Generating-Functions of Interconnection Networks

Jayadev Misra
The University of Texas at Austin
Austin, Texas 78712, USA
email: misra@cs.utexas.edu

July 28, 1999

Abstract

Generating functions have long been used to analyze properties of sequences of numbers. In this note, we use generating functions to analyze a class of combinatorial objects, called interconnection networks. In particular, we prove that two families of interconnection networks are isomorphic by showing that the corresponding generating functions are isomorphic.

1 Introduction

Generating functions have long been used to analyze properties of sequences of numbers (see Chapter 7 of [5] for an excellent introduction to this topic). In this note, we use generating functions to analyze a class of combinatorial objects, called interconnection networks. In particular, we prove that two families of interconnection networks are isomorphic by showing that the corresponding generating functions are isomorphic.

Interconnection networks (also known as permutation networks) are used for regular interconnections of processors in a parallel computer. Such a network consists of N *nodes*, also known as switches, each of which has k input and output lines (except that *initial* nodes have no input lines and *final* nodes have no output lines). In this note $k = 2$, i.e., each node has two input and two output lines. It is often required to show that two given families are isomorphic, i.e., the networks corresponding to N, for each N, in both families are isomorphic (we define "isomorphic" later in this paper).

Each family consists of an infinite number of networks, for various values of N. The typical proof of isomorphism, see for instance [11, 1], is based on finding a mapping between the nodes of the two networks of size N. We proceed differently. We represent a family by a single function – its generating function – from which each member of the family can be generated by computing the function value with specific arguments. In this note, we show how the structure of a family can be described by a function on powerlists[9]; a short introduction to powerlists is given in section 3. The functional description can be used to deduce properties of the networks in the family. Also, different network families may be proven isomorphic by proving a certain kind of equivalence among the functions.

Powerlists are convenient for representing recursive structures; so, they are well-suited for representations of interconnection networks which are often recursive. We find that functions over powerlists are easier to manipulate than the traditional mappings between nodes.

The proofs in [11, 1] work only for networks in which k, the input (and output) degrees of the nodes, is 2. McIlroy and Savicki[7] have developed more sophisticated proofs for arbitrary k. Their proofs are also based on using mapping functions. The original powerlist theory, on which this paper is based, can handle only $k = 2$. Kornerup has extended the powerlist theory, and he has proven similar isomorphism results for arbitrary k, see [6].

2 Interconnection Networks

An interconnection network of size N, where $N = 2^n$, for some n, $n \geq 0$, has $n+1$ stages numbered 0 through n (the stages will appear in increasing order from left to right in the figures). Each stage has N nodes. Nodes in stage 0 are called *initial* nodes and those in stage n are *final* nodes. Each non-initial node has two input ports, known as *top* and *bottom*. Each non-final node has two output ports known as *top* and *bottom*. Each output port of a node in stage i is connected to a distinct input port of a node in stage $i+1$, $0 \leq i < n$. Examples of interconnection networks are butterfly networks[4], Benes[2] and Clos[3] networks. Each of these, e.g. butterfly network, actually denotes a family of networks where for each value of n there is a specific network in the family.

Two networks are *isomorphic* if they are isomorphic in a graph-theoretic sense; i.e., there is a 1-1 correspondence between the nodes of the two networks such that for two nodes u, v in one network where an output port (top or bottom) of u connects to an input port (top or bottom) of v, the corresponding nodes are similarly connected in the other network. Two families of interconnection networks are isomorphic if for each N, where N is a power of 2, the networks of size N from the families are isomorphic. We give an alternate characterization of isomorphism in section 5.

2.1 Benes Networks

We consider a family where the network for $N = 4$ is shown in Figure 1. The structure of the network is iterative; the connections are identical from stage to stage. For each non-initial stage, the top input lines of all nodes come in order from the upper half of the nodes of the previous stage and the bottom input lines come in order from the lower half.

Each network in this family is half of Benes network[2]; the other half is a mirror image of this half (the final nodes of the first half are the initial nodes of the second half). In this paper, we consider only the half of the network, as shown in Figure 1, and call it a Benes network.

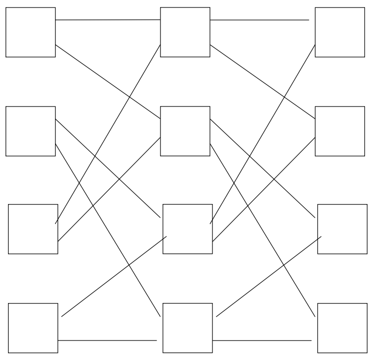

Figure 1: Benes Network, $N = 4$

2.2 Clos Networks

Now, we consider an interconnection network created in a recursive fashion. For $N = 1$, the network is a single node. For $N = 2$, the network is a butterfly network with 2 stages, as shown in Figure 2. We show the general construction scheme in Figure 3, for $N = 4$. For stage 1, the input lines of the nodes in the upper half are the top output lines of all the nodes of the previous stage and the input lines of the nodes in the lower half are the bottom lines of all nodes in the previous stage, in order. Next, two copies of the same network of the next smaller size, (for $N = 2$ in Figure 3), are appended to the upper and lower halves.

Each network in this family is half of Clos network[3]; the other half is a mirror image of this half. In this paper, we consider only the half of the network, as shown in Figure 3, and call it a Clos network.

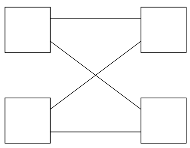

Figure 2: Clos Network for $N = 2$

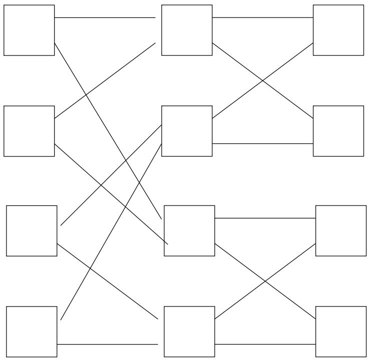

Figure 3: Clos Network, $N = 4$

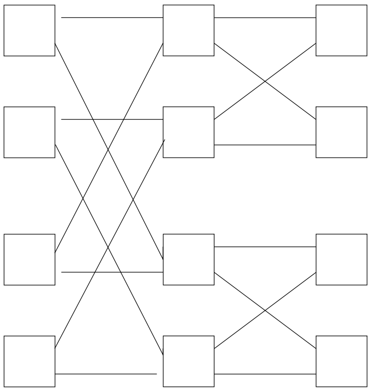

Figure 4: Butterfly Network for $N = 4$

2.3 Butterfly Network and its Mirror Image

A butterfly network[4] of size N, where $N = 2$ is shown in Figure 2, and for $N = 4$ is shown in Figure 4. The interconnection structure can be described as follows. The nodes in the upper half of the initial stage have their top lines connected to the top ports in the upper half of the next stage and their bottom lines connected to the top ports of the lower half of the next stage, in order. The connections for the lower half of initial nodes is analogous.

The mirror image of the butterfly network for $N = 8$ is shown in Figure 5.

3 Powerlists

The *powerlist* data structure was introduced in [9] to facilitate descriptions of parallel algorithms. The smallest powerlist—corresponding to the empty list for

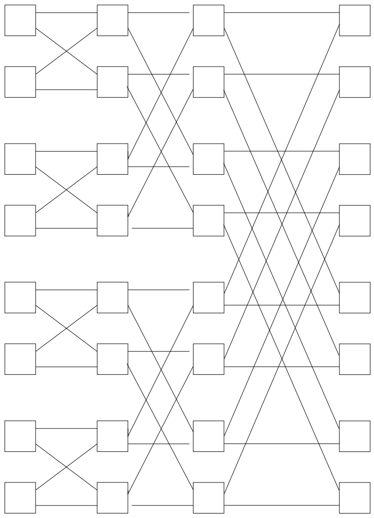

Figure 5: Mirror Image of the Butterfly Network, $N = 8$

the linear case—is a list of one element. There are two different ways in which two powerlists are joined to create a longer powerlist. If p, q are powerlists of the same length then

$p \mid q$ is the powerlist formed by concatenating p and q, and

$p \bowtie q$ is the powerlist formed by successively taking alternate items from p and q, starting with p.

Thus, the length of $p \mid q$ or $p \bowtie q$ is double the length of p (and q). Hence, the length of a powerlist is 2^n, for some n, $n \geq 0$. Powerlists can be nested, but we will not use that feature in this note.

In the following examples the sequence of elements of a powerlist are enclosed within angular brackets.

$\langle 0 \rangle \mid \langle 1 \rangle = \langle 0\ 1 \rangle$, $\langle 0 \rangle \bowtie \langle 1 \rangle = \langle 0\ 1 \rangle$,
$\langle 0\ 1 \rangle \mid \langle 2\ 3 \rangle = \langle 0\ 1\ 2\ 3 \rangle$, $\langle 0\ 1 \rangle \bowtie \langle 2\ 3 \rangle = \langle 0\ 2\ 1\ 3 \rangle$

The operation $\mid$ is called *tie* and $\bowtie$ is *zip*.

Convention: We write function application without parantheses where no confusion is possible. Thus, we write "$f\ x$" instead of "$f(x)$" and "$g\ x\ y$" instead of "$g(x, y)$". The constructors $\mid$ and $\bowtie$ have the same binding power and their binding power is lower than that of function application. □

Functions over linear lists are typically defined by case analysis—a function is defined over the empty list and, recursively, over non-empty lists. Functions over powerlists are defined analogously. For instance, the following function, *rev*, reverses the order of the elements of the argument powerlist.

$rev\langle x\rangle = \langle x\rangle$
$rev(p \mid q) = (rev\ q) \mid (rev\ p)$

The case analysis, as for linear lists, is based on the length of the argument powerlist. We adopt the pattern matching scheme of ML[8] and Miranda[10][1] to *deconstruct* the argument list into its components, p and q, in the recursive case. Deconstruction, in general, uses the operators $\mid$ and $\bowtie$. In the definition of *rev*, we have used $\mid$ for deconstruction; we could have used $\bowtie$ instead and defined *rev* in the recursive case by

$$rev(p \bowtie q) = (rev\ q) \bowtie (rev\ p)$$

It can be shown that the two proposed definitions of *rev* are equivalent and for any powerlist P

$$rev(rev\ P) = P.$$

3.1 Laws about powerlists

L0. For singleton powerlists, $\langle x\rangle, \langle y\rangle$
$\langle x\rangle \mid \langle y\rangle = \langle x\rangle \bowtie \langle y\rangle$

L1. (Dual Deconstruction)
For any non-singleton powerlist, P, there exist similar powerlists r, s, u, v such that
$P = r \mid s$ and $P = u \bowtie v$

L2. (Unique Deconstruction)
$(\langle x\rangle = \langle y\rangle) \;\equiv\; (x = y)$
$(p \mid q = u \mid v) \;\equiv\; (p = u \;\wedge\; q = v)$
$(p \bowtie q = u \bowtie v) \;\equiv\; (p = u \;\wedge\; q = v)$

L3. (Commutativity of $\mid$ and $\bowtie$)
$(p \mid q) \bowtie (u \mid v) = (p \bowtie u) \mid (q \bowtie v)$

These laws can be derived by suitably defining tie and zip, using the standard functions from the linear list theory. One possible strategy is to define tie as the concatenation of two equal length lists and then, use the Laws L0 and L3 as the definition of zip; Laws L1, L2 can be derived next. Alternatively, these laws may be regarded as axioms relating tie and zip.

Law L0 is often used in proving base cases of algebraic identities. Laws L1, L2 allow us to uniquely deconstruct a non-singleton powerlist using either $\mid$ or $\bowtie$. Law L3 is crucial. It is the only law relating the two construction operators, $\mid$ and $\bowtie$, in the general case. Hence, it is invariably applied in proofs by structural induction where both constructors play a role.

[1] Miranda is a trademark of Research Software Ltd.

3.2 An Example: The Function *inv*

We define a function *inv* that arises in a variety of contexts. In particular, *inv* is used to permute the output of a Fast Fourier Transform network into the correct order. This function is also central to our development of proofs of isomorphisms of interconnection networks.

For a powerlist of 2^n elements we associate an n-bit index with each element, where the indices are the binary representations of $0, 1, .., 2^n - 1$ in sequence. (For a powerlist $u \mid v$, indices for the elements in u have "0" as the highest bit and in v have "1" as the highest bit. In $u \bowtie v$, similar remarks apply for the lowest bit.) Any bijection, h, mapping indices to indices defines a permutation of the powerlist: the element with index i is moved to the position where it has index $(h\ i)$.

The function *inv* permutes the elements of the argument powerlist. It is defined by the following function on indices. An element with index b in P has index b' in $(inv\ P)$, where b' is the reversal of the bit string b. Thus,

$$\begin{array}{rccccccccll} & {\scriptstyle 000} & {\scriptstyle 001} & {\scriptstyle 010} & {\scriptstyle 011} & {\scriptstyle 100} & {\scriptstyle 101} & {\scriptstyle 110} & {\scriptstyle 111} & & \\ inv\langle & a & b & c & d & e & f & g & h & \rangle & = \\ \langle & a & e & c & g & b & f & d & h & \rangle & \end{array}$$

The definition of *inv* is

$$inv\langle x\rangle = \langle x\rangle$$
$$inv(p \mid q) = (inv\ p) \bowtie (inv\ q)$$

The following proof shows a typical application of structural induction.

INV1. $inv(p \bowtie q) = (inv\ p) \mid (inv\ q)$

Proof is by structural induction on p and q.

Base : $inv(\langle x\rangle \bowtie \langle y\rangle)$

$= \{\text{From Law L0} : \langle x\rangle \bowtie \langle y\rangle = \langle x\rangle \mid \langle y\rangle\}$

$inv(\langle x\rangle \mid \langle y\rangle)$

$= \{\text{definition of } inv\}$

$inv\langle x\rangle \bowtie inv\langle y\rangle$

$= \{inv\langle x\rangle = \langle x\rangle, inv\langle y\rangle = \langle y\rangle. \text{ Thus, they are singletons. Applying Law L0}\}$

$inv\langle x\rangle \mid inv\langle y\rangle$

Induction :

$inv((r \mid s) \bowtie (u \mid v))$

$= \{\text{commutativity of } \mid,\ \bowtie\}$

$inv((r \bowtie u) \mid (s \bowtie v))$

$= \{\text{definition of } inv\}$

$inv(r \bowtie u) \bowtie inv(s \bowtie v)$

$$= \quad \{\text{induction}\}$$

$$(\mathit{inv}\ r \mid \mathit{inv}\ u) \bowtie (\mathit{inv}\ s \mid \mathit{inv}\ v)$$

$$= \quad \{\mid,\ \bowtie\ \text{commute}\}$$

$$(\mathit{inv}\ r \bowtie \mathit{inv}\ s) \mid (\mathit{inv}\ u \bowtie \mathit{inv}\ v)$$

$$= \quad \{\text{apply definition of } \mathit{inv} \text{ to both sides of } \mid\}$$

$$\mathit{inv}(r \mid s) \mid \mathit{inv}(u \mid v) \qquad \square$$

Using INV1 and structural induction, it is easy to establish

$$\mathit{inv}(\mathit{inv}\ P) = P\ ,$$

$$\mathit{inv}(\mathit{rev}\ P) = \mathit{rev}(\mathit{inv}\ P).$$

4 Generating-Functions of Networks

We represent a family of networks by a function from powerlists to powerlists, as described below.

4.1 Notation

In this note, all powerlist elements are strings over some given alphabet. For a powerlist p,

$\overline{p}$ is the logarithm (base 2) of the length of p,
$\{p\}$ is the set of elements of p,
$p0$ is the powerlist obtained by concatenating "0" to the end of each element of p,
$p1$, similarly, is the powerlist obtained by concatenating "1" to the end of each element of p,
$p.q$, where p and q have the same length, is the powerlist obtained by concatenating corresponding elements of p and q in this order.

We use the following precedence among the various operators: concatenation with 0 or 1 binds the strongest followed by . and then $\mid$ and $\bowtie$ as operators of equal binding power. Thus, $p0.q0 \bowtie p1.q1$ is $[(p0).(q0)] \bowtie [(p1).(q1)]$.

Observations: For powerlists p, q, u, v of the same length

1. $\{u \bowtie v\} = \{u\} \cup \{v\}$, and $\{u \mid v\} = \{u\} \cup \{v\}$.
2. $\overline{p} = \overline{p0} = \overline{p.q}$.
3. $(p \bowtie q)0 = (p0 \bowtie q0)$, and $(p \mid q)0 = (p0 \mid q0)$.
4. $(p \bowtie q).(u \bowtie v) = (p.u) \bowtie (q.v)$, and $(p \mid q).(u \mid v) = (p.u) \mid (q.v)$.
5. For any function h that permutes the elements of a powerlist, $(h\ p)0 = h(p0)$ and $h(p.q) = (h\ p).(h\ q)$.

4.2 Naming the Nodes and Lines

We adopt the following scheme to name the nodes and lines of a network. Name each node in stage 0 by a distinct symbol from some alphabet. For a node named b, name its top output line $b0$ and its bottom output line $b1$. A node whose top input line is named u and the bottom v, is assigned $u.v$ as its name. Thus, in Figure 6, given that the nodes in stage 0 are named a, b, c, d from top to bottom, other nodes are named as shown in that figure. It is clear that given the *list* of names for the nodes in stage 0, all the node and line names are determined. Further, given the *set* of node names at the last stage, it is possible to reconstruct the names assigned to all the nodes, lines and their interconnections.

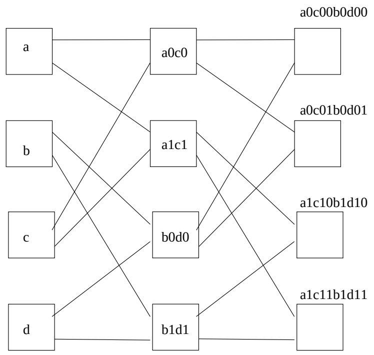

Figure 6: Naming the Nodes in a Network

We describe the structure of a network by a function whose argument is a powerlist of names, to be assigned in sequence to the nodes in stage 0, and whose result is either a powerlist of node names or a *set* of final node names.

4.3 Generating-Functions for the Example Networks

We describe the structure of a Benes network using the *set* of names for the final nodes. The set of names is adequate to describe the structure of a network and, hence, prove isomorphism of two network structures. We describe the other networks by functions that return a powerlist of names for the final nodes. The powerlist representation is useful when we wish to concatenate different networks; then functional composition corresponds to network concatenation.

Benes Network: In the following, function b_k describes the structure of the Benes network that has $k + 1$ stages.

$$b_0\,(p) = \{p\}$$
$$b_{k+1}(u \mid v) = b_k(u0.v0 \bowtie u1.v1), \text{ for } 0 \le k \le \overline{u}$$

Clos Network: The following function, c, describes the Clos network.

$$c(\langle x \rangle) = \langle x \rangle$$
$$c(u \bowtie v) = c(u0.v0) \mid c(u1.v1)$$

Butterfly Network and its Mirror Image: The butterfly network is described by function f and its mirror image by m.

$$f(\langle x \rangle) = \langle x \rangle$$
$$f(u \mid v) = f(u0.v0) \mid f(u1.v1)$$

$$m(\langle x \rangle) = \langle x \rangle$$
$$m(u \bowtie v) = m(u0.v0) \bowtie m(u1.v1)$$

5 Isomorphisms of the Example Networks

Two families of interconnection networks, given by generating functions g and h, are isomorphic provided $g = h \circ \pi$ for some permutation function π (function composition is denoted by $\circ$). Isomorphism is an equivalence relation from this definition, and it is equivalent to the previous definition of isomorphism.

We show that all four networks – Benes, Clos, butterfly and its mirror image – are isomprphic. We prove these results by showing that the corresponding functions return the same *set* of final node names given identical *list* of names for the initial nodes; see corollary at the end of theorem 2. In some cases – see theorem 1 – we can identify the exact permutation that is to be applied to the input list to make the resulting lists of final nodes identical. In the following theorem, we use the permutation function *inv* which was defined in section 3.2 as follows.

$$inv(\langle x \rangle) = \langle x \rangle$$
$$inv(p \mid q) = (inv\ p) \bowtie (inv\ q)$$

Theorem 1:

1. $c \circ inv = f$
2. $c = inv \circ m$

Proof: All the proofs are by induction on the structures of the argument powerlists. The base case in each identity is straightforward. So, we prove the result only for the inductive case.

1. Let the argument powerlist be $p \mid q$.

$$\begin{array}{ll} & c[inv(p \mid q)] \\ = & \{\text{definition of } inv\} \\ & c[(inv\ p) \bowtie (inv\ q)] \\ = & \{\text{definition of } c\} \end{array}$$

$$
\begin{array}{cl}
 & c[(\mathit{inv}\ p)0.(\mathit{inv}\ q)0] \mid c[(\mathit{inv}\ p)1.(\mathit{inv}\ q)1] \\
= & \{(\mathit{inv}\ t)0 = \{\text{Observation 5}\}\ \mathit{inv}(t0), \text{ for any powerlist } t\} \\
 & c[(\mathit{inv}\ p0).(\mathit{inv}\ q0)] \mid c[(\mathit{inv}\ p1).(\mathit{inv}\ q1)] \\
= & \{(\mathit{inv}\ r).(\mathit{inv}\ s) = (\mathit{inv}\ r.s), \text{ from Observation 5}\} \\
 & c[\mathit{inv}(p0.q0)] \mid c[\mathit{inv}(p1.q1)] \\
= & \{\text{induction. Note that } \overline{p} = \overline{p0.q0} = \overline{p1.q1}\} \\
 & f(p0.q0) \mid f(p1.q1) \\
= & \{\text{Definition of } f\} \\
 & f(p \mid q) \qquad \Box
\end{array}
$$

2. Let the argument powerlist be $u \bowtie v$.

$$
\begin{array}{cl}
 & \mathit{inv}(m(u \bowtie v)) \\
= & \{\text{definition of } m\} \\
 & \mathit{inv}[m(u0.v0) \bowtie m(u1.v1)] \\
= & \{\text{Property INV1 of } \mathit{inv}\} \\
 & \mathit{inv}[m(u0.v0)] \mid \mathit{inv}[m(u1.v1)] \\
= & \{\text{induction}\} \\
 & c(u0.v0) \mid c(u1.v1) \\
= & \{\text{definition of } c\} \\
 & c(u \bowtie v) \qquad \Box
\end{array}
$$

Corollary: $f \circ \mathit{inv} = \mathit{inv} \circ m$ $\Box$

In the following theorem, $\{f(u)\}$ is the set of values in the powerlist $f(u)$.

Theorem 2: $b_{\overline{u}}(u) = \{f(u)\}$

Proof: First, we prove a lemma about b.

Lemma: $b_n(u \bowtie v) = (b_n\ u) \cup (b_n\ v)$, for all n, $0 \leq n \leq \overline{u}$.
Proof: We prove the result by induction on n. For $n = 0$, we have,

$$
\begin{array}{cl}
 & b_0(u \bowtie v) \\
= & \{\text{definition of } b\} \\
 & \{u \bowtie v\} \\
= & \{\text{Observation 1}\} \\
 & \{u\} \cup \{v\} \\
= & \{\text{definition of } b\} \\
 & (b_0\ u) \cup (b_0\ v)
\end{array}
$$

For the inductive case, let $n = k + 1$, for some $k \geq 0$. Since $n \leq \overline{u}$, u (and v) is not a singleton list. Let u be $p \mid q$ and v be $r \mid s$. We show that for all k, $0 \leq k + 1 \leq \overline{p \mid q}$ (or equivalently, $0 \leq k \leq \overline{p}$),

$b_{k+1}[(p \mid q) \bowtie (r \mid s)] = b_{k+1}(p \mid q) \cup b_{k+1}(r \mid s)$.

$$
\begin{array}{cl}
 & b_{k+1}[(p \mid q) \bowtie (r \mid s)] \\
= & \{\text{commutativity of } \mid \text{ and } \bowtie\}
\end{array}
$$

$$
\begin{array}{cl}
 & b_{k+1}[(p \bowtie r) \mid (q \bowtie s)] \\
= & \{\text{definition of } b\} \\
 & b_k([(p \bowtie r)0.(q \bowtie s)0] \bowtie [(p \bowtie r)1.(q \bowtie s)1]) \\
= & \{\text{commutativity of zip and concatenation with 0, Observation 3}\} \\
 & b_k([(p0 \bowtie r0).(q0 \bowtie s0)] \bowtie [(p1 \bowtie r1).(q1 \bowtie s1)]) \\
= & \{\text{induction, } 0 \le k \le \overline{p} < \overline{p0 \bowtie r0} = \overline{(p0 \bowtie r0).(q0 \bowtie s0)}\} \\
 & b_k[(p0 \bowtie r0).(q0 \bowtie s0)] \cup b_k[(p1 \bowtie r1).(q1 \bowtie s1)] \\
= & \{\text{commutativity of . and } \bowtie \text{, Observation 4}\} \\
 & b_k[(p0.q0) \bowtie (r0.s0)] \cup b_k[(p1.q1) \bowtie (r1.s1)] \\
= & \{\text{induction, } 0 \le k \le \overline{p} = \overline{p0.q0}\} \\
 & b_k(p0.q0) \cup b_k(r0.s0) \cup b_k(p1.q1) \cup b_k(r1.s1) \\
= & \{\text{rearranging the terms}\} \\
 & b_k(p0.q0) \cup b_k(p1.q1) \cup b_k(r0.s0) \cup b_k(r1.s1) \\
= & \{\text{induction, } 0 \le k \le \overline{p} = \overline{p0.q0}\} \\
 & b_k(p0.q0 \bowtie p1.q1) \cup b_k(r0.s0 \bowtie r1.s1) \\
= & \{\text{definition of } b\} \\
 & b_{k+1}(p \mid q) \cup b_{k+1}(r \mid s) \qquad \Box
\end{array}
$$

Now we are ready to prove the main result: $b_{\overline{u}}(u) = \{f(u)\}$. The base case is straightforward. For the inductive case, let the argument powerlist be $p \mid q$.

$$
\begin{array}{cl}
 & b_{\overline{p \mid q}}(p \mid q) \\
= & \{\text{definition of } b\} \\
 & b_{\overline{p}}(p0.q0 \bowtie p1.q1) \\
= & \{\text{Lemma}\} \\
 & b_{\overline{p}}(p0.q0) \cup b_{\overline{p}}(p1.q1) \\
= & \{\text{induction. Note that } \overline{p} = \overline{p0.q0} = \overline{p1.q1}\} \\
 & \{f(p0.q0)\} \cup \{f(p1.q1)\} \\
= & \{\text{Observation 1}\} \\
 & \{f(p0.q0) \mid f(p1.q1)\} \\
= & \{\text{Definition of } f\} \\
 & \{f(p \mid q)\} \qquad \Box
\end{array}
$$

Corollary: $b_{\overline{u}}(u) = \{f(u)\} = \{c(u)\} = \{m(u)\}$. □

Acknowledgement: I am indebted to Doug McIlroy who suggested that powerlists may be effective in proving properties of interconnection networks. Jacob Kornerup showed me how to extend these results to networks of higher degrees.

References

[1] D.P. Agrawal. Graph theoretic analysis and design of multistage interconnection networks. *IEEE Trans. on Computers*, C-32:637–648, 1983.

[2] V. E. Benes. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, New York, 1965.

[3] C. Clos. A study of non-blocking switching networks. *Bell Syst. Tech. J.*, 32:406–424, 1953.

[4] J. M. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19(90):297–301, 1965.

[5] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley Publishing Company, 1989.

[6] Jacob Kornerup. *Data Structures for Parallel Recursion.* PhD thesis, University of Texas at Austin, 1997. Available for download as http://www.cs.utexas.edu/users/kornerup/dis.ps.Z.

[7] M. D. McIlroy and J. P. Savicki. Isomorphism of classical rearrangeable networks. In D.-Z. Du and F. K. Hwang, editors, *Advances in Switching Networks*, volume 42 of *DIMACS Series in Discrete Math. and Theoretical Comp. Sci.*, pages 147–156, Providence, 1998. American Math. Soc.

[8] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.

[9] J. Misra. Powerlist: A structure for parallel recursion. *ACM Transactions on Programming Languages and Systems*, 16(6):1737–1767, November 1994.

[10] David Turner. An overview of Miranda. *ACM SIGPLAN Notices*, 21:156–166, December 1986.

[11] C. Wu and T. Feng. On a class of multistage interconnection networks. *IEEE Trans. on Computers*, C-29:694–702, 1980.