

pairing function*

rspuzio[†]

2013-03-21 17:49:59

A pairing function is a function $P: \mathbb{Z}_+^2 \rightarrow \mathbb{Z}_+$ which establishes a one-to-one correspondence between \mathbb{Z}_+^2 and \mathbb{Z}_+ . Such functions are useful in the theory of recursive functions because they allow one to express recursive functions of m variables in terms of recursive functions of n variables with $m \neq n$.

Two examples of pairing functions are the following;

$$P_1(x, y) = (x + y)(x + y + 1)/2 + y$$

$$P_2(x, y) = 2^x(2y + 1) - 1$$

It is not hard to see that these functions are recursive (actually, primitive recursive). For instance, one could use the recursion relations and initial conditions

$$P_1(x + 1, y) = P_1(x, y) + x + y + 1$$

$$P_1(0, y) = T(y) + y$$

$$T(y + 1) = T(y) + y + 1$$

$$T(0) = 0$$

where $T(n)$ is the n -th triangular number to show that P_1 is recursive. Likewise, one could use the recursions

$$P_2(x + 1, y) = P_2(x, y) + P_2(x, y)$$

$$P_2(0, y) = y + y$$

to show that P_2 is recursive.

An easy way to see that P_1 effects a one-to-one correspondence between \mathbb{Z}_+^2 and \mathbb{Z}_+ is as follows: Define the “successor” of a pair $(x, y) \in \mathbb{Z}_+^2$ to be the pair $(x - 1, y + 1)$ when $x \neq 0$; otherwise, when $x = 0$, the successor is $(y + 1, 0)$. It is easy to see that every pair has a successor and that every pair except $(0, 0)$ is the successor of exactly one other pair. With this definition of successor, the set of

**(PairingFunction)* created: *(2013-03-21)* by: *(rspuzio)* version: *(36140)* Privacy setting: *(1)* *(Definition)* *(03D20)*

[†]This text is available under the Creative Commons Attribution/Share-Alike License 3.0. You can reuse this document or portions thereof only if you do so under terms that are compatible with the CC-BY-SA license.

pairs of positive integers satisfies the Peano axioms and, hence, is isomorphic to the integers. From the definition of P_1 it follows that, if (x', y') is the successor of (x, y) , then $P_1(x', y') = P(x, y) + 1$ and that $P_1(0, 0) = 0$. This means that P_1 is the isomorphism described two sentences ago.

That P_2 effects a one-to-one correspondence between positive integers and pairs of positive integers follows readily from uniqueness of factorization of integers. On the one hand, for any number z , one can find numbers x and y such that $z = P_2(x, y)$ by factoring $z + 1$ and letting x be the power of 2 which appears in the factorization. On the other hand, this is the only solution of $z = P_2(x, y)$ because prime factorization is unique.

Since a pairing function P sets up a 1-1 correspondence between \mathbb{Z}_+ and \mathbb{Z}_+^n , there exist uniquely defined unpairing functions R and L such that

$$P(L(x), R(x)) = x$$

It is not hard to show that, if P is recursive, R and L will also be recursive.

Once one has a pairing function $P^{(2)}$, one can use it to set up 1-1 correspondences between \mathbb{Z}_+ and \mathbb{Z}_+^n for any n . For instance, one could define

$$P^{(3)}(x, y, z) = P^{(2)}(x, P^{(2)}(y, z))$$

$$P^{(4)}(x, y, z, w) = P^{(2)}(x, P^{(3)}(y, z, w)) = P^{(2)}(x, P^{(2)}(y, P^{(2)}(z, w)))$$

In general,

$$P^{(n+1)}(x_1, \dots, x_{n+1}) = P^{(2)}(x_1, P^{(n)}(x_2, \dots, x_n))$$

(This manner of encoding a list one pair at a time will be familiar to anyone who has programmed a computer in LISP. In fact, LISP was designed to be serve as a mathematical definition of computability equivalent to Turing machines or recursive functions. A fun exercise is to write a compiler which translates LISP programs into recursive functions using the representation of lists by single integers defined above.)

An important consequence of the fact noted above is that there is a 1-1 correspondence between recursive functions of n variables and recursive functions of a single variable. If we have a function $F: \mathbb{Z}_+^n \rightarrow \mathbb{Z}_+$, we can associate to it the function $G: \mathbb{Z}_+ \rightarrow \mathbb{Z}_+$ by the formula

$$F(x_1, \dots, x_n) = G(P^{(n)}(x_1, \dots, x_n))$$

Doing this can often save work by allowing one to draw conclusions about recursive functions of several variables from the special case of functions of one variable.