $(s)_i := 0$, i.e. **clear the i-th element on s by**

$$\mathbf{Decr}_i(\mathbf{Push}; \mathbf{Pop})$$

We can do $(s)_i := (s)_j$, i.e. non-destructively assign, the j-th element of s to j-th one by:

$$(s)_i := 0; \mathbf{Push}; \mathbf{Decr}_{j+1}(\mathbf{Incr}_0; \mathbf{Incr}_{i+1}); \\ \mathbf{Decr}_0(\mathbf{Incr}_{j+1}); \mathbf{Pop}$$

We can do **If** $(s)_i > 0$ **then** p **else** q by

$$\mathbf{Push}; \mathbf{Incr}_0; \mathbf{Push}; (s)_0 := (s)_{i+2}; \\ \mathbf{Decr}_0((s)_0 := 0; p^{+2}; (s)_1 := 0); \\ \mathbf{Decr}_1(q^{+2}); \mathbf{Pop}; \mathbf{Pop}$$

where p^{+2} is like p but with every **stack index** increased by **two**. For instance;

$$p = \mathbf{Incr}_3; (s)_6{:=}(s)_8 \Rightarrow p^{+2} = \mathbf{Incr}_5; (s)_8{:=}(s)_{10}$$

Coding of Terms and Their Denotations

Turing complete terms

Instead of using the minimal stack machine we will study a stack machine for the computation of **functional terms** which are the minimal set of **expressions** formed from: the variable V and decimal numerals n by Incr(a), Decr(a), Head(a), Tail(a), Pair(a, b), If(a, b, c), Apply(a, b), and R(a) where a, b, and c are previously constructed functional terms.

We can show that every **Turing computable** function f can be computed by **evaluating** a **functional term** for f.

Instead of evaluating n-ary functions f we will work with their **unary contractions** $\langle f\rangle(x)$ such that

$$\langle f\rangle(x) = f((x)_0, \ldots, (x)_{n-1})$$

Thus $f(x_1, \ldots, x_n) = \langle f\rangle(x_1, \ldots, x_n, 0)$

Explanatation of functional terms

Functional terms have two **variables** V standing for the single argument v of $\langle f\rangle(v)=r$ and R which stands for the **body**, i.e. the functional term, r of $\langle f\rangle$. Thus $R(a)$ occurring within a stands for the **recursive call** $\langle f\rangle(a)$.

The functional term Add_t for the function $\langle \text{Add}\rangle(x)=(x)_0+(x)_1$ is

$$\begin{array}{l}\text{If}(\text{Var}(0),\\ \ \text{Incr}R\text{Pair}(\text{DecrVar}(0),\text{Pair}(\text{Var}(1),0)),\\ \ \text{Var}(1))\end{array}$$

where Var(i) **abbreviates** the term

$$\text{Head}\ \overbrace{\text{Tail}\cdots\text{Tail}}^{i}(V)$$

accessing the $i+1$-st variable of f.

Denotation of functional terms

A functional term a **denotes** (has as its value, evaluates to) a number in an **assignment** of a number v to the variable V and a functional term r to the variable R.

We thus have a three-place **denotation function** $[a]_r^v$ yielding the value of a. We have

$$[\text{Add\_t}]_{\text{Add\_t}}^{3,2,0} = 5$$

The term Apply(a, b) stands for the **application** of the function with the functional term a to the argument denoted by the term b:

$$[\text{Apply}(a, b)]_r^v = [a]_a^{[b]_r^v}$$

Note: the denotation function is only a **partial function** because it has **no value** when the recursion does not terminate.