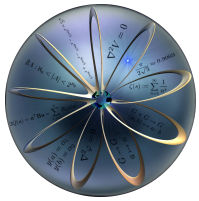

planetmath.org

Math for the people, by the people.

combining URMs

Canonical name	CombiningURMs
Date of creation	2013-03-22 19:04:07
Last modified on	2013-03-22 19:04:07
Owner	CWoo (3771)
Last modified by	CWoo (3771)
Numerical id	16
Author	CWoo (3771)
Entry type	Application
Classification	msc 68Q05
Classification	msc 03D10

Combining URMs

The basic idea of combining existing URMs is to produce URMs that can perform more complicated computations. Suppose we are given two URMs $M = I_1, \ldots, I_m$ and $N = I'_1, \ldots, I'_n$, define M, N to be the URM

$$M, N := I_1, I_2, \ldots, I_{m+n}$$

where $I_{m+i} = I'_i$, where $i \in \{1, \ldots, n\}$. In other words, the instructions of N are re-indexed and appended to the last instruction of M.

In theory, we would like M, N to first go through the instructions of M, and if the computations of M terminate, go through the computations of N. In order to achieve this goal, modifications to M and N need to be made so computations of M are kept apart from the computations of N:

1. If $I_k = J(i, j, q)$ is in N, then change I_k to $J(i, j, q+m)$. This is clearly necessary as the indices of the instructions of N have been shifted by m. Let the modified N be denoted by $N(m)$.

2. If $I_k = J(i, j, p)$ is in M, and $p > m$, then change I_k to $J(i, j, m+1)$. The purpose of this change is to ensure that computations of M do not accidentally jump to an instruction of N. Furthermore, when M halts, N starts. Call this modified machine M'. It is easy to see that every M can be modified to M'.

When M computes a function $f : \mathbb{N}^n \to \mathbb{N}$ and N computes a function $g : \mathbb{N} \to \mathbb{N}$, we would further want the combined machine to compute $g \circ f$. In essence, if r is an input, and if $f(r)$ is computed by M, we want $f(r)$ to be the input of the machine N, so that $g(f(r))$ may be computed. To get the desired result, one more modification should be made:

3. Suppose M computes $f : \mathbb{N}^n \to \mathbb{N}$. Define $k = \max(n, \rho(M))$. Then M^* is defined as the URM $M', Z[k]$, where M' is defined in 2. above, and $Z[k]$ is the machine with instructions $Z(2), Z(3), \cdots, Z(k)$.

In other words, whenever $f(r)$ is computed by M, M^* resets the contents of all registers potentially used by M to 0, except the first one, which contains the result $f(r)$. As a result, the input for N has 0 in all but possibly the first register.

With the three modifications above, let us define $M \circ N$ to be the machine $M^*, N(|M|)$. It is easy to see that $\circ$ is an associative operation. We can summarize our discussion above into the following:

Proposition 1. *Let M, N be URMs that compute $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ respectively, then $M \circ N$ computes $g \circ f$. In other words, if f and g are URM computable, so is $g \circ f$.*

For example, if M computes $f(r, s) := r + s$ for any $r, s \in \mathbb{N}$, and N computes $g(r) := r \dot{-} 1$ for any $r \in \mathbb{N}$, then $M \circ N$ computes $(g \circ f)(r, s) = (r + s) \dot{-} 1$. In addition, with input r, the URM

$$\underbrace{N \circ N \circ \cdots \circ N}_{s}$$

computes $g^s(r) := r \dot{-} s$. Note that g^s is unary, and therefore not the same as the binary operation $h(r, s) := r \dot{-} s$. To construct a URM computing h using the URM N, the method described above does not work, and a more general construction is needed.

URM as a Subroutine

Another way of generating URMs from existing ones is by inserting the existing URMs, so called subroutines, as part of a larger URM. Given a URM $M = I_1, \ldots, I_m$, a *subroutine* of M is defined as a block of consecutive instructions in M. Based on this definition, we see that combining machines can be seen as a special case: the URMs M and N are subroutines of the URM M, N. In fact, each instruction in a URM can be considered as a subroutine.

As in the case with combining two URMs, when inserting a URM as a subroutine in a larger URM, one would like to keep the computations done within the subroutine isolated from the remaining computations. Since only one tape is allowed, one can allocate a portion of the tape reserved only for subroutine computations, another portion for storage, while the rest for computations by the main program. This can be done as follows:

4. For any URM N, define $\rho^*(N) := \max(\rho(N), n(N))$, where $\rho(N)$ is the number of registers used for computations by N, and $n(N)$ is the arity of the function computed by N.

5. Suppose $N_1, \ldots, N_k$ are to be inserted as subroutines in a program M, define
$$n := \max\{\rho^*(N_i) \mid i = 1, \ldots, k\}.$$
Allocate the first n registers for computations by the subroutines N_i.

6. Allocate the next n^* registers for storage, where $n^* = n_1 + \cdots + n_k$.

7. For any URM N, and any positive integers $R_1, \ldots, R_s, R_t$, define the following URM:

   $$N[R_1, \ldots, R_s; R_t] := T(R_1, 1), T(R_2, 2), \ldots, T(R_s, s), N, T(1, R_t), Z(1), \ldots, Z(k),$$

   where $k = \rho^*(N)$.

8. Computations of by the remaining instructions of M will take place in registers $n + n^* + 1$ or beyond.

9. For each N_i to be inserted, insert $N[R_1, \ldots, R_s; R_t]$ instead, where each R_j is determined by the computations done by instructions just prior to the point of insertion.

The above only serves as a guideline, not a definite rule to follow. The following example serves as an illustration: let $N_1, \ldots, N_k$ be URMs that compute m-ary functions $f_1, \ldots, f_k$, and let M be a URM that computes a k-ary function g. Define an m-ary function $h : \mathbb{N}^m \to \mathbb{N}$ as the composition of the f's and g:

$$h(r_1, \ldots, r_m) := g(f_1(r_1, \ldots, r_m), \ldots, f_k(r_1, \ldots, r_m)),$$

provided, of course, that the right hand side is defined. We show that h is URM-computable by finding a suitable URM P that computes h:

1. Given input r occupying the first n registers, first copy contents of the input to the storage area by the following instructions:

   $$P_1 := T(1, n+1), T(2, n+2), \ldots, T(m, n+m)$$

2. Insert programs

   $$P_2 := N_1[n+1, \ldots, n+m; n+m+1], \ldots, N_k[n+1, \ldots, n+m; n+m+k]$$

   This has the effect of computing $f_1(r_1, \ldots, r_m), \ldots, f_k(r_1, \ldots, r_m)$, and putting the results in registers $n+m+1, n+m+2, \ldots, n+m+k$.

3. Insert the program

   $$P_3 := M[n+m+1, \ldots, n+m+k; 1]$$

   so the results are copied to the first k registers, and $g(f_1(r_1, \ldots, r_m), \ldots, f_k(r_1, \ldots, r_m))$ is computed.

4. Then $P := P_1, P_2, P_3$ computes h.

Note that $r \in \mathrm{dom}(h)$ iff $N_1(r)\downarrow, \ldots, N_k(r)\downarrow$ and $M(f_1(r_1, \ldots, r_m), \ldots, f_k(r_1, \ldots, r_m))\downarrow$.

What we have shown above is summarized as follows:

Proposition 2. *If m-ary functions $f_1, \ldots, f_k$ and a k-ary function g are URM-computable, then so is $g(f_1, \ldots, f_k)$.*

As a corollary, the following function is URM-computable:

Corollary 1. *If $f(x_1, \ldots, x_n)$ is URM-computable, so is $f(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$, where σ is a permutation on $\{1, \ldots, n\}$.*

Proof. For each $i \in \{1, \ldots, n\}$, the function $g_i(x_1, \ldots, x_n) = x_{\sigma(i)}$ is computed by $T(\sigma(i), 1)$, and $f(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) = f(g_1, \ldots, g_n)$. □

References

[1] N. Cutland, *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, (1980).