PROPERTIES OF MULTISTAGE INTERCONNECTION NETWORKS

Abdou S. Youssef
(Thesis)

CS-TR-135-88

February 1988

To the memory of my father,

Sleyman R. Youssef

PROPERTIES OF MULTISTAGE INTERCONNECTION NETWORKS

Abdou S. Youssef

A Dissertation
Presented to the
Faculty of Princeton University
In Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance by the
Department of
Computer Science

February 1988

ABSTRACT

Regular rectangular multistage interconnection networks that are complete and have the single path property are increasingly important in large parallel computing systems. The efficiency of such networks is critical to the overall system performance, and it depends on the structure, functional capabilities and routing control of the network.

Much of the previous research has focused on specific networks. The objective of this dissertation is to study and characterize an important class of such networks. In particular, the relationships between network functionality and topology, switch size, control and modularity.

The one-to-one correspondence between topology and functionality is shown, necessary and sufficient topological conditions for a network to realize all the permutations of another network are established, and optimal algorithms to decide if a network realizes all the permutations of another are developed.

The single path property makes control via control tags potentially efficient. Two different control schemes are introduced where the control tags are the same as, or a function of, destination tags. The structure of these "easy-to-control" networks is shown to be recursive. Based on this recursiveness, the networks of several interesting network subclasses are shown to be functionally equivalent, namely, the subclass of *doubly controllable* networks, that is, easy-to-control from left to right and from right to left, the subclass of *modular* networks where all the stages are identical, and the subclass of *r-ary* networks, where the stages permute and transform *r*-ary digits.

These single path networks do not realize all permutations. Multiple path Benes networks do but are a slow-to-control alternative. Accordingly, a new scheme for fast control of 3-stage Benes networks is introduced and studied, and several problems are shown to fit the new scheme, namely, bitonic sorting, FFT, tree algorithms and matrix computations. Also, the random walk computation is parallelized and shown to fit this control scheme.

ACKNOWLEDGEMENT

I would like to thank first and foremost, my supervisor Dean Bruce Arden for his guidance, steadfast support and encouragement. I am particularly grateful for the generous use of his time in reading and polishing this dissertation. His supervision has contributed greatly to my intellectual growth.

I also want to thank Professors Andrea Lapaugh and Rafael Alonso for reading this dissertation. Their insightful criticisms and comments improved its quality considerably.

I am very grateful to all the department faculty members, particularly Prof. Ken Steiglitz, Bernard Chazelle and Hector Garcia-Molina. I extend my thanks to the department secretaries, especially Sharon Rodgers, Winnie Waring, Gerry Pecht and Eugene Davidson, for their assistance and kindness throughout.

I would like to express my appreciation to my friends and fellow graduate students, particularly Toshio Nakatani, Patricia Simpson, Arvin Park, Pandu, Steve Kugulemass, Mike Laszlo, Robert Abbott, Claire Mathiew, Boris Kogan, William Lin, Susan Yeh, Deborah Silver, Ken Salem, Luen Heng, Mark Weiss, Luis Cova, Dimitris Dukas and others. They created a pleasant and productive work environment.

I felt fortunate to have had Mohammed Dahleh, Khaled Nuseibeh and Monsef Ben Abd Aljalil as roommates and dear friends. Their support and understanding helped me immensely throughout my graduate years.

I also would like to express my deep thanks to my dear friend Haydee Claus who has been a source of strength and caring.

I am especially grateful to my family for their love which has nurtured me throughout my studies in the United States.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

B(r,k) : The $r^k \times r^k$ Baseline network where the building block is an $r \times r$ switch

BC(r,k) : The $r^k \times r^k$ Benes network where the building block is an $r \times r$ switch

BF(r,k) : The $r^k \times r^k$ butterfly network where the building block is an $r \times r$ switch

CT : Control Tag

DPN : Digit-Permutation Network

FFT : Fast Fourier Transform

G(r,k) : The $r^k \times r^k$ Generalized cube network

G(W) : The representing banyan of the network W

GEMINE: GEneralized Multistage Interconnection NEtworks

GRN : Generalized Recursive Network

I(r,k) : The $r^k \times r^k$ Indirect r-ary cube network

IMIN : Incomplete Multistage Interconnection Network

MIN : Multistage Interconnection Network

MRB : Modular Recursive Banyan

PDE : Partial Differential Equation

pwc : Permute Within Column

sws : Swap Within Switch

$\Omega(r,k)$: The $r^k \times r^k$ Omega network where the building block is an $r \times r$ switch

$\Omega^{-1}(r,k)$: The $r^k \times r^k$ Omega Inverse where the building block is an $r \times r$ switch

$S_{N,r,i}$: Shuffle within segment of length r^{k-i} where $N = r^{k-i}$

$U_{N,r,i}$: Unshuffle within segment of length r^{k-i} where $N = r^{k-i}$

$g_{N,r,i}$: A digit Permutation : $x_{k-1} \cdots x_0 \rightarrow x_{k-1} \cdots x_{i+2} x_0 x_i \cdots x_1 x_{i+1}$

$h_{N,r,i}$: A digit Permutation : $x_{k-1} \cdots x_0 \rightarrow x_{k-1} \cdots x_{k-i} x_0 x_{k-i-2} \cdots x_1 x_{k-i-1}$

Chapter 1

Introduction

Regular, single path, complete interconnection networks are increasingly important in large parallel computing systems [FEN81], [THU74]. Their importance grows with the need for fast computation and with the increasing feasibility of systems of thousands of processing elements afforded by the VLSI technology. The lower cost of multistage networks renders them preferable to crossbars, the single stage version of such networks, and their reconfigurability makes them more functionally capable than fixed interconnection networks.

The efficiency of these networks is critical to overall system performance, and it depends on the structure, functional capabilities and routing control of the network. Several networks of this type have been proposed and studied, such as omega networks [LAW75], indirect binary n-cube [PEA76], baseline [WU80b], and the generalized cube network [SIE79].

Previous research has focused for the most part on the aforementioned networks with respect to functionality, control, and other aspects [WU80a-b], [LAW75], [SIE78], [SIE79], [SIE81], [BAT74]. Recently, however, it was shown that most of the proposed networks were functionally equivalent [WU80a]. This increased interest in network equivalence in general [ORU85a,b,c], [ATT84]. Oruc developed conjugate equivalence algorithms of complexity $O(N^2 \log N)$ for the 2×2 switch case, and $O(N^3 \log^2 N)$ for larger switch sizes [ORU85a].

There remains a strong need to look at these kinds of regular networks as a class and to study certain aspects in that context rather than in specific networks.

The primary objective of this dissertation is to study and characterize this network class. In particular, the goal is to study the relationships between network functionality and topology, switch size, control and modularity.

The interdependence of functionality and topology is quite strong. Two networks are topologically equivalent if one is a different layout of the other. Clearly, changing the layout does not alter the functionality. Thus, two topologically equivalent networks have the same functionality. It is interesting to examine the converse. We prove that two networks having the same functionality must be topologically equivalent, establishing thus the equivalence between functionality and topology, and paving the way for an optimal functional equivalence algorithm, of complexity $O(N \log N)$.

The next variable on which functionality depends is the switch size, where the switch is the building block. It should be noted that a good measure of functionality is the set of permutations a network realizes without conflict. A simple combinatorial argument shows that the number of realizable permutations grows as the switch size increases. But what is more important is how the *set* of realizable permutations, rather than its cardinality, changes as the switch size increases. Stated otherwise, it is important to establish necessary and sufficient conditions on two arbitrary networks W and W', of respective switch sizes r and s, for W to realize all permutations realizable by W'. W is then said to *functionally cover* W'. It will be shown that W functionally covers W' if and only if r is a power of s and W can be transformed to a network that is topologically equivalent to W' by replacing each $r \times r$ switch of W by an $r \times r$ network whose building block is an $s \times s$ switch. This equivalence between functional covering and "topological covering" will lead to an optimal $O(N \log_s N)$ algorithm to decide functional covering.

A major aspect of these kinds of networks is their controllability and the control mechanisms' dependence on network topology. Understanding the

relationship between control and topology is important to the design of efficiently controllable networks. Controlling a network involves establishing the paths between sources and destinations via control tags. An "irregular topology" would require storing the control tags which correspond to the source-destination pairs. The amount of storage required for large systems is prohibitive. We introduce two control schemes in which control tags are easy to compute and do not have to be stored, such as those where the source-destination control tag is the destination address or a function thereof. The first control scheme, where the control tag is the destination address is called *D-control*, and the second, where the control tag is a function of the destination address, is called *FD-control*. The structure of networks with these control schemes is one of the main issues studied and it will be shown to be recursive. Other control related issues are studied, particularly double controllability, where a network is D- or FD-controllable from left to right and from right to left. It will be shown that all such networks are topologically (and hence functionally) equivalent to the baseline network.

The subclass of networks, where the interconnections between columns can be described by operations that permute bits, includes all existing networks of interest. The $r \times r$ generalization of these networks are called r-ary digit permutation networks. Necessary and sufficient conditions for a logarithmic number of label transformations to yield complete networks are derived, and an efficient control scheme is developed. Moreover, it will be shown that all these networks are doubly FD-controllable and hence functionally equivalent.

The fourth relationship involves a curious feature of modularity. A network is modular if all its stages are identical. Omega and inverse omega [LAW75] are two examples. Modularity has two advantages. The first is manufacturing efficiency. The second is the potential hardware saving because the whole modular network can be replaced and simulated by a single stage by using as many passes through the

single stage as the number of stages of the network. These advantages motivate the study of modular networks, their functionality and their underlying structure. This study will lead to a result of "reductive nature": All modular networks that are D- or FD-controllable turn out to be topologically (and thus functionally) equivalent to omega, and hence of little interest.

Control efficiency is not without a price. The major deficiency of the networks of interest is their incapability of realizing all permutations without conflict. The rearrangeable Benes network removes this deficiency but at the expense of control efficiency [BEN65], [WAK68], [NAS80]. Control tags are insufficient because there are multiple paths between sources and destinations. Controlling Benes networks involves either computing the switch configurations of permutations *a priori* and storing them, or computing them during execution. The first way is costly in space for large systems, and the second is costly in time [WAK68].

Two different approaches have been introduced to bypass this control complexity. The first, due to Nassimi and Sahni [NAS81], consists of using destination addresses in a specified manner, and allows for efficient realization of a subset of permutations. The second, due to Lenfant [LEN78], identifies what are called "frequently used permutations," and develops a specialized control algorithm that realizes these permutations efficiently.

Another objective of this dissertation is to develop a new scheme for fast control of 3-stage Benes networks. The Benes network of size $N = r^2$ has 3 stages of $r \times r$ crossbar switches. The scheme consists of fixing the first stage to some configuration h so that the remaining two stages are self-controlled. A family of permutations for which there exists an h such that the remaining network passes the family without conflict is called *compatible*. All Ω-conflict-free permutations are compatible because setting the first stage to the identity configuration yields a network that functionally covers Ω. Compatibility is characterized and its

determination is shown to reduce to a graph coloring problem. The characterization gives a powerful technique to determine compatibility, by virtue of which the families of permutations for bitonic sorting and FFT are proved to be compatible.

The last objective is to illustrate a few parallel techniques by developing an algorithm for the random walk computations (RWC) in the Monte Carlo solution of partial differential equations; to design a suitable architecture for the algorithm; and to simulate that architecture on 3-stage Benes by a minimum number of compatible permutations. The algorithm is a three-phase tree-structured algorithm and employs a function composition technique which could also be used to parallelize recurrence equations. The architecture consists of a pipeline of three trees of processing elements. The efficient simulation of the architecture on 3-stage Benes by compatible permutations illustrates the wide applicability of the new control scheme of 3-stage Benes.

The dissertation is organized as follows. Chapter 2 reviews multistage interconnection networks, and defines a specific class. In Chapter 3, the equivalence between topology and functionality of this class is proved and a functional equivalence algorithm is given, analyzed and shown to be optimal.

Chapter 4 deals with network covering, and gives an optimal algorithm to decide functional covering. Also, the definitions of some of the existing networks are generalized to accommodate various switch sizes, and each of these networks is shown to be functionally covered by the network in the same class but whose switch size is a power of the old size.

Chapter 5 explores the topological structure of D-controllable and FD-controllable networks. It also analyzes doubly controllable networks proving their functional equivalence to the baseline network.

Chapter 6 studies r-ary digit permutation networks, develops an efficient control scheme for them, and shows that every such network is doubly FD-controllable and thus functionally equivalent to the baseline.

In Chapter 7, modular networks are studied and shown to be functionally equivalent to omega.

Chapter 8 introduces the new scheme of fast control of 3-stage Benes networks, characterizes the compatibility and shows that each of the families of permutations of FFT and bitonic sorting is compatible. In Chapter 9 a parallel algorithm for the random walk computation is given and integrated into the Monte Carlo solution of partial differential equations. A suitable architecture for the algorithm is designed and shown to be efficiently mapped onto 3-stage Benes with the new control scheme.

Chapter 10 concludes the dissertation and suggests natural extensions and possible directions for further research.

One final note. The proofs of some theorems in the following chapters involve certain intermediary steps that bear no immediate relevance to the central issues. Those proofs have been moved to the appendices.

Chapter 2

Multistage Interconnection Networks

In this chapter the themes of the dissertations are stated along with the necessary mathematical objects and preliminaries. The class of interconnection networks of interest is specified, existing instances is surveyed, relations within the class are defined, some network aspects are discussed, and new related concepts are introduced. Banyan networks, a related family, provide a useful graph representation and abstraction. This family is presented along with some of its basic features. Lastly, Benes networks are presented.

2.1 Overview of Multistage Interconnection Networks

In this section the interconnection networks of interest and their charateristics will be specified and discussed, and their general routing control will be described and categorized. As this class is closely related to banyan networks, and as some of the network characteristics are better dealt with in terms of banyan properties, banyan networks are presented and discussed first.

2.1.1 Banyan Networks

Banyan networks were defined and studied in [GOK73] as partitioning networks. They were further studied by others [LIP77], [LIP79], [PRM81], [DEG81].

A banyan network is a *directed graph* with two special subsets of nodes, the *bases* and the *apexes*. A base has no incoming arcs, and an apex has no outgoing arcs. The other nodes, with both types of connrcting arcs, are called intermediate. The network has to be *complete* and *single path*, that is, there is a single path between each base and each apex. Fig. 2.1 shows some banyan networks.

The single path property implies that each base is the root of a tree, and the completeness implies that the leaves of the tree are the apexes. Hence, a banyan network can be viewed as a network of superposed trees rooted at the bases and having the apexes as leaves. It follows that every intermediate node is the root of a subtree whose leaves are a subset of the apexes.

An *L-level* banyan is a banyan where the nodes can be arranged into $L+1$ levels such that the arcs can exist only between adjacent levels. Note that all base-apex paths have the same length L. Fig. 2.1b, shows an L-level banyan.

A regular banyan is a banyan where all the non-base nodes have the same indegree F (called the fanout), and all the non-apex nodes have the same outdegree S (called spread). Fig. 2.1c shows a regular banyan.

It was shown in [GOK73] that the i-th level of an L-level regular banyan has S^iF^{L-i} nodes, where the levels are numbered 0, 1, ..., L, from base-level to apex-level.

A *rectangular* banyan is an L-level regular banyan of fanout F and spread S where $F = S = r$. It follows then that every level i has the same number of nodes $S^iF^{L-i} = r^ir^{L-i} = r^L$. r is said to be the degree of the network. Fig. 2.1d shows a rectangular banyan.

Rectangular banyans are of special interest because they represent a wide class of multistage interconnection networks that are the focus of this dissertation.

A multistage interconnection network W can be derived from a rectangular banyan of degree r by replacing each node of the latter by an $r \times r$ permutation switch such that each of the r outgoing arcs is now linked to exactly one output port of the switch, and similarly, each incoming arc is linked to exactly one input port of the switch. The input ports of the switches that replace the bases are the input terminals of the W, and the output ports of the switches that replaced the apexes are the output terminals. Each switch can be configured to link its input ports to its output ports in any of the $r!$ possible ways. Between each input terminal and each output terminal

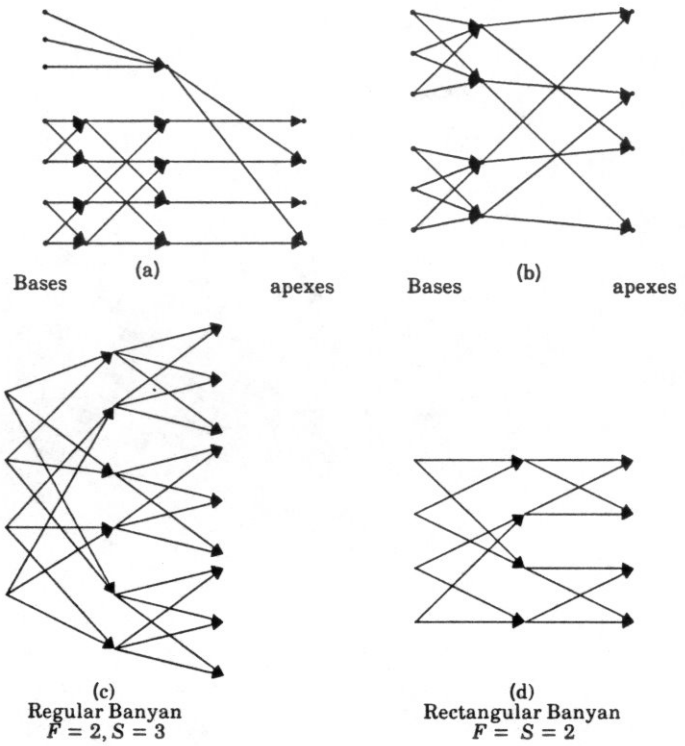


Fig. 2.1
Banyan Networks

there exists one path where the intermediate nodes are switches. Each switch can be configured to link the incoming arc of the path to the outgoing arc of the path. Hence, W is complete, that is, each input terminal can reach all the outputs. W is also single

path, that is, there are no multiple paths between any input terminal and any output terminal.

The banyan network underlying W is called the *representing banyan* of W, and is denoted G(W).

Fig. 2.8 is a network that can be derived from the rectangular banyan of Fig. 2.1d. Alternately, the rectangular banyan of Fig. 2.1d is the representing banyan of the network of Fig. 2.8.

The preceding description illustrates that there are many variants of banyan networks but they all have distinguished input and output nodes with unique, usually multistep paths between every pair of input and output nodes. The added restriction that the interconnecting nodes have the same degree leads to the subclass of banyans that are the subject of this dissertation.

It is possible to characterize this subclass starting with two properties. Completeness is required for communication between all inputs and outputs and the single path property simplifies routing through a network. From these two properties one can conclude that such networks must be directional. Excluding the trivial case where the number of inputs or the number outputs is unity, the multistep paths must be directional. If links can be traversed in either direction, completeness implies that there are multiple paths connecting every input and output. That contradicts the single path property.

If such directional networks, or graphs, are further constrained to be regular in the banyan sense, other important properties can be derived. Let the N_{in} input nodes have outdegree = r, the N_{out} output nodes have indegree = r, and all the intermediate nodes have indegree = outdegree = r. The following lemma and theorem show that such complete, single path r-regular networks must be rectangular with uniform length k paths and $N_{in} = N_{out} = r^k$.

Lemma 2.1: If an interconnection network is complete, single path and r-regular, then $N_{in} = N_{out}$.

Proof: The network can be considered bidirectional in that the simultaneous direction reversal of all connecting links simply reverses the roles of the input and output nodes and the directions of all interconnecting single paths are reversed. There are $N_{in}.N_{out}$ paths through the network. If $N_{in} \neq N_{out}$, say, $N_{in} < N_{out}$, then every input node branches to more leaves than the output nodes do. But the input and output nodes are leaves of r-ary trees in both directions. The number of leaves can be greater in one direction than the other, with the same path lengths in both directions, only if there is a branch greater than r somewhere in the tree, contradicting the hypothesis. Thus, $N_{in} = N_{out}$. □

Theorem 2.1: If an interconnection network is complete, single path, and r-regular with N inputs and outputs and at least one maximal path of length k, then all paths are of length k and $N = r^k$.

Proof: Consider an input node to be the root of a partial r-ary tree with at least one path of length k. Let n_i be the number of terminating paths of length i (from the root) and r_i be the total number of paths of length i.
This is a single partial r-ary tree. To have N such superposed trees the N-1 other input roots must connect to the r_i nodes for the smallest i such that $n_i \neq 0$. There are r^i nodes at this distance i. These N-1 roots cannot connect at some distance $j > i$ because the unidirectionality would make paths to the n_i terminating (i.e., output) nodes at distance i impossible. Thus, there must be a complete, single path, connecting r-ary network from N roots to r_i nodes at distance i. From the previous lemma the number of inputs must equal the number of outputs, in this case r^i, i.e., $N = r^i$. Moreover, every node of the r^i nodes at distance i must be a terminating node because otherwise there would branch out of the $(r^i - n_i)$ non-terminating nodes more

than $(r^i - n_i)$ terminating nodes, making the number of all the terminating nodes greater than $n_i + r^i - n_i = r^i = N$.
Consequently, $n_i = r^i = N$, and all the apexes are at distance i. In particular, $k = i$. It follows that $N = r^k$ and all the trees rooted at the bases must be complete r-ary trees of depth k. Every input-output path is then of length k. □

Since the superposed r-ary trees are all complete, the nodes can then be broken into levels where there are edges only between adjacent levels. The r-regularity forces the number of nodes in each level to be the same as the number of inputs. Thus, all levels have $N = r^k$ nodes. Hence, such networks are rectangular.

The combinatorics of banyan networks are complicated, and will be discussed briefly, showing the vast size of the class of regular rectangular banyan networks.

The number of r-degree interconnections between seperate levels where each level has r^k nodes can be shown to be

$$f(r,k) = \left[\binom{r^k}{r} \binom{r^k-r}{r} \binom{r^k-2r}{r} \binom{r^k-3r}{r} \cdots \binom{r^k-(r^{k-1}-1)r}{r} \right]^r = \frac{(r^k!)^r}{(r!)^{r^k}}$$

(r^{k-1} terms)

However, it should be noted that if k such graphs are concatenated, the resulting network need not be complete and single path.

It is not difficult to obtain a more restricted combinatorial expression for the number of complete trees that can be superposed on $r^k \times k$ $r \times r$ nodes but, other than further illustrating the combinatorial explosion, it is not particularly helpful.

Along these lines, a subclass of regular rectangular banyan networks is specified here recursively, and its size is still very large. For $k = 1$, the subclass has only one network which is the complete $r \times r$ bipartite graph. For larger k, the networks of level size r^k are constructed by putting in parallel r arbitrary networks in that subclass of level size r^{k-1}, then add one level of size r^k to one side of these networks and connect it to their bases so that each of the bases of each of these networks can reach a subset of r nodes that is disjoint from other such subsets reached from the other bases of the same network. The resulting network is clearly a rectangular banyan.

The number of ways of making this last interconnection is obviously $f(r,k)$. the number of ways of constructing the connection next to it must then be $[f(r,k-1)]^r$, and so on. Therefore, the number of different banyan networks in that subclass is

$g(r,k) = f(r,k)\,[f(r,k-1)]^r\,[f(r,k-2)]^{r^2}\,[f(r,k-3)]^{r^3} \ldots [f(r,k-1)]^{r^{k-1}}$.

For $r = 2$, and for the following different values of k, the number of these banyan networks is:

$g(2,1) = 1$

$g(2,2) = 36$

$g(2,3) = 6350400$

$g(2,4) =$ larger than 23×10^{40}.

These figures show the rapid growth of the number of regular rectangular banyan networks.

2.1.2 Definition of the class of interconnection networks of interest

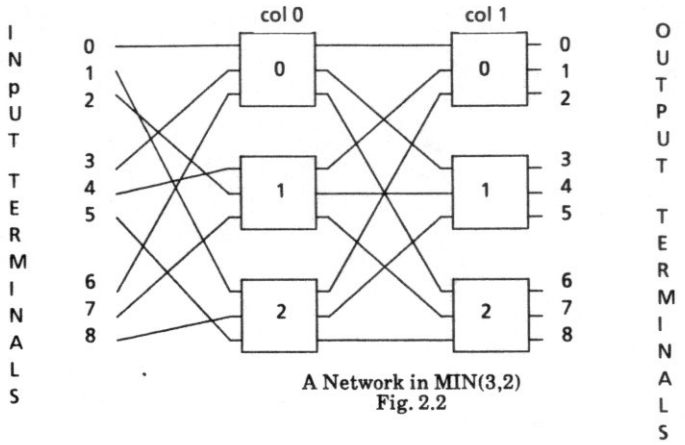
The interconnection networks dealt with throughout this dissertation are derived from regular rectangular banyans as described earlier. In particular, they have $N = r^k$ input terminals on the left end, N output terminals on the right end, and $\log_r N = k$ columns of $r \times r$ permutation switches (i.e., regular, directional graphs, where in-degree = out-degre = r), each column has N/r switches. Only successive columns are interconnected, the input terminals are linked to the leftmost column and the output terminals are linked to the rightmost column.

The banyan formulation, with its distinctive base and apex nodes, precludes $k = 1$, a single crossbar. Our formulation does not preclude this special case but the emphasis is on $k > 1$, hence the use of "multistage" in the descriptions.

This class of networks is denoted MIN(r,k), and the acronym "MIN" is derived from Multistage Interconnection Networks. These networks are complete and single path. Fig. 2.2 shows a network in MIN(3,2).

Throughout this dissertation, networks are considered to be in MIN(r,k) unless stated otherwise.

MIN's are used for communication in shared memory models (processor-to-memory) as in Fig. 2.3, or in private memory models (pe-to-pe) as in Fig. 2.4. In the shared memory model, the input terminals are the processors, the output terminals the memory modules, and the reverse direction is obviously used also. Note that the terminals are of degree 1, and hence, each terminal is linked to one switch only. In private memory models, input terminal i and output terminal i are identical and represent the i-th processing element. In this case, the network has conceptually a cylindrical shape, although, for the purposes in this dissertation, all the networks will be shown in a "flat", two dimensional fashion, where the left end represents the input terminals and the right end the output terminals.



A Network in MIN(3,2)
Fig. 2.2

The input terminals, the output terminals, the input ports and the output ports of each column are numbered *0, 1, ... , r^k-1* from top to bottom. The columns are numbered *0, 1, ... , k-1* from left to right, and the switches of each column are numbered *0, 1, ... , r^{k-1} - 1* from top to bottom. The interconnection between each two successive columns can be viewed as a permutation *f* of $S_N = \{0, 1, ... , N-1\}$, such that the output port *i* of the left column is linked to the input port *f(i)* of the right column. Switch *i* of column *j* is identified by switch *(i,j)*. The interconnection between column *i* and column *i + 1* is denoted (col *i* , col *i+1*), the leftmost interconnection (- , col 0) and the rightmost interconnection (col *k-1* , -). These interconnections are also referred to as stages. The link between a port *i* and a port *j* in a specific stage is identified by link *(i , j)*.

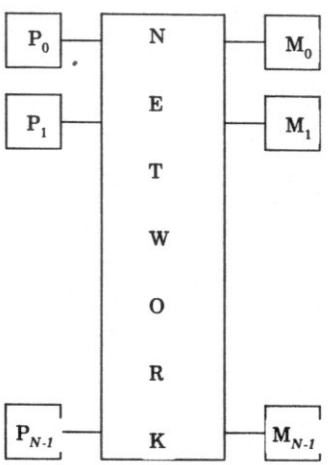


Fig. 2.3
Shared Memory Model

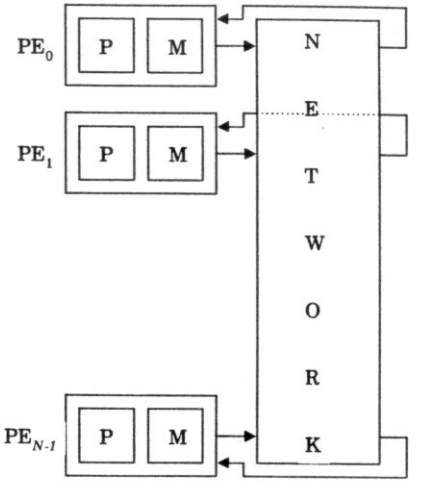


Fig. 2.4
Private Memory Model

If (- , col 0) is the identity permutation, the network is said to be *left bare-ended*, and if (col *k-1* , -) is the identity, then the network is *right bare-ended*. A network is *bare-ended* if it is both left bare-ended and right bare-ended.

2.1.3 Permutation networks

These multistage interconnection networks are best viewed as *permutation* networks. A permutation π is *realizable* by a network if the switches can be configured so that input terminal *i* is simultaneously connected to output terminal π(*i*), for all $i = 0, 1, ... , N-1$. The network is also said to *pass* π *without conflict*. The

vector notation is used for permutation representations, where the i-th component is the mapping of i. For example, π = [5 7 3 6 0 1 2 8 4] is the permutation that maps 0 to 5, 1 to 7, 2 to 3, ... , 8 to 4. This permutation is realizable by the network in Fig. 2.2. The corresponding configuration is shown in Fig. 2.5.

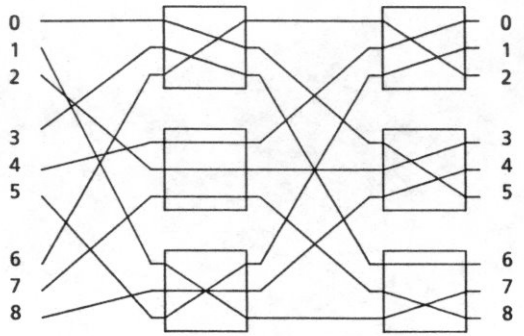


Fig. 2.5

Permutation [5 7 3 6 0 1 2 8 4]

Not every permutation is realizable by a network in MIN(r,k). For instance, the permutation [0 2 3 4 5 6 1 7 8] conflicts in the network of Fig. 2.2 because 0 is mapped to 0, 6 is mapped to 1, and the paths 0 → 0 and 6 → 1 conflict over the link (0,0) of stage (col 0 , col 1).

The number of realizable permutations of any network in MIN(r,k) can be easily computed due to the single path property [LAW75]. It is equal to the number of all possible configurations of the network because each configuration uniquely corresponds to a permutation. Each $r \times r$ switch has $r!$ configurations corresponding to the number of permutations of $\{0, 1, \ldots, r\text{-}1\}$. As the number of switches in the

network is kr^{k-1}, the number of configurations of the network is then $(r!)^{kr^{k-1}}$. Therefore, the number of realizable permutations of a network in MIN(r,k) is $(r!)^{kr^{k-1}}$. This number is a small fraction of $r^k!$, the overall number of permutations of S_N. In fact, $((r!)^{kr^{k-1}} / r^k!) \to 0$ as $r^k \to \infty$ for a fixed r. This fact has prompted interest in equivalence algorithms to decide if two networks in MIN(r,k) realize the same permutations. This is the subject of Chapter 3.

2.1.4 Network control

The main advantage of MIN's, beside their hardware efficiency, arises from the single path property. Dynamic, local routing, called self-routing here, is possible using control tags (CT) to establish source-destination paths. Since there is only one path between any input terminal i and any output terminal j in a network W in MIN(r,k), there exists a unique r-ary number $a_{k-1}a_{k-2} \ldots a_0$ which can be used to establish the path $i \to j$.

To show that, let $s_0, s_1, \ldots, s_{k-1}$ be the consecutive switches lying on the path between i and j. The unique link between switch s_l and switch s_{l+1} that lies on the path $i \to j$ leaves switch s_l from relative output port b_l ($0 \le b_l \le r\text{-}1$). Let a_l be b_l for all $l = 0,1, \ldots, k\text{-}1$. Note that s_k is considered output terminal j. Now that $a_{k-1}a_{k-2} \ldots a_0$ is defined, it can be used as a control tag as follows. Input terminal i sends $a_{k-1}a_{k-2} \ldots a_0$ along the control lines of W, and col l uses the r-ary digit a_{k-1-l} to link the input port of the switch s_l, to which the control signal comes, to output port a_{k-1-l} of s_l. Clearly then, switch s_0 uses digit a_{k-1}, s_1 uses digit a_{k-2} and so on, establishing the path along the way to output terminal j.

The uniqueness of the path between i and j implies the uniqueness of the number $a_{k-1}a_{k-2} \ldots a_0$. This number is denoted CT(i,j,W), or just CT(i,j) when no confusion arises.

Example: CT (3,6) = 011 (in binary) for the network in Fig. 2.1. CT(3,6) = 20 (in ternary) for the network in Fig. 2.2.

It is clear that the control tag $a_{k-1}a_{k-2} \ldots a_0$ corresponding to the path $i \rightarrow j$ depends on the structure of W. The CT(i,j,W)'s can be computed and stored in a 2-dimensional array such that the i-th row is stored in the input terminal i. The amount of storage needed is $N^2k\lceil\log_2 r\rceil$ bits because there are N^2 entries and each entry needs $k\lceil\log_2 r\rceil$. For large systems where $N \geq 2^{16}$, the memory size is more than 64 G bits, which is prohibitive. Therefore, what is needed arenetworks for which the CT(i,j)'s are simple to compute and do not require storage. For instance, when CT(i,j) = j, or when CT(i,j) = $f(j)$ for some easy-to-compute permutation f.

No research has yet attempted to categorize networks according to their routing control. Rather, the trend has been to propose and study individual networks and the peculiar routing control of each.

We introduce in this dissertation three control schemes or categories, and study the structure of networks of each category. The first, *D-control*, is when the control tags are the same as the destination addresses; the second, *FD-control*, when the control tags are functions of destination addresses; and the third, *G-control*, the general schemes where the control tags are functions of both sources and destinations:

(a) *D-control*: CT(i,j) = j.

(b) *FD-control*: CT(i,j) = $f(j)$, where f is a permutation of $S_N = \{0,1, \ldots ,N-1\}$.

(c) *G-control*: CT(i,j) = $f(i,j)$, where $f(i,.)$ as a function of j is a permutation of S_N for every fixed i.

It turns out that all existing MIN's, surveyed and generalized in the next section, fall into the first two categories,

Definition 2.1: A network W in MIN(r,k) is *D-controllable* if its control belongs to the D-control scheme, *FD-controllable* if its control belongs to the FD-control scheme, and *G-controllable* if its control belongs to the G-control scheme.

Note that the D-controllable networks are a subset of the FD-controllable networks, which in turn are a subset of the G-controllable networks. Note also that every network in MIN(r,k) is G-controllable.

Definition 2.2: The *inverse* of a network W, denoted W^{-1}, is the mirror image of W. In other terms, it is the network W except that the input terminals are the output terminals and the output terminals the input terminals.

Definition 2.3: A network W in MIN(r,k) is *doubly D-controllable* (*doubly FD-controllable*) if both W and W^{-1} are D-controllable (FD-controllable).

The structure of D-controllable networks, FD-controllable networks, doubly D-controllable networks, and doubly FD-controllable networks is the focus of Chapter 5.

2.2 Survey of Existing MIN's

In this section most MIN's that have been proposed and studied are surveyed and generalized. They are omega, omega inverse, baseline, indirect binary n-cube, butterfly and generalized cube network, all defined with 2×2 switches as building blocks. The generalizations of these networks are defined to include switches of arbitrary size.

One common pattern among all these networks is that their stages are defined using digit permutations, that is, permutations that manipulate the digits of the r-ary representations of integers. Some of these permutations are defined next.

Definition 2.4: Let $N = r^k$ and $x_{k-1}x_{k-2}...x_0$ be an arbitrary r-ary number.

$S_{N,r,i}(x_{k-1}x_{k-2}...x_0) = x_{k-1}...x_{k-i}x_{k-i-2}...x_1x_0x_{k-i-1}$: Shuffle within segments of length r^{k-i}.
$U_{N,r,i}(x_{k-1}x_{k-2}...x_0) = x_{k-1}...x_{k-i}x_0x_{k-i-1}...x_1$: Unshuffle within segments of length r^{k-i}.
$g_{N,r,i}(x_{k-1} ... x_0) = x_{k-1} ... x_{i+2}x_0x_i ... x_1x_{i+1}$
$h_{N,r,i}(x_{k-1} ... x_0) = x_{k-1} ... x_{k-i}x_0x_{k-i-2} ... x_1x_{k-i-1}$
e: The identity permutation.

2.2.1 Omega networks and their inverse ($\Omega(r,k)$ and $\Omega^{-1}(r,k)$)

$\Omega(r,k)$ is defined as follows. (-,col 0) = (col i, col $i+1$) = $S_{N,r,0}$ for i = 0, 1, ... , k-2, and (col k-1, -) = e. $\Omega(2,k)$ is defined in [LAW75]. Fig. 2.6 shows $\Omega(2,3)$ and Fig. 2.7 $\Omega(3,3)$.

$\Omega^{-1}(r,k)$ is the inverse of $\Omega(r,k)$ and can then be defined as follows. (col k-1,-) = (col i, col $i+1$) = $U_{N,r,0}$ for i = 0, 1, ... , k-2, and (-, col 0) = e.

2.2.2 The baseline networks (B(r,k))

B(r,k) is defined as follows. (-, col 0) = (col k-1, -) = e, (col i, col i+1) = $U_{N,r,i}$ for i = 0, 1, ... , k-2. B($2,k$) is defined in [WU80b]. Fig. 2.8 shows B(2,3) and Fig. 2.9 B(3,3).

2.2.3 Indirect r-ary n-cube (I(r,k))

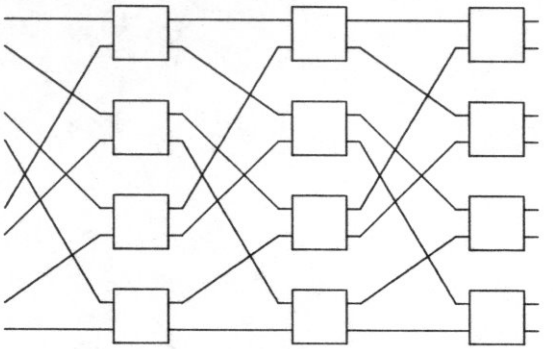
The indirect r-ary n-cube I(r,k) is defined as follows. (col k-1,-) = (-,col 0) = e, (col i, col i+1) = $g_{N,r,i}$, for i = 0, 1, ... , k-2. I($2,k$) is defined in [PEA76]. Fig. 2.10 shows I(2,3) and Fig. 2.11 I(3,3).

2.2.4 Butterfly networks (BF(r,k))

The butterfly BF(r,k) is defined as follows. (col k-1,-) = $U_{N,r,0}$, (-,col 0) = e, (col i, col i+1) = $g_{N,r,i}$, for i = 0, 1, ... , k-2. Fig. 2.12 shows BF(2,3) and Fig. 2.13 BF(3,3).

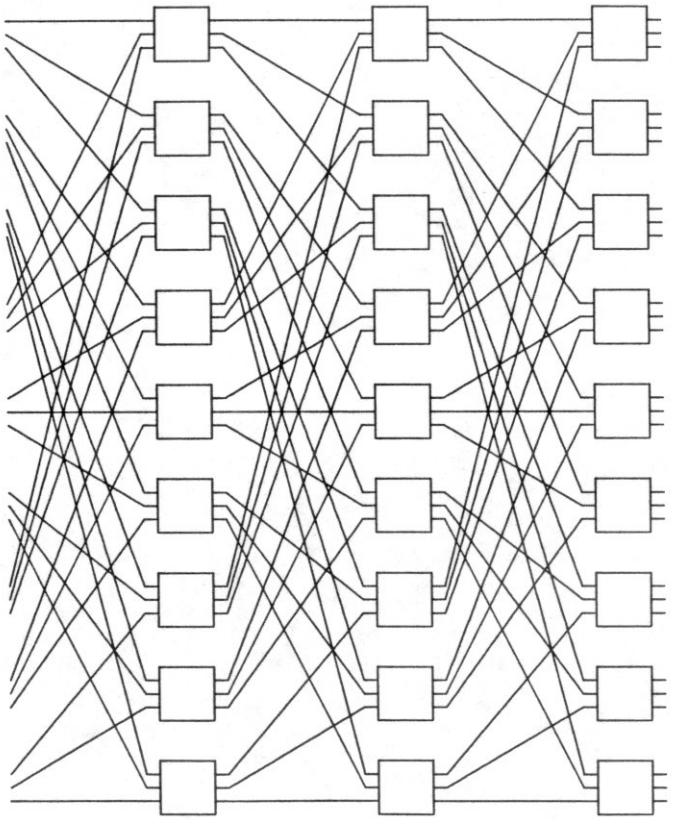
2.2.5 Generalized cube networks (G(r,k))

G(r,k) is defined as follows. (-,col 0) = $S_{N,r,0}$, (col k-1,-) = e, (col i, col $i+1$) = $h_{N,r,i}$, for i = 0, 1, ... , k-2. G($2,k$) is defined in [SIE79]. G(r,k) can be shown to be BF^{-1}(r,k). Thus, G(2,3) and G(3,3) are the mirror images of BF(2,3) and BF(3,3) shown in Fig. 2.12 and Fig. 2.13, respectively.



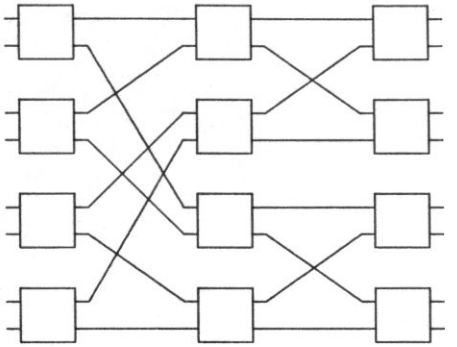
Omega $\Omega(2,3)$

Fig. 2.6



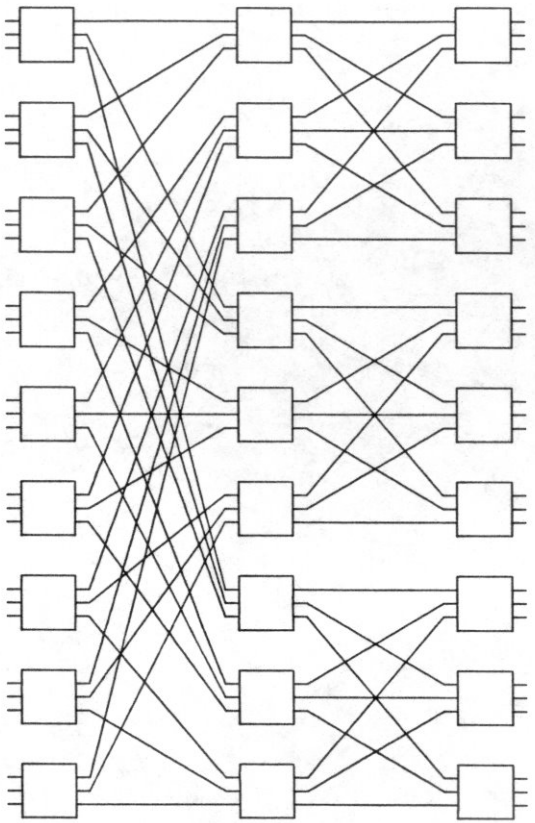
Omega Ω(3,3)

Fig. 2.7



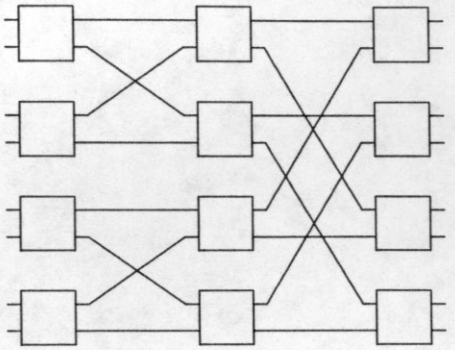
Baseline B(2,3)

Fig. 2.8



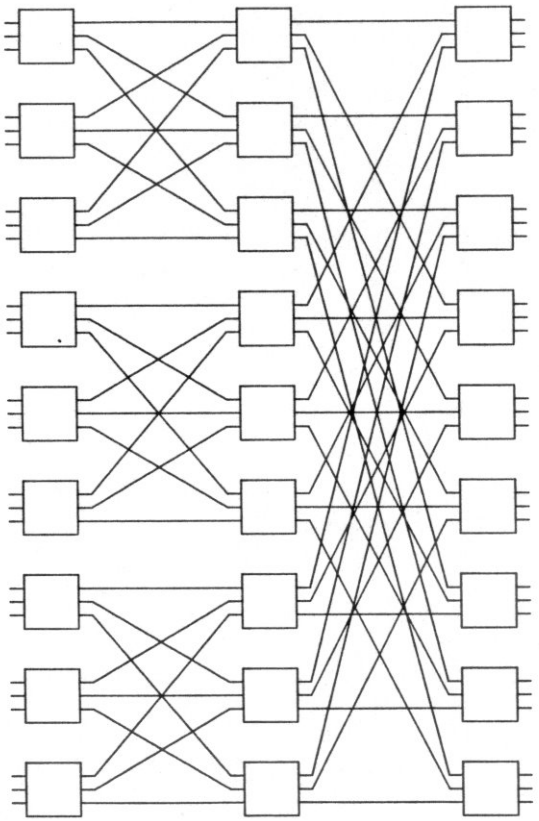
Baseline B(3,3)

Fig. 2.9



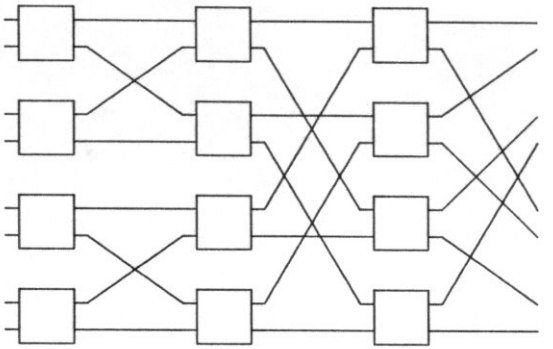
Indirect binary n-Cube(2,3)

Fig. 2.10



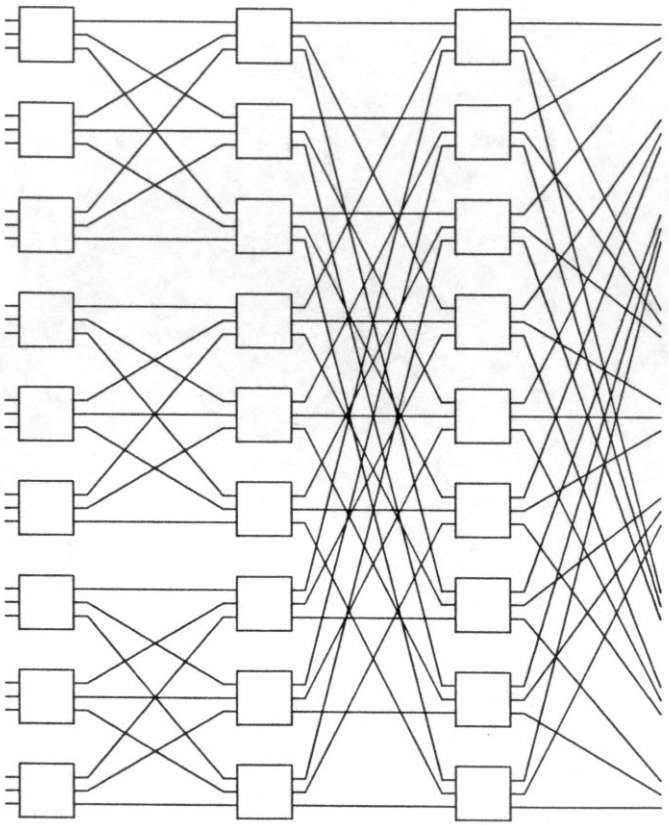
Indirect 3-ary n-Cube I(3,3)

Fig. 2.11



Butterfly BF(2,3)

Fig. 2.12



Butterfly BF(3,3)

Fig. 2.13

2.3 Incomplete, Single Path Networks

One of the implications of the single path property of the networks in MIN(r,k) is that each input terminal i is the root of a full r-ary tree of depth $k = \log_r N$ where the leaves are the output terminals and the internal nodes at level t are the input ports of the switches of col t that are reachable from i, for all $t = 0, 1, \ldots, k\text{-}1$ (see Fig. 2.14). In particular, each input port of each switch of col t is the root of a full r-ary subtree of depth $k\text{-}t$ embedded in the columns $t, t+1, \ldots, k\text{-}1$. The leaves of that subtree are the only output terminals reachable from that input port. Stated otherwise, between any input port of any switch and any output terminal there is either one single path or no path at all.

It can then be concluded that if a number of stages (say $k\text{-}l$) are deleted from the left end of a network W in MIN(r,k) (as in Fig. 2.15 and 2.16), the resulting network W' has $l < k$ columns and is incomplete and single path. Precisely, the number of output terminals reachable from any input terminal in W' is r^l.

The class of such networks W' is denoted IMIN(r,k,l), where $l < k$. This notation is derived from Incomplete Multistage Interconnection networks.

2.4 Network Relations

As mentioned earlier, networks in MIN(r,k) do not realize all permutations. Consequently, functional relationships among MIN's are of special interest. Since topology and functionality are interdependent, topological relationships among MIN's are of interest too.

Several network relations have been defined [WU80a], [ORU85a, b], [SIE78], [ATT84]. In this section, these relation are presented and further refined, and new relations are introduced. A few operations on interconnections and networks are defined first.

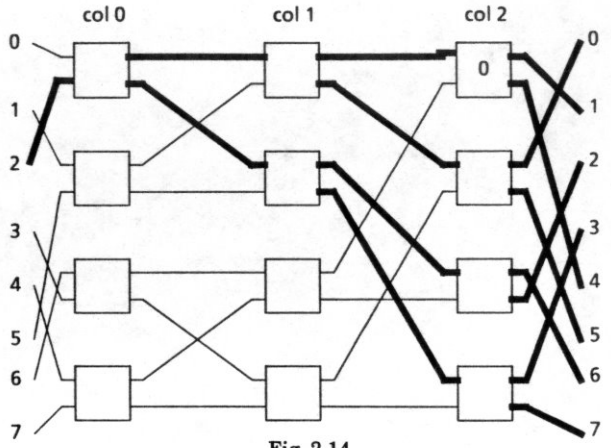


Fig. 2.14
An Embedded Full Binary Tree

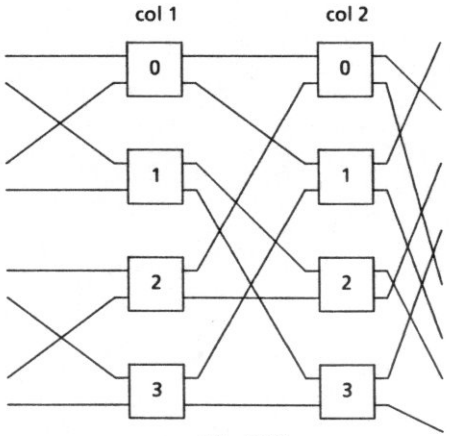


Fig. 2.15
The first stage deleted
from the network of Fig. 2.14

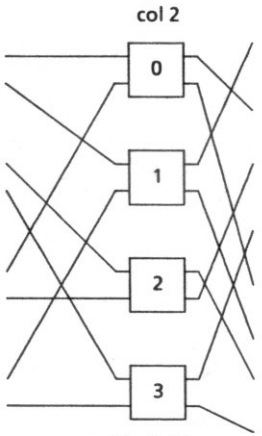


Fig. 2.16
The first two stages deleted
from the network of Fig. 2.14

If f is a permutation of $S_N = \{0, 1, \ldots, N\text{-}1\}$, by *interconnection* f is meant the bipartite graph (S_N, S_N, E) where $E = \{(i,f(i)) \mid i \in S_N\}$. If W is in MIN(r,k), *appending* f to the left of W results in a new network denoted fW such that $(\text{-}, \text{col } 0)_{fW} = f(\text{-}, \text{col } 0)_W$ (composition of permutations) and all the other stages of fW are the same as the corresponding stages of W. Appending f to the right of W is defined similarly, resulting in a network denoted Wf such that $(\text{col } k\text{-}1, \text{-})_{Wf} = (\text{col } k\text{-}1, \text{-})_W f$.

By *relabeling* the input terminals (resp., the output terminals) of W by f it is meant that input terminal i (resp., output terminal i) is relabeled $f(i)$, for all i. The inputs (resp. outputs) are also said to be *permuted by* f. Clearly, relabeling the input terminals of W by f is equivalent to appending interconnection f^{-1} to the left of W, and relabeling the output terminals by f is equivalent to appending interconnection f to the right of W.

Relabeling inputs or outputs by f can also be thought of as *repositioning* them so that input i (output i) is moved to position $f(i)$.

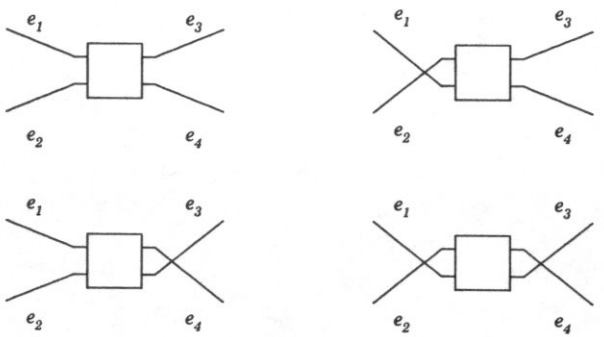
Permuting a column of a network W by f means that each switch i of that column is moved (along with the links connected to it) to position $f(i)$. This is also called a *permute within column* (pwc) operation.

If the points of arrivals (resp., departures) of the links coming to (resp., going from) a switch in a network are swapped (i.e., permuted), the operation is called *swap within switch* (sws). Fig. 1.17 shows such operations for the 2×2 switch case.

2.4.1 Functional relations

Definition 2.5: Two networks W and W' are *strongly equivalent* ($W \equiv W'$) if they realize the same permutations.

Definition 2.6: Two networks W and W' are *weakly equivalent* ($W \approx W'$) if W' can realize the same permutations as W after relabeling its input and/or output



SWS Operation

A change from one to another is a (sws)

Fig. 2.17

terminals. Alternately, there exists two permutations f and g such that W is strongly equivalent to $fW'g$.

Definitions 2.6, 2.7 and 2.8 adapt to the shared memory model, but for the private memory model a new equivalence relation has to be defined. It is a special case of Definition 2.6.

Definition 2.9: Two networks W and W' are *conjugately equivalent* (or *conjugates*) if W' can pass the same permutations as W after relabeling its input terminals and its output terminals *identically*. That is, if input i is relabeled r_i, then output i is also relabeled r_i [ORU85-b].

From Definitions 2.5 and 2.6 and from the discussion of terminal relabeling and interconnection appending, the following theorem follows easily.

Theorem 2.2: W and W' are weakly equivalent if and only if there exist two permutations f and g such that W and $fW'g$ are strongly equivalent.

Strong equivalence is also called strict functional equivalence, and weak equivalence called wide functional equivalence [ORU85-b], [SIE78].

2.4.2 Topological relations

Note that sws and pwc operations applied on a network do not alter its underlying structure. This justifies the following definition.

Definition 2.10: Two networks W and W' are *strongly topologically equivalent* if W' can be transformed to W by some sequence of sws and pwc operations.

Definition 2.11: Two networks W and W' are *weakly topologically equivalent* if W' can be transformed to W by some sequence of sws and pwc operations and by relabeling the input and output terminals of W'.

It can be clearly seen that strong topological equivalence implies strong equivalence, and weak topological equivalence implies weak equivalence. The converse is not obvious. It will be proved in the next chapter.

2.4.3 Network covering

In the two preceding subsections, relations among networks in the same class MIN(r,k) were defined. In this subsection, relations among two networks, one in MIN(r,k) and the other in MIN(s,k') where $r^k = s^{k'}$, are defined.

Definition 2.12: Let W be in IMIN(r,k,l) and W' in IMIN(s,k',t) where $l \leq k$ and $t \leq k'$. W is said to *functionally cover* W' if (1) every permutation realizable by W' is realizable by W and (2) if there is a path between an input terminal and output terminal in one network, then there is a path between these terminals in the other network.

Note that if W is MIN(r,k) and W' in MIN(s,k'), condition (2) of Definition 2.12 is always satisfied.

Definition 2.13: A network W in IMIN(r,k,l) is said to *topologically cover* another network W' in IMIN(s,k',t) if $r = s^m$ for some integer m and the $r \times r$ switches of W can be replaced by networks in MIN(s,m) so that the resulting network is strictly topologically equivalent to W'.

If W topologically covers W', W is said to *unfold* to W', W' is called an *unfolding* of W, and W a *folding* of W'.

Note that topological covering implies functional covering, but the converse is not obvious. It will be shown in Chapter 4.

2.5 Modular Interconnection Networks

Definition 2.14: A network W in MIN(r,k) is *modular* if it is right bare-ended and all its stages are identical.

Omega networks are clearly modular (see Fig. 2.6 and Fig. 2.7). Fig. 2.18 shows another modular network.

In Chapter 7, modular D-controllable networks will be shown to be strongly equivalent to omega networks, and modular FD-controllable networks conjugately equivalent to omega.

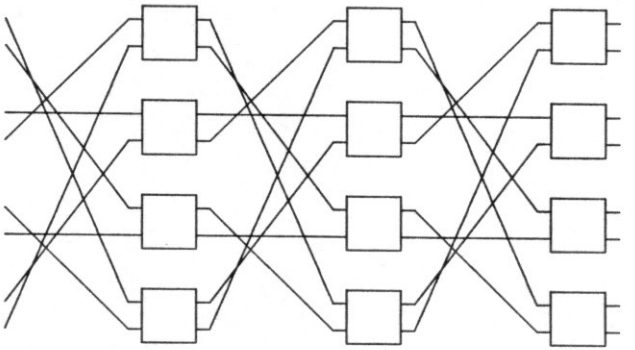


Fig. 2.18
A Modular Network

2.6 Benes Networks

Benes networks (also called Bene-Clos networks (BC)) realize all possible permutations without conflict and have therefore a clear functional advantage over logarithmic stage networks. However, the routing control of BC's is costly [BEN65],

[WAK68], [NAS80]. Chapter 8 will introduce and study a new control scheme for 3-stage Benes networks that reduce the control time for certain classes of problems.

In this section, these networks are defined. The definitions are from [BEN65].

Definition 2.15: A Benes network BC(r,k) has $N = r^k$ input terminals, N output terminals, $2k\text{-}1$ columns of r^{k-1} $r\times r$-switches each. The connectivity between the columns is the following: (-, col 0) = (col $2k\text{-}2$, -) = e, where $k = \log_r N$, (col i, col $i+1$) $= U_{N,r,i}$ for $i = 0, 1, \ldots, k\text{-}2$, and (col i, col $i+1$) $= S_{N,r,2k-3-i}$ for $i = k\text{-}1, k, \ldots, 2k\text{-}3$.

This regular, rectangular network is necessarily complete, but not single path. It has $2k\text{-}1$ columns. Therefore, it is not a MIN in our definition of the term but, interestingly, it is composed of two concatenated MIN's with the identity center stage and the two adjacent switch columns combined into a single column of $r\times r$ switches. One can conclude that there are r^k paths between any pair of input and output terminals.

Note that the middle $2k\text{-}3$ columns of BC(r,k) form a column of r Benes networks BC(r,$k\text{-}1$). The interconnection between the leftmost column of BC(r,k) and the column of the smaller Benes is $U_{N,r,0}$, and that between the the smaller Benes and the rightmost column of BC(r,k) is $S_{N,r,0}$.

Consequently, Benes networks can be defined recursively, noting that BC(r,1) is merely an $r\times r$ switch. This is how they were defined in [BEN65].

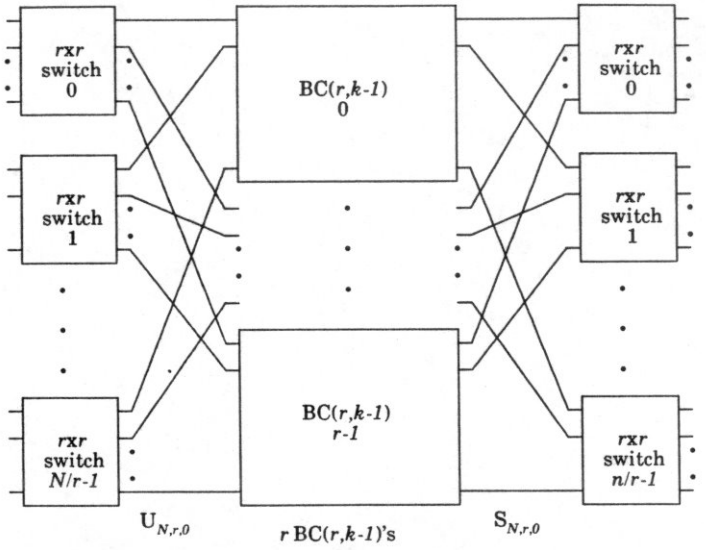
Fig. 2.19 shows the recursive construction of BC(r,k). Fig. 2.20 and Fig. 2.21 show BC(2,3) and BC(4,2), respectively.

BC(r,2), where $N = r^2$, is an interesting special case. It has three columns because $k = 2$ in this case. Fixing the first column to an arbitrary configurarion results in a network that has $k = 2$ columns and is thus a self-routed MIN. This

feature can be exploited to speed up the slow routing control of BC(r,2). This will be the focus of Chapter 8 as mentioned earlier. Fig. 2.21 shows BC(4,2).

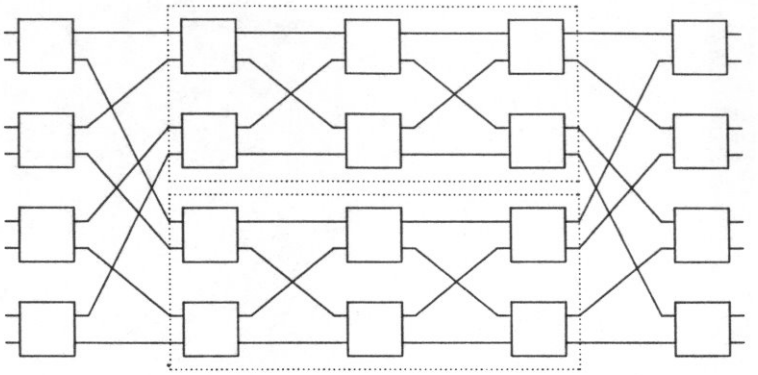
Note that since the switch size r is restricted by technology, BC(r,2)'s are practical only for relatively small N. With the current state-of-the-art technology, N can be up to $1024 = 32^2$.

This completes the overview of multistage interconnection networks, the definition of the entities and concepts that are the building blocks of this dissertation, and the statement of the main subjects that will be treated in the following chapters.



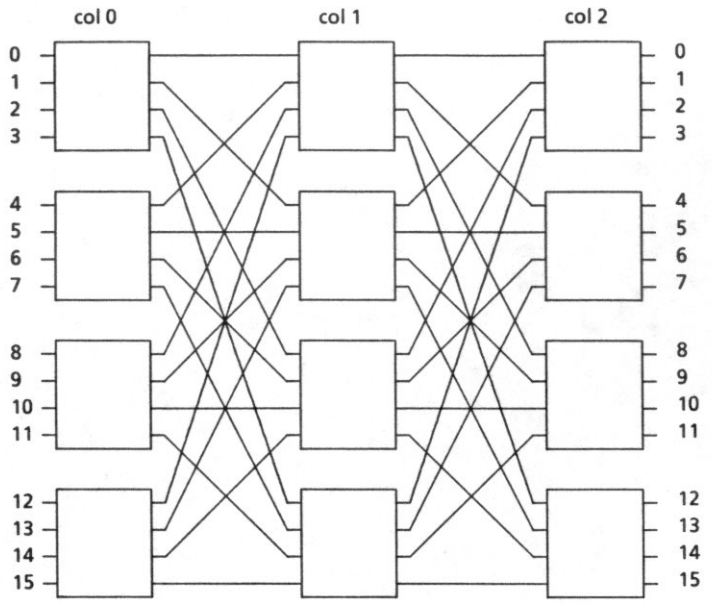
Recursive Construction of BC(r,k)

Fig. 2.19



BC(2,3) -The networks in dashed boxes are BC(2,2)

Fig. 2.20



BC(4,2) = 16x16 3-stage Benes

Fig. 2.21

Chapter 3

Network Equivalence

Multistage interconnection networks, as defined earlier, do not realize all possible permutations. This has prompted work to find procedures to decide network equivalence [ORU85a, b, c]. The algorithms found for conjugate equivalence in [ORU85a] are of complexity $O(N^2 \log N)$ for networks in MIN$(2,k)$ where $N = 2^k$ and $O(N^3 \log^2 N)$ for networks in MIN(r,k) for $r > 2$ and $N = r^k$. This is too costly for large N, and does not achieve the lower bound $\Omega(N \log N)$. Moreover, the algorithms rely on permutational analysis, shedding no light on nor taking any advantage of the relationship between topology and functionality.

In this chapter, the one-to-one correspondence between topology and functionality of networks in MIN(r,k) is shown. Precisely, if two networks realize the same permutations, they are shown to have the same underlying topology. Based on this, an algorithm to decide strong equivalence between two arbitrary MIN's is developed. The algorithm achieves the lower bound. The analysis is carried out for the networks in MIN$(2,k)$ but can be easily generalized to the networks of MIN(r,k) for arbitrary r.

No conjugate or weak equivalence algorithms are developed here, but it is hoped that the one-to-one correspondence between topology and functionality of MIN's will lead to more efficient conjugate and weak equivalence algorithms than the existing ones.

This chapter is organized as follows. Section 3.1 shows the one-to-one correspondence between topology and functionality. In Section 3.2, the strong equivalence algorithm is developed and analyzed. Some concluding remarks are presented in Section 3.3.

3.1 Strong Equivalence and Topological Equivalence

It is somewhat intuitive that the structure of a network and its set of realizable permutations are in one-to-one correspondence. In other terms, if two networks have different structures, they cannot pass the same permutations. The proof of this is the focus of this section.

Two observations can be made. The first is that if the incoming (or outgoing) links of a switch of anetwork are shuffled within the switch (an sws operation), the overall connectivity, and hence the power of the network, does not change

The second observation is that if the switches are permuted within a column without altering the connectivity to and from that column (a pwc operation), then the overall connectivity and power of the network do not change.

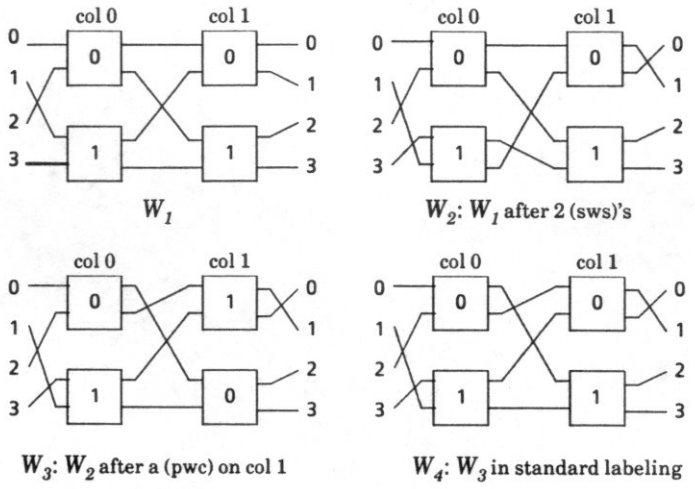
So if a sequence of (sws) and (pwc) operations is applied on a network, the permutation power remains unchanged, that is, the resulting network is strongly equivalent to the original one. Stated otherwise, if W and W' are strictly topologically equivalent, then they are strongly equivalent. Fig. 3.1 shows an example of the effect of sws and pwc operations.

The important fact is that if $W \equiv W'$, then W' can be derived from W by applying a certain sequence of (sws)'s and (pwc)'s on W. This is summarized in the following theorem.

Theorem 3.1: Let W and W' be two networks in IMIN$(2,k,l)$. W and W' are strongly equivalent if and only if W and W' are strictly topologically equivalent.

The proof is surprisingly long and involved.

The if part has been proved in the preceding paragraph. The only-if part involves a series of lemmas and propositions which are proved in Appendix I . The



Effect of sws and pwc Operations

Fig. 3.1

main two facts that are of immediate use in the proof of the theorem are presented in the following three propositions. Note that in numbering the lemmas and theorems in each chapter, those in the corresponding appendix are considered part of the chapter.

Proposition 3.1: Let W and W' be two one-column networks in IMIN($2,k,l$) such that $W \equiv W'$. Then W' can be transformed to W by a sequence of (sws) and (pwc) operations.

Proof: In Appendix I. □

Proposition 3.2: Let W and W' be two networks in IMIN($2,k,l$), where $l \leq k$, such that $W \equiv W'$ and the first stage (- , col 0) of W is identical to that of W'. We can apply a sequence of (sws)'s on the right side of the switches of the leftmost column of W' so that if the first stage (i.e., the leftmost interconnection and column of switches) is deleted from both networks, the resulting networks are strongly equivalent.

Proof: In Appendix I. □

Proposition 3.3: Let W and W' be two networks in IMIN($2,k,l$) such that $W \equiv W'$. If a and b are two input terminals that are linked to one switch x in W', then a and b must both be linked to one switch in W too.

Proof: In Appendix I. □

Proof of the only-if part of theorem 3.1: The proof is by induction on the number of columns l, $1 \leq l \leq k$. The networks of IMIN($2,k,l$) are assumed to have $N = 2^k$ inputs and N outputs, and each of their columns thus has $N/2$ switches.

Basis step: $l = 1$. Let W and W' be two one-column strongly equivalent networks. By Proposition 3.1, W' can be transformed to W by a sequence of (sws)'s and (pwc)'s.

Induction step: Assume that the theorem is true for all values up to l-1. It will be proved true for the value l. Let W and W' be two strongly equivalent networks in MIN($2,k,l$). Both W and W' have l columns each.

Due to Proposition 3.3, it should be clear that we can permute the switches of the leftmost column of W' (i.e., apply pwc), then apply (sws) operations on the left side of that column so that the first stage of W' becomes identical to that of W. Assume that those operations have been done.

By Proposition 3.2, we can apply a sequence of (sws)'s on the right side of the switches of the leftmost column of W' so that if the first stage is deleted from both networks, the resulting networks (say, W_1 *and* W'_1) are strongly equivalent.

By the induction hypothesis W_l' can be transformed to W_1 by applying (sws) and (pwc) operations on the columns of W_l'. Applying the same sequence of operations on the last l-1 columns of W' will then transform it to W.

Hence, W' can be transformed to W by (sws) and (pwc) operations.

Therefore, the theorem is true for all values of l, $1 \leq l \leq k$. In particular, for $l = k$ which corresponds to the class of logN-stage networks MIN($2,k$). □

Corollary 3.1: Let W and W' be in MIN($2,k$). W and W' are weakly equivalent if and only if W and W' are widely topologically equivalent.

Proof: Follows immediately from Theorems 3.1 and the definitions of weak equivalence and weak topological equivalence. □

Theorem 3.1 and Corollary 3.1 show the one-to-one correspondence between functional equivalence and topological equivalence in the strong sense and in the weak sense.

3.2 Strong Equivalence Algorithm

The main idea behind deciding the strong equivalence of two networks W and W' is to recover the sequence of (sws)'s and (pwc)'s that transforms W' to W.

Let $P_i(W) = \{ (a,x,b) \mid x$ is a switch in col i of W, a (resp. b) is the smallest input (resp. output) terminal reachable from $x\}$.

Lemma 3.6: (a,x,b) and (a,y,b) are in $P_i(W) \Rightarrow x = y$

Proof: If $x \neq y$, then there would exist two different paths between a and b in W, one through switch x and the other through switch y, violating the single path property. □

Applying (sws) operations on W does not alter the $P_i(W)$'s, i = 0, 1, ... , k-1. Applying (pwc) operations would permute the switches within each column, and thus permute the middle component of the triplets of $P_i(W)$ but they would not change the other 2 components of the triplets of $P_i(W)$.

Hence, If $W \equiv W$', by Theorem 3.1, W' can be derived from W by applying a sequence of (sws) and (pwc) operations on W. Therefore, $P_i(W')$ can be derived from $P_i(W)$ by permuting the middle components of the elements of $P_i(W)$. In particular, if (a,x,b) is in $P_i(W')$, then there exists a unique y such that (a,y,b) is in $P_i(W)$.

So the heart of the algorithm consists of computing $P_i(W)$ and $P_i(W')$ for all i, then for each switch x of each column i of W', look for the triplet (a,x,b) of $P_i(W')$ that contains x, then look for y such that (a,y,b) is in $P_i(W)$; if no y is found, then W is not strongly equivalent to W' and the algorithm should stop; if y is found (it has to be unique, after Lemma 3.6), relabel x of W' by y. After successfully relabeling all the switches of W', which amounts to applying the appropriate sequence of (pwc) operations, call the resulting network W".

If $W \equiv W$", then W can be derived from W" by applying a list of (sws) operations on W". This can be easily checked if switches are replaced by nodes, because the resulting graphs, designated $G(W)$ and $G(W")$, must be identical.

The algorithm is an implementation of the steps explained above. Efficient computing of $P_i(W)$ for i = 0, 1, ... , k-1 can be done by computing $L_i(W)$ and $R_i(W)$:

$L_i(W) = \{(a,x) \mid x$ is a switch in col i of W, a is the smallest input terminal reachable from $x\}$,

$R_i(W) = \{(x,b) \mid x$ is a switch in col i of W, b is the smallest output terminal reachable from $x\}$.

Note that $P_i(W) = L_i(W) * R_i(W)$, where * is the "join" operator (i.e., $(a,x)*(x,b) = (a,x,b)$).

$L_i(W)$ can be computed from $L_{i-1}(W)$ in O(N) time and space as follows:

For each switch x of column i do

1. Let y and z be the two switches of column i-1 that are connected to x;
2. Find (a_1,y) in $L_{i-1}(W)$;(a_1 is stored in a record identified by y)
3. Find (a_2,z) in $L_{i-1}(W)$;
4. $a = \min(a_1, a_2)$;
5. Store a in the record of x and add (a,x) to $L_i(W)$;

Each of these five steps takes constant time. Hence, $L_i(W)$ takes $O(N)$ time and space.

$L_0(W)$ is computed by taking for each switch x the minimum of the two input terminals connected to it. This also takes $O(N)$ time and space.

Similar procedure can be applied to compute $R_i(W)$ for i = k-1, k-2, ... , 0 in this order by computing $R_{k-1}(W)$ first, then computing $R_i(W)$ from $R_{i+1}(W)$.

The algorithm follows.

Strong-Equivalence(W, W')

1. **for** i = 0 **step** 1 to k-1
   compute $L_i(W)$ and $L_i(W')$;
2. **for** i = k-1 **step** -1 to 0
   compute $R_i(W)$ and $R_i(W')$;
3. **for** i = 0 **step** 1 to k-1
   $P_i(W) = L_i(W) * R_i(W)$;
   $P_i(W') = L_i(W') * R_i(W')$;
4. **for** i = 0 **step** 1 to k-1
   4.1. sort $P_i(W)$ and $P_i(W')$ with respect to the 1st and 3rd components;
   4.2. compare $P_i(W)$ and $P_i(W')$ with respect to the 1st and 3rd components;

   if($P_i(W) \neq P_i(W')$)
      output(W and W' are not strongly equivalent) **and stop**;
   else
      for j = 0 step 1 to N/2 - 1
         relabel the switch of the j-th triplet of the sorted $P_i(W')$ by the switch of the j-th triplet of the sorted $P_i(W)$;
5. Let W'' be the relabeled W';
6. **if** ($G(W) \neq G(W'')$)
      output(W and W' are not strongly equivalent) **and stop**;
   else
      output($W \equiv W'$);

Complexity: Steps 1, 2 and 3 each take $O(N\log N)$ because each loops log$N$ times. Step 4.1 involves radix sorting, which takes $O(N)$ [AHU74]. Step 4.2 takes $O(N)$ because the comparison involves matching items of the same rank. Step 5 is notational. Step 6 takes $O(N\log N)$ because comparing $G(W)$ and $G(W'')$ involves scanning once the two data structures that represent $G(W)$ and $G(W'')$ from left to right, making the comparison along the way. Hence, the algorithm takes $O(N\log N)$ time . The space complexity can be similarly analyzed concluding that it is $O(N\log N)$ too. This complexity achieves the lower bound because the size of the data structure is $O(N\log N)$ (i.e., $O(N\log N)$ switches) and any strong equivalence algorithm has to scan the input at least once.

Generalization: Theorem 3.1 can be easily generalized to networks in MIN(r,k) for arbitrary r, and consequently, the strong equivalence algorithm can be modified to accommodate those networks. The only modifications are in the computation of $L_i(W)$, $L_i(W')$, $R_i(W)$, $R_i(W')$ (Steps 1 and 2 of the algorithm). The modified version of the procedure that computes $L_i(W)$ from $L_{i-1}(W)$ is as follows:

For each switch x of column i do

1. For each switch y_i of column i-1 that are connected to x do

   (comment: There are r such switches, say, $y_1, y_2, \ldots, y_r$)

   1.1. Find (a_i, y_i) in $L_{i-1}(W)$;(a_i is stored in a record identified by y_i)

2. $a = \min(a_1, a_2, \ldots, a_r)$;

3. Store a in the record of x and add (a,x) to $L_i(W)$;

Step 1.1 takes constant time. Therefore, step 1 takes $O(r)$. Step 2 takes also $O(r)$ time. The last step takes constant time. The outer for-loop iterates N/r times because column i has N/r switches. As each iteration takes $O(r)$ time, the overall time to compute $L_i(W)$ from $L_{i-1}(W)$ is $O(r * N/r) = O(N)$.

$L_0(W)$ is computed by taking for each switch x of column 0 the minimum of the r input terminals connected to it. Hence, computing $L_0(W)$ takes also $O(N)$ time.

In a similar fashion, $L_i(W')$, $R_i(W)$, $R_i(W')$ are computed in $O(N)$ time.

The other steps of the strong-Equivalence algorithm need not be changed. Hence, the complexity of the modified algorithm is $O(Nk) = O(N\log_r N)$.

3.3 Conclusion

This chapter has treated strong equivalence of logN-stage networks. It has been shown that two networks are strongly equivalent if and only if one network is a different layout of the other network. This fact led to an $O(N\log N)$ strong equivalence algorithm to decide if two arbitrary logN-stage networks realize the same permutations.

It was also shown that two networks are weakly equivalent if and only if they are widely topologically equivalent. However, it remains an open problem to

determine if that can help develop fast algorithms to decide weak equivalence relations. Further research is needed in that direction.

Chapter 4

Network Covering

Network equivalence relations among networks in MIN(r,k) having the same switch size were the focus of the previous chapter. A natural extension of the equivalence problem is the *covering problem*, that is, the relationship between networks of different switch sizes.

If two NxN MIN's W and W' have different building block sizes, then the number of permutations admissible by one network is generally different from that of the other network. W is said to *functionally cover* W' if the permutations realizable by W' are realizable by W too.

To see better the merit of this problem, consider the following. $\Omega(4,3)$ and $\Omega(8,2)$ are two 64×64 omega networks of different switch size, the switch of the first is 4×4 and that of the second is 8×8. As the number of permutations realizable by a network in MIN(r,k) is $(r!)^{kr^{k-1}}$, the first omega realizes $(4!)^{3\times16} = (13824)^{16}$ permutations, and the second omega realizes $(8!)^{2\times8} = (40320)^{16}$ permutations, a much larger number of permutations than the first. The question is whether the set of permutations realizable by the second omega includes the set of permutations realizable by the first omega. The answer, for this example, is negative. However, $\Omega(8,2)$ does realize all permutations realizable by $\Omega(2,6)$.

To show that $\Omega(8,2)$ does not include the set of permutations of $\Omega(4,3)$, we should find a permutation π of $\{0, 1, \ldots, 63\}$ that maps 0 to 0 and 24 to 5 such that it is realizable by $\Omega(4,3)$ but not by $\Omega(8,2)$. It is enough for our purpose to show that the paths from 0 to 0 and from 24 to 5 do not conflict in $\Omega(4,3)$ but do conflict in $\Omega(8,2)$. Non-conflicting paths in $\Omega(r,k)$ can be characterized in the same way as in [LAW75] as follows:

(C): The two paths $s_{k-1}s_{k-2}\ldots s_0 \rightarrow d_{k-1}d_{k-2}\ldots d_0$ and $s'_{k-1}s'_{k-2}\ldots s'_0 \rightarrow d'_{k-1}d'_{k-2}\ldots d'_0$ do not conflict if and only if there is no i where $d_{k-1}\ldots d_{k-i}s_{k-i-1}\ldots s_0 = d'_{k-1}\ldots d'_{k-i}s'_{k-i-1}\ldots s'_0$.

The paths $0 \rightarrow 0$ and $24 \rightarrow 5$ in $\Omega(4,3)$, expressed in 4-ary as $000 \rightarrow 000$ and $120 \rightarrow 011$, are easy to show that they satisfy (C), and hence they do not conflict in $\Omega(4,3)$. However, these same paths in $\Omega(8,2)$, expressed in 8-ary as $00 \rightarrow 00$ and $30 \rightarrow 05$ ($s_1s_0=00$, $d_1d_0=00$, $s'_1s'_0=00$, $d'_1d'_0=00$), do not satisfy (C) because $d_1s_0 = d'_1s'_0 = 00$ and thus they condlict in $\Omega(8,2)$.

The same question of covering can be raised with regard to other omegas of all sizes and of varying switch sizes, as well as to other existing networks, such as the baseline B(r,k)'s, omega inverse $\Omega^{-1}(r,k)$'s, the genralized cube networks G(r,k)'s, and the indirect cubes I(r,k)'s.

The more general question of determining if and when an arbitrary network in MIN(r,k) functionally covers another arbitrary network in MIN(s,k'), where $r^k=s^{k'}$, is then of interest.

No work has been done in this area, with the exception of a brief mention in [LAW75] that $N\times N$ omega networks whose switch size is a power of two realize all the permutations realizable by $N\times N$ omega networks built out of 2×2 switches.

In this chapter, network covering is studied. The contribution of this chapter is summarized below, but first, the notion of topological covering should be introduced.

If W is in MIN(r,k), where $r = s^l$, and if each switch of W is replaced by a network in MIN(s,l), then the resulting network W' is covered by W because every permutation realizable by W' is clearly realizable by W. W is said to *topologically cover* W'. W is also said to *unfold* to W', and W' is called an *unfolding* of W. Note that there are many unfoldings of W when $r = s^l$. Fig. 4.3 shows the network $\Omega(4,2)$, and Fig. 4.4 shows an unfolding of $\Omega(4,2)$.

It will be shown that if W and W' are two $N\times N$ MIN's of $r\times r$ and $s\times s$ switches as building blocks, respectively, and if W functionally covers W', then r is an integer power of s and W topologically covers W'. Based on this fact, an optimal algorithm, of complexity $O(N\log_s N)$, to decide functional covering, is given. Then, the covering relationships among the existing networks, which were generalized in Section 1.2, are studied. It will be shown that the set of admissible permutations (and not only their cardinality) of each of the following networks, omega, omega inverse, baseline, generalized cube networks, butterfly networks grows inclusively as the switch size is raised to a power. The indirect binary n-cube is shown to lack this property.

The equivalence between functional covering and topological covering is shown in Section 4.1. A network covering algorithm is given and analyzed in Section 4.2. Section 4.3 deals with the covering relationships among existing networks.

4.1 Functional and Topological Covering

If W unfolds to W'^{l}, then W functionally covers W'. In this section, the converse is shown to be true. That is, if W is in IMIN(r,k,l) and W' in IMIN(s,k',t) such that $r^k=s^{k'}$ and W functionally covers W', then r must be a power of s and W must topologically cover W'.

The proof may seem intuitive and straightforward, but it is actually long and involved. To show that W must topologically cover W', we have to prove that the first l columns of W' can be "clustered" to look like a column of blocks of switches such that this column can be said to unfold from the first column of W. This clustering is done by permuting switches within columns. Similarly, each next l columns of W' must be shown to cluster in the same way. In addition, the interconnections between these clustered columns should be proved to be the same as those between the corresponding columns of W.

Showing that the above can be done if W realizes all the permutations of W' is not straightforward, and involves several intermediary steps. We will proceed by induction on l, and follow the approach outlined in the preceding paragraph.

As the proof is somewhat long, it is broken into a series of lemmas and propositions which are only stated here to relieve the reader from tedious details. The interested reader can refer to Appendix II for proofs.

Proposition 4.1: If W is in IMIN(r,k,1) and W' in IMIN(s,k',t) and W functionally covers W', then r is a power of s and W topologically covers W'.

Proof: In Appendix II. □

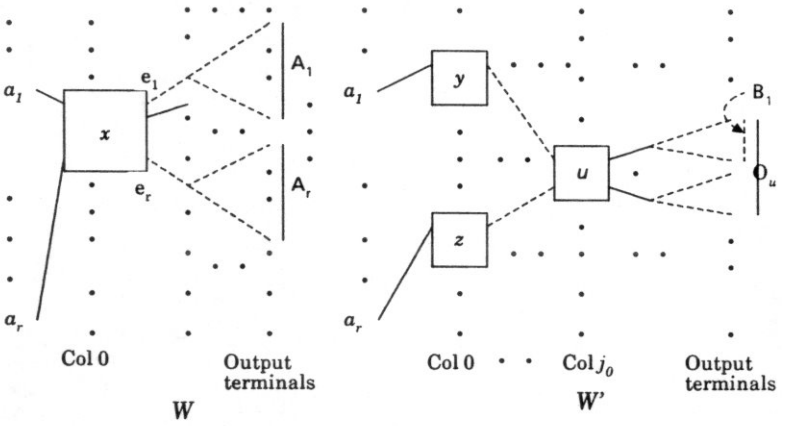


Fig. 4.1

Note that if W is in IMIN(r,k,t) and W' in IMIN(s,k',t') such that W covers W', then $r^t = s^{t'}$ because r^t(resp., $s^{t'}$) is the number of output terminals reachable from an input terminal of W (resp., W'). These two numbers of terminals must be equal because W functionally covers W'. If $N = r^k = s^{k'}$, then $r = s^{k'/k}$ and $r = s^{t'/t}$ and thus $k'/k = t'/t$.

Proposition 4.2: Let W be in IMIN(r,k,t), W' in IMIN(s,k',t') functionally covered by W, $a_1, a_2, \ldots, a_r$ the input terminals linked to a switch x in W, and j_0 the leftmost column in which there is a switch that is reachable from all $a_1, a_2, \ldots, a_r$ in W' (see Fig. 4.1). The switches that can be traced from $a_1, \ldots, a_r$ rightward to col j_0 in W' form an independent subnetwork S in MIN(s,j_0+1).

Proof: In Appendix II. □

Let $m = s^{j_0+1}$, $c_1, c_2, \ldots, c_m$ be the output terminals of S (as defined in Proposition 4.2), and $C_1, C_2, \ldots, C_m$ the sets of output terminals of W' that are reachable from $c_1, c_2, \ldots, c_m$, respectively. Let $A_1, A_2, \ldots, A_r$ be the sets of output terminals of W that are reachable from output ports 1, 2, ..., r of switch x (of Proposition 4.2), respectively; $A = (\cup A_j)_{i=1,\ldots,r}$, and $C = (\cup C_j)_{i=1,\ldots,m}$. Note that if p is in C_i and q is in C_j for $i \neq j$, then the paths $c_i \rightarrow p$ and $c_j \rightarrow q$ do not conflict in W'. A is the set of output terminals reachable from a_1 in W, and C is the set of terminals reachable from a_1 in W'. As W functionally covers W', $A = C$.

Proposition 4.3: For every $j = 1, 2, \ldots, m$, there is an i in {1,2, ..., r} such that $C_j = A_i$. Moreover, $r = m = s^{j_0+1}$.

Proof: In Appendix II. □

Network S of Proposition 4.2 is then in MIN(s,l), and has $r = s^l$ inputs ($a_1, \ldots, a_r$) and r outputs. As the switch x of col 0 of W is arbitrary, we conclude the following proposition.

Proposition 4.4: Every switch i in col 0 of W corresponds to a subnetwork S_i in MIN(s,l) embedded in the first j_0+1 columns of W' and having the same inputs as switch i of W. In other words, the first j_0+1 columns of W' form N/r independent subnetworks in MIN(s,l). Moreover, if $A_{i1}, A_{i2}, \ldots, A_{ir}$ (resp., $C_{i1}, C_{i2}, \ldots, C_{ir}$) are the sets of output terminals reachable from the output ports $a_{i1}, a_{i2}, \ldots, a_{ir}$ (resp., $c_{i1}, c_{i2}, \ldots, c_{ir}$) of switch i (resp., subnetwork S_i) in W (resp., W'), then $\{A_{i1}, A_{i2}, \ldots, A_{ir}\} = \{C_{i1}, C_{i2}, \ldots, C_{ir}\}$ (by Proposition 4.3).

Cluster the switches of the first j_0+1 columns of W' to form a column of N/r subnetworks S_i's ordered $S_1, S_2, \ldots, S_{N/r}$ from top to bottom. Then $\{a_{i1}, a_{i2}, \ldots, a_{ir}\} = \{c_{i1}, c_{i2}, \ldots, c_{ir}\}$ for all i. Relabel each c_{ij} by a_{il} where $C_{ij} = A_{il}$ (such l corresponding to j exists after Proposition 4.3). After the clustering and relabeling, it is easy to see that for each i, switch i of W and subnetwork S_i of W' have the same input port labels and same output port labels and the output terminals reachable from any output port of switch i in W are the same as those reachable from the same output port of S_i in W'. We thus have proved the following.

Proposition 4.5: If we delete the first column of W and the subnetworks $S_1, S_2, \ldots, S_{N/r}$ of W', the resulting networks (say W_1, W'_1) are in IMIN(r,k,t-1) and IMIN(s,k',t'-j_0-1), respectively, and W_1 functionally covers W'_1.

Theorem 1: If W is in IMIN(r,k,t) and W' in IMIN(s,k',t') such that W functionally covers W', then r is a power of s and W topologically covers W' (i.e., W unfolds to W').

Proof: By induction on t.

Basis: $t = 1$. By Proposition 4.1.

Induction: Assume the theorem is true for all values up to $t\text{-}1 \geq 1$, and it will be shown for t. By Proposition 4.3, r is a power of s. By Proposition 4.4, the first column of W unfolds to the the first $\log_s r$ columns of W'. After appropriate clustering and relabeling of the ports of col $(\log_s r - 1)$ of W', and after deleting the first column of W and the first $\log_s r$ columns of W', the resulting networks W_1 and W'_1 are in IMIN$(r,k,t\text{-}1)$ and IMIN$(s,k',t'\text{-}j_0\text{-}1)$, respectively, and W_1 functionally covers W'_1 (by Proposition 4.3). The inductive hypothesis can now be applied on W_1 and W'_1. Hence, W_1 unfolds to W'_1, and all in all, W unfolds to W'. Thus, W topologically covers W'. □

It follows then that if W is in MIN(r,k) and W' in MIN(s,k'), then W functionally covers W' if and only if r is a power of s and W topologically covers W'.

As mentioned earlier, the equivalence between functional covering and topological covering is a generalization of the one-to-one correspondence between the topology and the functionality of MIN's. In fact, a network W in MIN(r,k) is strongly equivalent to another network W' in MIN(r,k) iff each one functionally covers the other and therefore iff each one topologically covers the other, which amounts to saying that the two have the same topology.

Henceforth, we will use the term "cover" without qualification since functional covering and topological covering are equivalent.

Based on this equivalence, an optimal algorithm to decide network covering is given in the next section.

4.2 Network Covering Algorithm

In this section, we present an algorithm that takes two networks W in MIN(r,k) and W' in MIN(s,k') as input and decides if W covers W'. The algorithm

checks first if r is a power of s and halts if not. Otherwise, it breaks the columns of W' into r consecutive *bands* of $l = \log_s r$ columns each, where the i-th band consists of columns $i\times l$, $i\times l+1$, ... , $(i+1)\times l - 1$. Afterwards, it examines each band to see whether it consists of N/r independent subnetworks in MIN(s,l). If one of the bands does not, the algorithm halts answering negatively; otherwise, it replaces each subnetwork by an $r\times r$ switch (i.e., "fold" each subnetwork into a switch). The links coming to and going from these $r\times r$ switches are the same links coming to the leftmost column and going from the rightmost column of the corresponding subnetwork. Finally, the algorithm checks if the folded network (say W'') is strongly equivalent to W, by calling the strong equivalence algorithm of the preceding chapter. W is strongly equivalent to W'' if and only if W covers W'.

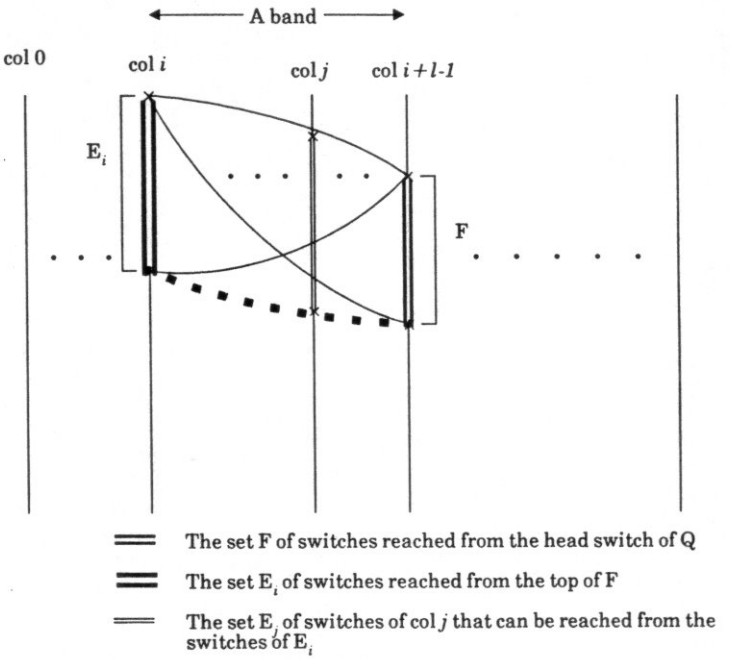
Note that there are many different foldings of a network W' if W' is an unfolding of some network W. The difference lies in the ordering of the switches of the left and rightmost columns of each subnetwork in W'. This leads to different ordering in the incoming and outgoing links to each corresponding folding switch in the folding networks. However, this difference does not affect the functionality of the folding networks. That is, all the foldings of W' are strongly equivalent. Therefore, it suffices to find one arbitrary folding of W' and check its strong equivalence with W.

It remains to see how to decide the breakability of bands into subnetworks in MIN(r,l). The main idea is to start with the top leftmost switch of the band, trace the tree of switches to rightmost column of the band, and denote by F the set of the r/s switch-leaves. Then trace backward starting from the first switch of F, up to the leftmost column of the band, denoting by E the set of switches reached in that column. Afterwards, march rightward from the switches of E. If the number of the reached switches in any intermediate column is greater than r/s, the procedure halts the overall algorithm answering that W does not cover W' (because otherwise these numbers must be r/s in order for the switches swept in the march to form an

independent subnetwork in MIN(s,l), after Lemma 4.2 of Appendix II). If all these numbers of swept switches in each of the column of the band are equal to r/s, then these switches form an independent network in MIN(s,l) and are replaced by an $r \times r$ switch. The same cycle is repeated on the remaining switches of the band. This procedure is called FOLD-BAND and is summarized below.

FOLD-BAND(i) **(comment**: col i is the leftmost column of the band)
begin
1 Let Q be the set of switches in col i and $l = \log_s r$
2 **While** ($Q \neq \emptyset$)
begin
(comment: The following 3 steps cluster the switches that may form an $r \times r$ network in MIN(s,l). This is illustrated in Fig. 4.2)
3 trace the switches from the top switch of Q down to col $i+l-1$;
(comment: in depth first search)
4 let F be the set of traced switches in col $i+l-1$; **(comment**: $|F| = r/s$)
5 trace the switches from the top switch of F back to col i;
6 let E_i be the set of traced switches in col i; **(comment**: $|E_i| = r/s$)
7 **for** $j = i+1$ **to** $i+l-1$ **step 1 do**
(commenet: This **for** loop checks if the above culstered switches form an $r \times r$ independent network in MIN(s,l))
begin
8 let E_j be the set of switches in col j that are reachable from switches in E_{j-1};
9 **if** ($|E_j| \neq r/s$)
output (W does not cover W') and **exit**;
end
10 replace the switches of ($E_i, E_{i+1}, \ldots, E_{i+l-1} = F$) by an $r \times r$ switch;

delete E_i from Q;
11 **end**
end



Switch Clustering

Fig. 4.2

Complexity of FOLD-BAND: The while-loop repeats at most $N/(r/s)$ times. The depth first search in Steps 3 and 5 takes $O(r/s)$. Steps 8 and 9 can be combined to keep

track of the cardinality of E_j as it is being computed and exit if and when this cardinality skips r/s. Thus, the for-loop takes $O((r/s)\log_s r)$. Therefore, the procedure takes $O(N\log_s r)$.

The overall covering algorithm is presented next.

NETWORK-COVERING(W,W') (**comment**: W in MIN(r,k) and W' in MIN(s,k'))
begin
1 **if**(r is not a power of s)
  output (W does not cover W') and **exit**;
2 **for** i = 0 **to** k-2 **step** 1 (**comment**: $k = \log_r N$)
3 FOLD-BAND($i \times \log_s r$);
  let W'' be the folded W'; (**comment**: at this point, W' is tested to be foldable and has been folded into W'' by the previous for-loop)
4 **call** (STRONG-EQUIVALENCE(W,W'');
  if ($W \equiv W''$)
    output(W covers W');
  else
    output(W does not cover W');
end

Complexity of NETWORK-COVERING: Step 1 can be done in $O(\log_s r)$. The for-loop in Step 2 takes $O(kN\log_s r) = O(N\log_s r \log_r N) = O(N\log_s N)$. Step 4 takes $O(N\log_r N)$ which is dominated by $O(N\log_s N)$. Consequently, the overall complexity of the covering algorithm is $O(N\log_s N)$. This is the lower bound up to a constant factor because the size of input W' is $O(N\log_s N)$.

4.3 Covering Relationships among Existing Networks

This section explores the covering relationships among networks within each class of networks defined in Section 2.2, namely, omega networks $\Omega(r,k)$, omega inverse $\Omega^{-1}(r,k)$, baseline networks $B(r,k)$, generalized cube networks $G(r,k)$, butterfly networks $BF(r,k)$, and indirect r-ary n-cube networks $I(r,k)$.

The approach to proving that $W(r,k)$ covers $W(s,k\times l)$ for $W = \Omega, \Omega^{-1}$, B, G, BF is to replace each $r\times r$ switch by the network $W(s,l)$ (the same network but of different parameters s and l, where $r = s^l$), then permute (i.e., reposition) the $s\times s$ switches within each column of the resulting network in a specified manner, then show that the final resulting network is $W(r,k)$. Note that when a switch is repositioned, the links attached to it remain attached. Therefore, such repositioning of switches (i.e., pwc operations) does not alter the functionality of the network, and thus, the three main steps of this approach are enough to show that $W(r,k)$ covers $W(s,k')$ where $r^k = s^{k'}$ and r is power of s.

If a column of a network $W(s,k')$ is permuted by a permutation f (i.e., switch i is moved to position $f(i)$), then the input and output ports of the column are permuted by a permutation g, denoted $R(f)$, such that $g(y_{k'-1}y_{k'-2}\cdots y_0) = f(y_{k'-1}y_{k'-2}\cdots y_1)y_0$.

The following lemma is straightforward and will prove useful throughout.

Lemma 4.7: if (col i, col $i+1$) = h and if col i is permuted by f_1 and col $i+1$ by f_2, then (col i, col $i+1$) becomes equal to $g_1^{-1} h\, g_2$, where $g_1 = R(f_1)$ and $g_2 = R(f_2)$.

Theorem 4.2: $\Omega(r,k)$ covers $\Omega(s,k')$, where $r = s^l$ and $k' = kl$.

See Fig. 4.3, Fig. 4.3 and Fig. 4.5 for illustration.

Proof: Step 1: Replace each $r\times r$ switch of $\Omega(r,k)$ by $\Omega(s,l)$ and denote by W' the resulting network. W' has $kl = k'$ columns. $(\text{col } jl+i, \text{col } jl+i+1)_{W'} = S_{N,s,k'-l}$ for j =

$0, 1, \ldots, k\text{-}1$ and $i = 0, 1, 2, \ldots, l\text{-}2$, because that stage is the vertical concatenation of the stages $(\text{col i}, \text{col i}+1)_{\Omega(s,l)}$ = shuffle of a segment of length $r = s^l = N/s^{k'-l}$.

$(\text{-}, \text{col } 0)_{W'} = (\text{col } jl\text{-}1, \text{col } jl)_{W'} = S_{N,r,0} S_{N,s,k'-l}$, for $j = 1, 2, \ldots, k\text{-}1$.

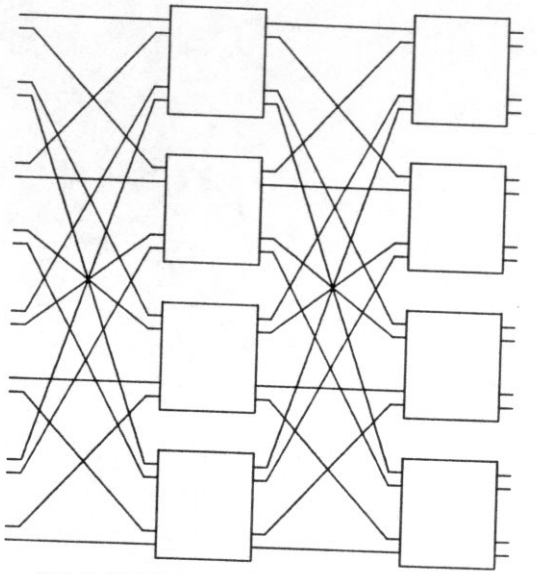
Step 2: Permute col $jl+i$ of W' by f_{jl+i} such that $f_{jl+i}(y_{k'-1}y_{k'-2} \cdots y_1) = y_{l-1} \cdots y_{i+1} y_{k'-1} \cdots y_k y_i \cdots y_1$, for $j = 0, 1, \ldots, k\text{-}1$ and $i = 0, 1, \ldots, l\text{-}1$. Let W'' denote the resulting network, and $g_{jl+i} = R(f_{jl+i})$. Note that f_{jl+i} is independent of j, and $f_{jl+l-1} = e$ and hence $g_{jl+l-1} = e$.

W'' will be shown to be $\Omega(s,k')$. For $j = 0, 1, \ldots, k\text{-}1$ and $i = 1, 2, \ldots, l\text{-}2$, $(\text{col } jl+i, \text{col } jl+i+1)_{W''} = g^{-1}_{jl+i} S_{N,s,k'-l} g_{jl+i+1}$ by Lemma 4.7. This is shown to be $S_{N,s,0}$ next.

$$
\begin{aligned}
(y_{k'-1}y_{k'-2} \cdots y_0) g^{-1}_{jl+i} S_{N,s,k'-l} g_{jl+i+1} &= (y_{k'-l+i} \cdots y_{i+1} y_{k'-1} \cdots y_{k'-l+i+1} y_i \cdots y_0) S_{N,s,k'-l} g_{jl+i+1} \\
&= (y_{k'-l+i} \cdots y_{i+1} y_{k'-2} \cdots y_{k'-l+i+1} y_i \cdots y_0 y_{k'-1}) g_{jl+i+1} \\
&= y_{k'-2} \cdots y_{k'-l+i+1} y_{k'-l+i} \cdots y_{i+1} y_i \cdots y_0 y_{k'-1} \\
&= y_{k'-2} \cdots y_0 y_{k'-1} = S_{N,s,0} (y_{k'-1} y_{k'-2} \cdots y_0).
\end{aligned}
$$

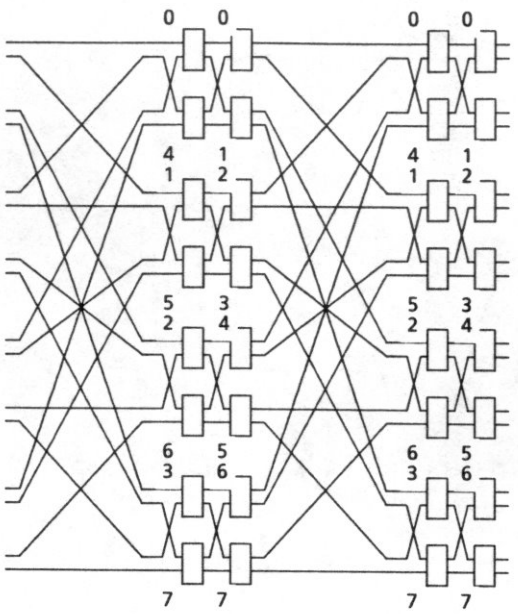
$(\text{-}, \text{col } 0)_{W''} = (\text{col } jl+l\text{-}1, \text{col } (j+1)l)_{W''}$ because $g_{jl+l-1} = e$. This is $S_{N,r,0} S_{N,s,k'-l} g_0$. This permutation can be similarly shown to be $S_{N,s,0}$.

Consequently, all the stages of W'' are the same as those of $\Omega(s,k')$, and hence $W'' = \Omega(s,k')$. Thus, $\Omega(r,k)$ covers $\Omega(s,k')$. □



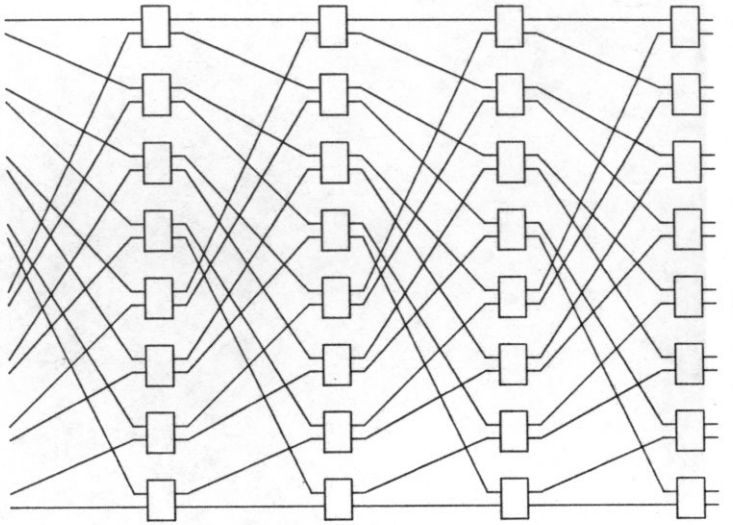
This is $\Omega(4,2)$. It is folded into a network in Fig. 4.4 by replacing each switch by $\Omega(2,2)$.

Fig. 4.3



The folded $\Omega(4,2)$. Each 4×4 switch is replaced by $\Omega(2,2)$. The labels are the new labels of the switches to get $\Omega(2,4)$. Fig. 4.5 shows the same network where each switch is put in the position of its label, and which is clearly $\Omega(2,4)$.

Fig. 4.4



This is the network of Fig. 4.4 except that the switches are permuted (repositioned) according to the labels shown in Fig. 4.4 so that each switch is in its normal positon agreeing with its label. Clearly, it is $\Omega(2,4)$.

Fig. 4.5

Theorem 4.3: $\Omega^{-1}(r,k)$ covers $\Omega^{-1}(s,k')$.

Proof: It is obvious that if a network covers another, the inverse of the first covers the inverse of the second. □

It can be easily shown from subsection 2.2.2 that (col 1, col 2, ... , col k-1)$_{B(r,k)}$ is a vertical concatenation of r B(r,k-1) networks (see Fig. 2.9).

Theorem 4.4: B(r,k) covers B(s,k').

Proof: By induction on k.

Basis: $k = 1$. Then $N = r = s^l$ and therefore B(r,k) = B(r,1) is just an $r \times r$ cross bar switch which obviously covers B(s,k') = B(s,l).

Induction: Assume the theorem is true for all values up to k-1, and it will be shown true for the value k. (col 1, col 2, ... , col k-1)$_{B(r,k)}$ is a vertical concatenation of r B(r,k-1) networks. As $N/r = r^{k-1}$, the inductive hypothesis applies to the r B(r,k-1) networks and are therefore folded to B(s,k'-l). Replace each switch of col 0 of B(r,k) by B(s,l), and let W' denote the resulting network. (col i, col i+1)$_{W'} = U_{N,s,k'-l}$ for i = 0,1, ... , l-2. (col l-1, col l)$_{W'} = U_{n,r,0}$.

Permute col i of W' by f_i such that $f_i(y_{k'-1}y_{k'-2} \cdots y_1) = y_{l-i}y_{l-i+1} \cdots y_{l-1}y_{k'-1} \cdots y_0y_{l-i-1} \cdots y_1$ for i = 1, 2, ... , l-1. Col l, col l+1, ... , col k'-1 are permuted as follows. Assume the r B(s,k'-l) networks of (col l, col l+1, ... , col k'-1)$_{W'}$ are labeled 0,1, ... , r-1 and their labels are in s-ary representation. Permute these networks by t such that $t(y_{l-1} \cdots y_0) = y_1y_2 \cdots y_{l-1}y_0$. This amounts to permuting each of col l, col l+1, ... , col k'-1 of W' by f such that $f(y_{k'-1}y_{k'-2} \cdots y_1) = y_{k'-l+1}y_{k'-l+2} \cdots y_{k'-1}y_{k'-l}y_{k'-l-1} \cdots y_1$. Let $g_i = R(f_i)$ and $g = R(f)$. Let W'' denote the resulting network.

$g_i(y_{k'-1}y_{k'-2} \cdots y_0) = y_{l-i}y_{l-i+1} \cdots y_{l-1}y_{k'-1} \cdots y_0y_{l-i-1} \cdots y_0$.

By Lemma 4.7, (col i, col i+1)$_{W''} = g^{-1}{}_i U_{N,s,k'-l}g_{i+1}$ which will be shown to be $U_{N,s,i}$ for i = 1, ... , l-2. (col 0, col 1)$_{W''} = U_{N,s,k'-l}g_1$ because col 0 of W' is not permuted. (col l-1, col l) = $g^{-1}{}_{l-1} U_{N,r,0}\, g$. Both $U_{N,s,k'-l}g_1$ and $g^{-1}{}_{l-1} U_{N,r,0}\, g$ will also be shown to be $U_{N,s,0}$ and $U_{N,s,l-1}$, respectively.

$$(y_{k'-1}y_{k'-2} \cdots y_0)g^{-1}{}_i U_{N,s,k'-l}g_{i+1} = (y_{k'-i-1} \cdots y_{l-i}y_{k'-i} \cdots y_{k'-1}y_{l-i-1} \cdots y_0)U_{N,s,k'-l}g_{i+1}$$
$$= (y_{k'-i-1}y_{k'-i-2} \cdots y_{l-i}y_0y_{k'-i}y_{k'-i+1} \cdots y_{k'-1}y_{l-i-1} \cdots y_1)g_{i+1}$$
$$= y_{k'-1} \cdots y_{k'-i+1}y_{k'-i}y_0y_{k'-i-1}y_{k'-i-2} \cdots y_{l-i}y_{l-i-1} \cdots y_1$$
$$= y_{k'-1} \cdots y_{k'-i+1}y_{k'-i}y_0y_{k'-i-1}y_{k'-i-2} \cdots y_1$$
$$= (y_{k'-1}y_{k'-2} \cdots y_0)U_{N,s,i}$$

Hence, $g^{-1}{}_i U_{N,s,k'-l}g_{i+1} = U_{N,s,i}$ for $i = 0,1, \ldots , l$-2.

$$(y_{k'-1}y_{k'-2} \cdots y_0)U_{N,s,k'-l}g_1 = (y_{k'-1}y_{k'-2} \cdots y_iy_0y_{l-1} \cdots y_1)g_1$$
$$= y_0y_{k'-1}y_{k'-2} \cdots y_iy_{l-1} \cdots y_1 = (y_{k'-1}y_{k'-2} \cdots y_0)U_{N,s,0}.$$

$$(y_{k'-1}y_{k'-2} \cdots y_0)g^{-1}{}_{l-1} U_{N,r,0}\, g = (y_{k'-l}y_{k'-l-1} \cdots y_1y_{k'-l+1}y_{k'-l+2} \cdots y_{k'-1}y_0)U_{N,r,0}\, g$$
$$= (y_{k'-l+1}y_{k'-l+2} \cdots y_{k'-1}y_0y_{k'-l}y_{k'-l-1} \cdots y_1)g$$
$$= y_{k'-1} \cdots y_{k'-l+1}y_0y_{k'-l}y_{k'-l-1} \cdots y_1 = (y_{k'-1}y_{k'-2} \cdots y_0)U_{N,s,l-1}.$$

Therefore, (col 0, col 1, ... , col l)$_{W''}$ coincide with (col 0, col 1, ... , col l)$_{B(s,k')}$ and (col l, col l, col l+1, ... , col k'-1)$_{W''}$ are a vertical concatenation of r B(s,l) networks. It follows then that W'' is B(s,k'). □

Proposition 4.6: If col i of $\Omega(r,k)$ is permuted by $(S_{N/r,r,0})^{k-i-1}$ for i = 0, 1, ... , k-1, then the resulting network is G(r,k), and hence $\Omega(r,k) \equiv$ G(r,k).

Proof: The proof is straightforward using Lemma 4.7, and will not be presented. □

Theorem 4.5: G(r,k) covers G(s,k').

Proof: Follows immediately from the previous proposition and Theorem 4.2. □

Proposition 4.7: BF(r,k) = $G^{-1}(r,k)$.

Proof: $(\text{-},\text{col } 0)_{BF} = e = (\text{col } k\text{-}1,\text{-})_G$, $(\text{-},\text{col } 0)_G = S_{N,r,0} = U^{-1}{}_{N,r,0} = (\text{col } k\text{-}1,\text{-})^{-1}{}_G$, and $(\text{col } i, \text{col } i+1)_{BF} = g_i = h_{k\text{-}i\text{-}2} = h^{-1}{}_{k\text{-}i\text{-}2}$ because it is symmetric. Thus, $(\text{col } i, \text{col } i+1)_{BF} = (\text{col } k\text{-}i\text{-}2, \text{col } k\text{-}i\text{-}1)^{-1}{}_G$, and consequently, $BF(r,k) = G^{-1}(r,k)$. □

Theorem 4.6: $BF(r,k)$ covers $BF(s,k')$.

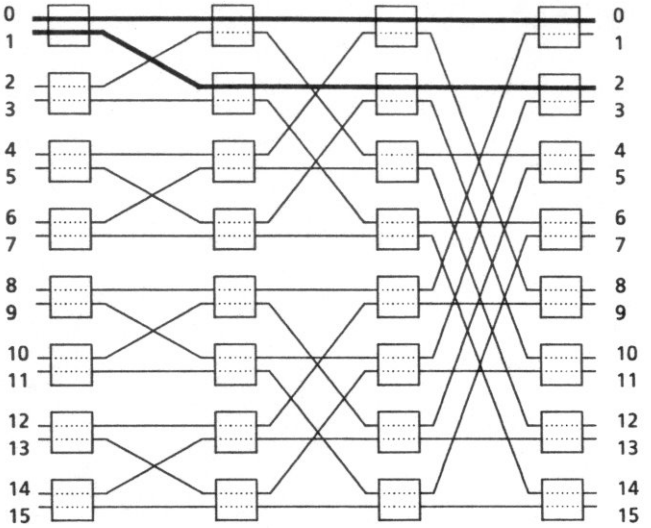
Proof: It follows from Proposition 4.7 and Theorem 4.5. □

Theorem 4.7: $I(r,k)$ *does not* cover $I(s,k')$ even though r is a power of s.

Proof: This is shown by a counter example. Take I(4,2) and I(2,4) (i.e., $r = 4$, $k = 2$, $r' = 2$ and $k' = 4$). It will be shown that I(4,2) does not cover I(2,4) even though 4 is a power of 2. Take the perfect shuffle on four-bit binary numbers:

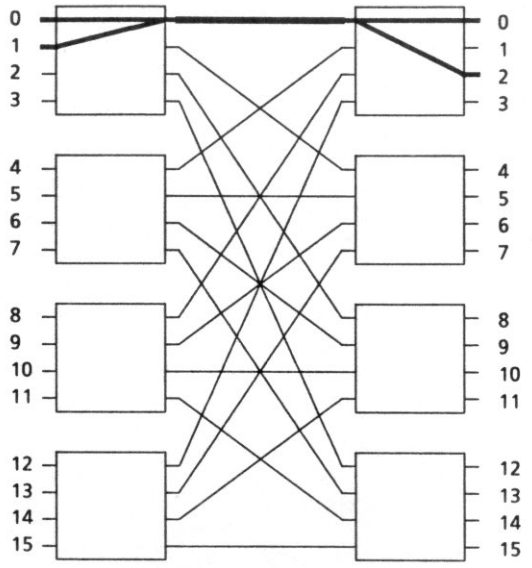
$$S(x_3x_2x_1x_0) = x_2x_1x_0x_3.$$

I(2,4) realizes S by setting all the switches to straightthrough (Fig. 4.6), but I(4,2) cannot realize S because the paths from 0 to 0 and from 1 to 2 conflict as shown in Fig. 4.7. □



I(2,4)
The configuartion of the perfect shuffle

Fig. 4.6



I(4,2)
The dark paths from 0 to 0 and from 1 to 2 conflict over the extra dark link. Note that these two paths do not conflict in I(2,4).

Fig. 4.7

A network $W(r,k)$ is said to be *parameterized* if its interconnections (col i, col $i+1$) are parameterized permutations for all i. Ω, Ω^{-1}, G, B, BF, I are all parameterized. A parameterized network $W(r,k)$ is said to be *incremental* if $W(r,k)$ covers $W(s,k')$ for all N, r and s such that $N = r^k = s^{k'}$ and $r = s^l$. An incremental network W has the property that if its switch size is "upgraded", that is, raised to a power, the upgraded network covers the original network.

It has been shown in this section that Ω, Ω^{-1}, G, B and BF are incremental but I is not.

Chapter 5

Structure of Easy-to-control Networks

The control efficiency of multistage interconnection networks depends on the speed of control tag computation. If the control tags (CT) depend on the sources and the destinations, they must be computed and stored. The resulting memory cost is $N^2 log_2 N$ bits for $N \times N$ networks, which is a prohibitive cost for large systems. For instance, systems that have a number of terminals (i.e., processors) greater than 2^{16} require more than 64 G bits of memory to store the control tags. Therefore, the networks whose CT's are easy to compute and need not be stored are of special interest.

The D-control and FD-control schemes were defined in Chapter 2. Recall that in D-control the control tag between a source and a destination is just the destination address, and in FD-control the control tag is a function of the destination address. D-controllable networks are the most efficiently controllable, and FD-controllable networks can be nearly as efficiently controllable if their functions of control tags are efficiently computable.

As pointed out in Chapter 2, no research work has yet attempted to make a control categorization of networks, for the focus in the literature has been on individual, proposed networks. Such categorization and the study of the underlying structure of the networks in each category would be beneficial and would lead to a better understanding of these networks and their interlationships. As will become clear later, one contribution of the categorization and study done in the present and next chapter is tying together and superseding the different approaches and results in [SIE78], [SIE79], [WU80a], and [SIE81] related to the existing multistage interconnection networks, their underlying structure, their control and their equivalence to one another.

In this chapter, the structure of D-controllable and FD-controllable networks is studied and optimal algorithms to decide D- & FD-controllability are devised. Also, the subclass of doubly controllable networks, which are D- or -FD-controllable from left to right and from right to left, is investigated. The feature double controllability is important for two-way communication in shared memory systems, and has a particular theoretical merit that becomes clear when one observes that all existing MIN's happen to be doubly controllable. It will be shown in this chapter that all doubly controllable networks are equivalent to the baseline network, and in the next chapter, a special class of networks, called digit-permutation networks, will be introduced and shown to be doubly controllable and consequently equivalent to the baseline.

This chapter is organized as follows. In Section 5.1 the structure of D-controllable networks is explored. Section 5.2 studies the structure of FD-controllable networks. Algorithms to decide D- & FD-controllability are given in Section 5.3. Section 5.4 deals with doubly controllable networks.

The analysis is restricted to the networks in MIN($2,k$) for the sake of simplicity, but it is straightforward to extend the results shown here to the class MIN(r,k) for arbitrary r.

5.1 Structure of D-Controllable Networks

A subclass of logN-stage networks of size $N = 2^k$, called Generalized Recursive Networks (GRN), will be defined recursively, and it will be shown that every D-controllable network is in GRN and conversely.

Definition 5.1: GRN_N will be defined recursively.

a- $k=1$: GRN_2 has only two networks (Fig. 5.1-a).

b- $k>1$: GRN_N is the class of networks (denoted $W=T(W_1,W_2)$, Fig. 5.1-b) of $k = \log N$ columns of switches, where T is the connection (i.e., permutation) between col 0

and col *1* of W, W_1 and W_2 are in $GRN_{N/2}$. W_1 and W_2, one on top of the other, make the k-*1* rightmost columns of W. T links the upper output ports of the leftmost column to the inputs of W_1, and the lower input ports to the inputs of W_2. Alternately, $0 \le T(2i) \le N/2-1$, and $N/2 \le T \le (2i+1) \le N-1$.

c- *Extra left interconnection*: If W in GRN_N as in **b** and f is a permutation of S_N, then fW is also in GRN_N (Fig. 5.1-c).

If W is in GRN, and is the switches are permuted within columns (i.e., pwc operations), the resulting network is still considered to be in GRN. To distinguish between the two forms, we call the first form (as shown in Fig. 5.2-a) *canonical*, and the second (Fig. 5.2-b) *non-canonical*.

Note that the above definition of GRN is for networks composed of 2×2 switches. GRN can be generalized to the class of networks composed of $r\times r$ switches for arbitrary r as follows. $r \times r$ networks are mere $r\times r$ switches. Inductively, $N\times N$ networks for $N = r^k$ are of the form $W = T(W_1, W_2, \ldots, W_r)$ where each of the W_i is an $N/r\times N/r$ network of GRN composed of $r\times r$ switches, and T is derived by linking the relative i-th output ports of the switches of the leftmost column to the inputs of W_i, for $i = 0, 1, \ldots, r-1$. W can have any arbitrary interconnection to its left end.

The size of the subclass of GRN (for arbitrary r) can be easily derived by its recursive structure. From the construction of the first stage T it follows that the r^{k-1} inputs of W_i are linked to the r^{k-1} i-th relative output ports of the switches of the leftmost column in one-to-one fashion. The number of such ways of interconnecting is then $r^{k-1}!$. Thus the number of possible T's is $(r^{k-1}!)^r$.

By induction, it can now be shown that the number of GRN networks (without left interconnections) is

$$(r^{k-1}!)r(r^{k-2}!)r^2(r^{k-3}!)r^3\ldots(r!)r^{k-1}$$

Therefore, the number of networks in GRN with left interconnections is the above number multiplied by $r^k!$ (which is the number of possible left internnections). The size of GRN is then

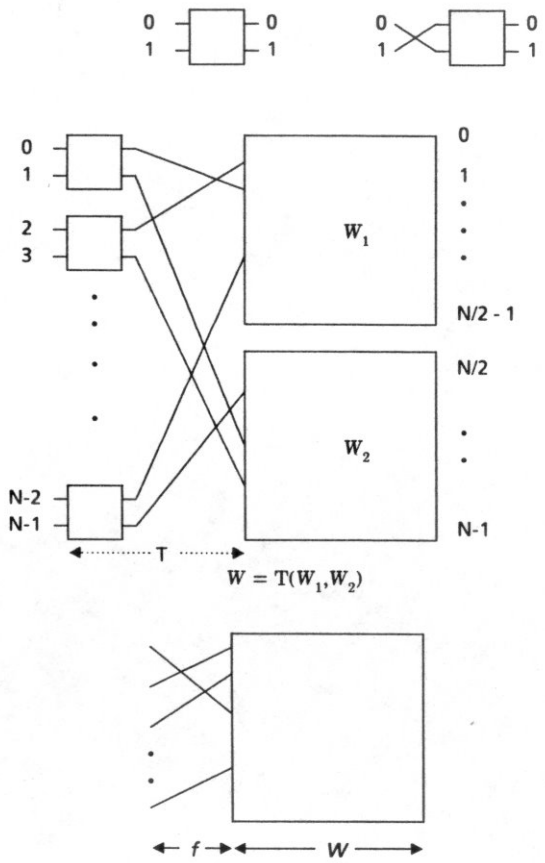
$$\prod_{1\le i\le k}(r^i!)r^{k-i}$$

The analysis and results of this chapter will be mainly related to networks in MIN(2,k) (unless stated otherwise), but can be extended easily to networks in MIN(r,k) now that GRN is defined for arbitrary r.

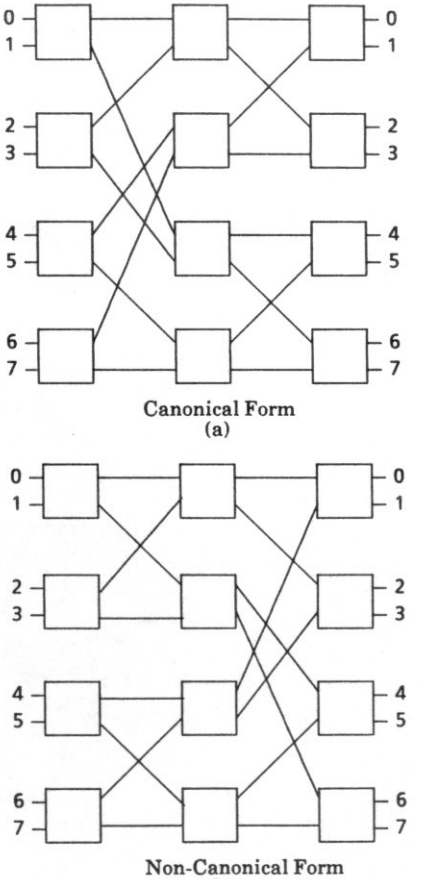
Next, it will be shown that every D-controllable network is in GRN and vice versa.

Proposition 5.1: If W is in GRN, then W is D-controllable.

Proof: Let $W = fT(W_1,W_2)$ be in GRN. Suppose the path $i \to j$ is to be established. Let $j_{k-1}j_{k-2}\cdots j_0$ be the binary representation of j. Using $j_{k-1}j_{k-2}\cdots j_0$ as the control tag will be shown to establish the path. As j_{k-1} is used to control the leftmost column, if j_{k-1} is 0, the incoming input from i will go to the upper half W_1 (where output j is), and if j_{k-1} is 1, the incoming input from i will go to the lower half W_2 (where output j is when $j_{k-1} = 1$). Working similarly in the half network (W_1 or W_2) and recursively as each half is in GRN, the path ends up at output j. □



GRN-Structure
Fig. 5.1



Canonical Form
(a)

Non-Canonical Form
(b)

Fig. 5.2

The proof of the converse of this proposition is somewhat intuitive, yet it is long and several lemmas will be needed.

Lemma 5.1: If W is D-controllable and f is the right connection of W, then for every $j \leq N/2 - 1$, the output terminals $2j$ and $2j+1$ are both linked to one switch s_j; moreover, $2j$ (resp. $2j+1$) is linked to the upper (resp. lower) output port of s_j.

Proof: Fix $j = j_{k-1}j_{k-2}\cdots j_1$. Then $2j = j_{k-1}j_{k-2}\cdots j_1 0$ and $2j+1 = j_{k-1}j_{k-2}\cdots j_1 1$. The two paths from an arbitrary source to the destinations $2j$ and $2j+1$ are identical up to the rightmost column because the $k-1$ leftmost bits of $2j$ and $2j+1$ are identical. Thus, $2j$ and $2j+1$ are linked to the same switch in the rightmost column. Clearly, $2j$ is linked to the upper output port of that switch because the rightmost bit of $2j$ is 0. Similarly, $2j+1$ is linked to the lower output port. □

Lemma 5.2: If W is D-controllable, then the switches of its rightmost column can be repositioned (i.e., relabeled) by a (pwc) operation in such a way that output terminals $2j$ and $2j+1$ are linked to the upper and lower output ports of switch j, respectively.

Proof: Follows immediately from the previous lemma. □

Lemma 5.3: If W is a logN-stage network of size $N = 2^k$ and s a switch in col i of W, then the number of output terminals reachable from s is 2^{k-i}.

Proof: Let r be an input terminal of W that can reach s (through the control tag $c_{k-1}\cdots c_{k-i}$, say). The output terminals reachable from s are those reachable from i by the CT's $c_{k-1}\cdots c_{k-i}x_{k-i-1}\cdots x_o$ for all values of $x_{k-i-1}\cdots x_o$. These outputs number 2^{k-i}. □

Let $U = \{0, 1, \ldots, N/2 - 1\}$ and $L = \{N/2, N/2 + 1, \ldots, N-1\}$ be the upper half and lower half of the output terminals, respectively. Let also for all i, $0 \leq i \leq k-1$, U_i (resp., L_i) be the set of switches of col i from which at least one output in U (resp., L) is reachable.

Lemma 5.4: Let W be a D-controllable network of size $N = 2^k$. Every switch s of every column i (except for col 0) can reach outputs in U or in L but not in both. That is, $U_i \cap L_i = \emptyset$.

Proof: If we follow the same notation as in Lemma 5.3, the terminals reachable from s are $E_s = \{c_{k-1}\cdots c_{k-i}x_{k-i-1}\cdots x_o$ for all values of $x_{k-i-1}\cdots x_o\}$ because W is D-controllable.

$E_s \subseteq U$ or $E_s \subseteq L$ depending on whether $c_{k-1} = 0$ or 1, respectively. Since $U \cap L = \emptyset$, the lemma follows. □

Lemma 5.5: For all i, $1 \leq i \leq k-1$, the switches of U_i are linked to switches in U_{i+1} but not to switches in L_{i+1}. A similar statement holds for L_i.

Proof: If a switch s in U_i were linked to a switch in L_{i+1}, then s could reach output terminals in L in addition to terminals in U, contradicting Lemma 5.4. □

Lemma 5.6: Let $G = (V_0, V_1, E)$ be a bipartite graph whose two sets of nodes are V_0 and V_1. If the degree of each node is 2, then $|V_0| = |V_1|$.

Proof: E must be a collection of disjoint cycles (one or more). Each cycle must have an even number of nodes because G is bipartite. Moreover, one half of the nodes of each cycle is in V_0 and the other half is in V_1. Hence, $|V_0| = |V_1|$. □

Lemma 5.7: Let W be a D-controllable network of size $N = 2^k$. For all i, $1 \leq i \leq k-1$, $|U_i| = |L_i| = N/4$.

Proof: By backward induction on i.

Basis: $i = k\text{-}1$. Using Lemmas 5.1 and 5.2, it can be concluded that there are $N/4$ switches in col $k\text{-}1$ linked to U and $N/4$ to L. Hence, $|U_{k-1}| = |L_{k-1}| = N/4$.

Induction: Assume that the lemma is true for all values of $i = k\text{-}1, \ldots, l+1$ and it will proved for $i = l$.

$|U_{l+1}| = |L_{l+1}| = N/4$ by the inductive hypothesis.

The graph $G=(U_l, U_{l+1})$, whose nodes are the switches of U_l and U_{l+1} and whose edges are the links between U_l and U_{l+1}, is bipartite. It is clear that the degree of each node of G is 2. Thus, $|U_l| = |U_{l+1}|$ (by Lemma 5.6) and therefore $|U_l| = N/4$.

Same proof applies for $|L_l| = N/4$. □

Proposition 5.2: If W is D-controllable , then W is in GRN.

Proof: By induction on k.

Basis: $k=1$. It is trivial to see that W should be one of the networks of Fig. 5.1-a.

Induction: Assume that the proposition is true for all values of $k < l$ and it will be proved for $k = l$. The notation U_i, L_i, U and L is as defined earlier.

$|U_i| = |L_i| = N/4 = 2^l/4 = 2^{l-2}$ for all i (by Lemma 5.7).

Pull up (resp., down) all the switches of U_i (resp. L_i) to the upper (resp. lower) half of column i, for $i = 1, 2, \ldots, l\text{-}1$. Since $U_i \cap L_i = \varnothing$ for all i (Lemma 5.4) and since no link goes between U_i and L_{i+1} for any i (by Lemma 5.5), the subnetworks formed by $W_1=(U_1, U_2, \ldots, U_{l-1})$ and $W_2=(L_1, L_2, \ldots, L_{l-1})$ are disjoint. It should be straightforward to see that the upper (resp., lower) link of each switch of col 0 of W goes to some input of W_1 (resp. W_2), and that W_1 and W_2 are D-controllable networks of $l\text{-}1$ columns each. By the inductive hypothesis, W_1 and W_2 are in GRN. Hence, W is in GRN. □

Theorem 1: W is D-controllable if and only if W is in GRN.

Proof: Propositions 5.1 and 5.2. □

This theorem reveals the structure of D-controllable networks. Next, the structure of FD-controllable networks will be explored.

5.2 Structure of FD-Controllable networks

In this section, the structure of FD-controllable networks is studied. Recall that for a FD-controllable network W, $CT(i,j) = f(j)$ for some permutation f. f is called the control function of W. Only FD-controllable networks of easy-to-compute control functions are of particular merit.

If the control function f of an FD-controllable network is not efficiently computable by an algorithm, then it has to be stored in a one dimensional table. Although this table is much smaller than that of G-controllable networks, a copy of it has to be stored in each input terminal, making the overall space complexity the same as for G-controllable networks.

However, as it turned out, Class *FD* is closely related to Class *D* (of D-controllable networks) in structure, and the analysis of FD-controllable networks of easy-to-compute control functions is the same as that of FD-controllable networks of table-stored control functions. Therefore, Class *FD* will be treated in its totality, in addition to Class *D*.

A final note on control functions f when they are easy to compute. $f(j)$ can be computed at the input terminal and the tag is sent through the network, or it can be distributedly computed in the network via additional network hardware. This, however, will not be pursued any further here, for the main concern is the structure of D- &FD-controllable networks.

Definition 5.2: *Extended GRN* is the class of networks of the form Wf where W is in GRN and f is an arbitrary permutation. Recall that Wf is derived from W by

appending the interconnection f to its right end, or equivalently, by relabeling the output terminals of W by f (i.e., output i is relabeled $f(i)$).

It will be shown that extended GRN is the class of FD-controllable networks.

Lemma 5.8: If W is in GRN (i.e., D-controllable) and f is a permutation, then Wf is FD-controllable and $CT(Wf,i,j) = f^{-1}(j)$.

Proof: It is straightforward to see that $CT(Wf,i,j) = CT(W,i,f^{-1}(j))$. As W is D-controllable, $CT(W,i,f^{-1}(j)) = f^{-1}(j)$, and therefore $CT(Wf,i,j) = f^{-1}(j)$. □

Lemma 5.9: If W is FD-controllable and $CT(W,i,j) = g(j)$, then Wg is D-controllable.

Proof: $CT(Wg,i,j) = CT(W,i,g^{-1}(j)) = g(g^{-1}(j)) = j$. □

Lemma 5.10: If W' is FD-controllable where $CT(W',i,j) = g(j)$, then there exist a D-controllable network W and a permutation f such that $W' = Wf$.

Proof: By Lemma 5.9, $W'g$ is D-controllable. Let $W = W'g$ and $f = g^{-1}$. $Wf = Wg^{-1} = (W'g)g^{-1} = W'$. Hence, $W' = Wf$ where W is D-controllable. □

Theorem 5.2: W is FD-controllable if and only if W is in extended GRN.

Proof: By Lemmas 5.8 and 5.10 and the definition of extended GRN. □

This makes the structure of FD-controllable networks well understood. Before leaving extended GRN, two points should be made regarding its size. First, the number of networks in extended GRN is the product of the size of GRN and the number of all possible $N!$ right interconnections. Second, extended GRN is a proper subclass of MIN(2,k), that is, not every network is in extended GRN. For example,

the network shown in Fig. 5.5 is not in extended GRN. To show that, note that the second leftmost stage of a network in extended GRN, viewed as a graph where the switches are the nodes, has at least two connected components. The second leftmost stage of the network in Fig. 5.5 (i.e., the middle stage) is a single connected component, and hence, this network cannot be in extended GRN.

It follows then that the D-controllable networks are a proper subclass of the class of FD-controllable networks, which is in turn a proper subclass of the class of G-controllable networks.

5.3 Algorithms to Decide D- & FD-Controllability

It is of interest to have efficient algorithms that can determine if a network is D-controllable or FD-controllable, and in the latter case, to determine the control function. This section gives two optimal algorithms for D-controllability and FD-controllability and a third optimal algorithm to find the control functions of FD-controllable networks.

The GRN-structure of D-controllable networks will be exploited to develop the D-controllability algorithm. Necessary and sufficient conditions for a network to be in GRN are given in the following theorem. These conditions form the heart of the D-controllability algorithm.

If W is a network of size $N = 2^k$, denote by $G_i(j)$ the set of switches of col k-i that are reachable from output terminal j, for $i = 1, 2, \ldots k$. $G_0(j) = j$ for all output terminals j. Note that $G_{i+1}(j)$ is the set of switches in col k-i-1 that are linked to switches (or terminals in case $j=0$) in $G_i(j)$. Thus, $G_{i+1}(j)$ can be computed from $G_i(j)$.

Theorem 5.3: Let W be a network of size $N = 2^k$. W is in GRN if and only if $G_i(j2^i) = G_i(j2^i + 2^{i-1})$ and the upper output links of the switches of $G_i(j2^i)$ go to those in $G_{i-1}(j2^i)$ for $1 \le i \le k-1$, $0 \le j \le 2^{k-i} - 1$.

The proof is simple and therefore will not be presented.

Procedure D-controllable(W)

1. **For** $i=1$ **to** $k-1$
2. **For** $j = 0$ **to** $2^{k-i} - 1$
3. Compute $G_i(j2^i)$ from $G_{i-1}(j2^i)$, and $G_i(j2^i + 2^{i-1})$ from $G_{i-1}(j2^i + 2^{i-1})$; (**Comment**: It is assumed that while computing $G_i(j2^i)$ from $G_{i-1}(j2^i)$, if the lower output port of a switch s in $G_i(j2^i)$ is linked to a switch in $G_{i-1}(j2^i)$, the algorithm halts answering W is not D-controllable)
4. **For** $j = 0$ **to** $2^{k-i} - 1$
5. **if**($G_i(j2^i) \neq G_i(j2^i + 2^{i-1})$)
6. **output** (W is not D-controllable) and **exit**;
7. **output** (W is D-controllable);

Complexity: It should be clear that $|G_i(j)| = 2^{i-1}$ for all i and can be computed in $O(2|G_{i-1}(j2^i)|) = O(2^i)$ using the already computed and stored $G_{i-1}(j2^i)$. The same applies to $G_i(j2^i + 2^{i-1})$. Note that $G_{i-1}(j2^i + 2^{i-1}) = G_{i-1}((2j+1)2^{i-1})$ and thus was computed in a previous round. Thus, Step 2 computes $G_i(j2^i)$ and $G_i(j2^i + 2^{i-1})$ for all $j \leq 2^{k-i}-1$ in $O(2^{k-i} \times 2^i) = O(N)$. The comparison in Step 5 can be done by bucket-sorting $G_i(j2^i)$ and $G_i(j2^i + 2^{i-1})$ then comparing them element by element. This takes $O(2^i)$. Therefore, Step 4 takes $O(N)$. As a result, Step 1 takes $O(N\log N)$ time and space. This is obviously the lower bound (up to a constant factor), for the size of the data structure representing W is $O(N\log N)$ and any D-controllability algorithm has to read the input at least once.

Next, the FD-controllability algorithm will be given, using the above algorithm and the fact that FD-controllable networks W are of the form $W'f$, where W' is in GRN and f is a permutation. Note that the rightmost column of W can be in a "scrambled" form. To "unscramble" that column so that it is restored to the form of the rightmost column of W', some procedure has to be executed. Then, the right connection is dropped. Afterwards, the D-controllability algorithm is called against the resulting network. The unscrambling procedure is presented below in a self-explanatory manner.

Procedure unscramble(W)

For $j = j_{k-1}j_{k-2}\cdots j_1 = 0$ **to** $N/2-1$

trace the path from input terminal 0 in W using the control tag $j_{k-1}j_{k-2}\cdots j_1 0$;

Let s be the switch reached in col $k-1$;

relabel s by j (i.e., reposition s and put it at position j);

Clearly, *unscramble* is a depth first search from input terminal 0. Consequently, it takes $O(N)$.

Procedure FD-controllable (W)

unscramble(W); (**Comment**: This makes W have the form $W'f$)

drop the right connection; (**Comment**: drops f)

D-controllable (W');

The complexity of the FD-controllability algorithm is dominated by the D-controllable (W') call. Thus, it is $O(N\log N)$ too.

The following procedure determines the control functions of FD-controllable networks in $O(N)$ time.

Procedure control-function(W)

unscramble(W);

Let f be the right connection of the unscrambled W;

$g = f^{-1}$;

output (g); (**Comment**: g is the control function by Lemma 8)

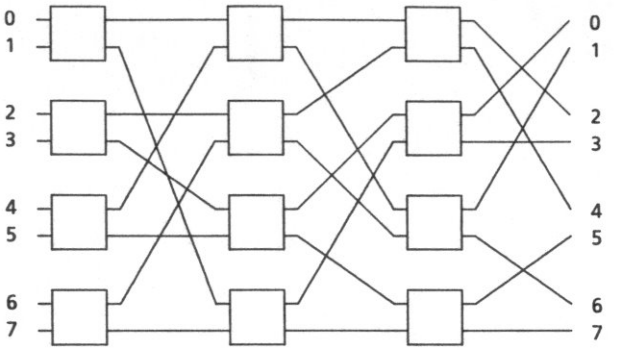
Fig. 5.3-a shows an FD-controllable network W, and Fig. 5.3-b shows the effect of calling the procedure "unscramble" on W, producing $W'f$, where f = [2 4 1 6 0 3 5 7]. Hence, the control function of W is $g = f^{-1}$ = [4 2 0 5 1 6 3 7].

5.4 Doubly Controllable Networks

So far, the control of networks has been from left terminals to right terminals (L-to-R). It is of theoretical and practical interest to study the networks that are also easily controllable from right terminals to left terminals (R-to-L). These networks are of importance for two-way communication in shared memory systems, where the left terminals of the network are processors and the right terminals are memory modules. These networks are the focus of this section.

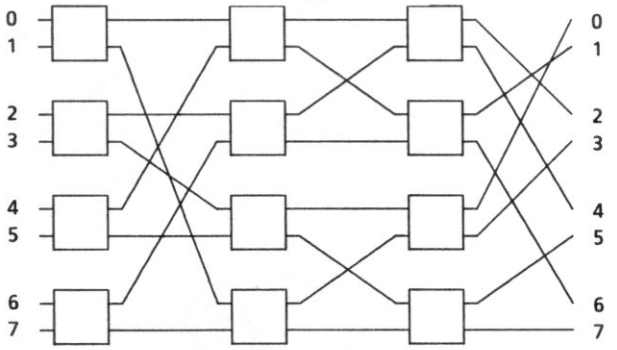
By right-to-left D-controllability it is meant that if right terminal j needs to communicate to left terminal $i = i_{k-1}i_{k-2}...i_0$ of a network W, then the control tag $i_{k-1}i_{k-2}...i_0$ is used to establish the path as follows. i_{k-1} controls the switches of col k-1 (i.e., The rightmost column), i_{k-2} controls the switches of col k-2, and so on. Right-to-left FD-controllability is defined similarly, where the control tag (denoted $CT_{R\text{-to-}L}(i,j)$) to establish the path from right terminal j to left terminal i is $f(i)$. f is called the *right-to-left control function*.

So for doubly controllable networks, we make a distinction between $CT_{L\text{-to-}R}$ (the old sense of CT) and $CT_{R\text{-to-}L}$, and a distinction between the left-to-right control



W - Before unscramble

(a)



After unscramble(W)

W' f

(b)

Fig. 5.3
The Effect of "unscramble"

functions and the right-to-left control functions, which do not have to be identical for a particular network.

Two questions arise at this point. The first is whether there are any doubly D-controllable (or FD-controllable) networks, and the second is whether there is any relationship among D-controllable (or FD-controllable) networks.

The baseline network B(N,2) is doubly D-controllable [WU80b]. Omega is doubly FD-controllable [SCH80]. This answers the first question.

As for the second question, it will be shown next that all doubly D-controllable networks have the same communication capabilities as B(N,2) and all doubly FD-controllable networks can be made equal to B(N,2) after renaming the input and output terminals of B(N,2). The proof of the former statement involves the following steps. If W is a doubly D-controllable network, it will be shown that the switches of the leftmost and the rightmost columns can be repositioned so that W is bare-ended and of the form $W = T(W_1,W_2)$; then, the switches of col *1* of W can be repositioned so that T becomes identical to the corresponding stage of B(N,2), and W_1 and W_2 become doubly D-controllable. Afterwards, the proof proceeds by simple induction. These steps will be proved and made more precise next.

The following lemma is straightforward.

Lemma 5.11: W is doubly D-controllable (resp., FD-controllable) if and only if W and W^{-1} are D-controllable (resp., FD-controllable). Moreover, the right-to-left control function of W is the left-to-right control function of W^{-1}, and $CT_{R\text{-}to\text{-}L}(W,i,j) = CT_{L\text{-}to\text{-}R}(W^{-1},j,i)$.

Hence, W is doubly D-controllable (resp., FD-controllable) if and only if W and W^{-1} are in GRN (resp., GRN-E).

Lemma 5.12: If W is doubly D-controllable , then the switches of col *0* and col k-*1* of W can be repositioned so that W is bare-ended.

Proof: Applying Lemma 5.2 on W, we conclude that col k-*1* can be repositioned to make W right bare-ended. Since col *0* of W is col k-*1* of W^{-1}, applying the same lemma on W^{-1} would make W^{-1} right bare-ended, which amounts to saying that col *0* of W can be repositioned to make W left bare-ended. □

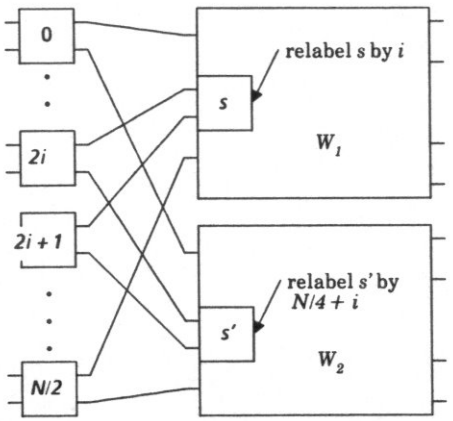
Lemma 5.13: Let $W = T(W_1,W_2)$ be a doubly D-controllable, bare-ended network. If a switch s of the leftmost column of W_1 (resp., W_2) is linked to switch $2i$ in col *0* of W, then s is linked to switch $2i+1$ of col *0* of W.

Proof: Col *0* and col *1* of W are col k-*1* and col k-*2* of W^{-1}. Since W^{-1} is in GRN and has no right connection, any switch s of col k-*2* that is linked to a switch of even label $2i$ in col k-*1* of W^{-1} is linked to switch $2i+1$ too. □

From the definition of B(N,2), it can be seen that B(N,2)=R(B_1,B_2) where B_1 and B_2 are two B(N/2,2)'s, and R = (col 0, col 1)$_B$ = $U_{N,2,0}$.

Lemma 5.14: Let $W = T(W_1,W_2)$ be a doubly D-controllable, bare-ended network. The switches of the leftmost column of W_1 (resp., W_2) can be repositioned so that T becomes identical with R.

Proof: After Lemma 5.13, if s is a switch of col *0* of W_1 (i.e., col *1* of W) that is linked to switch $2i$ of col *0* of W, then s is linked to switch $2i+1$. Relabel s by i (i.e., reposition switch s and put it at position i). The other switch s' of W_2 that is linked to switches $2i$ and $2i+1$ of col *0* of W is repositioned and put at position $N/4 + i$ (Fig. 5.4). After repositioning all the switches of Col *0* of W_1 and W_2, it should be clear that T becomes identical to R. □



$W = T(W_1,W_2)$

Fig. 5.4

Lemma 5.15: Let $W = T(W_1,W_2)$ be a doubly D-controllable, bare-ended network, such that T = R. Then W_1 and W_2 are doubly D-controllable.

Proof: W_1 and W_2 are D-controllable. It will be proved that W_1^{-1} is D-controllable. Let $i = i_{k-1}i_{k-3}...i_1$ and j be an input terminal and an output terminal of W_1, respectively. i is linked to switch $\lfloor i/2 \rfloor = i_{k-1}...i_2$ of col 0 of W_1. Combining the fact that $CT_{R\text{-to-}L}(W,2i,j) = 2i = i_{k-1}i_{k-3}...i_10$ and $R^{-1}(i) = 2i$, it can be concluded that the control tag $i = i_{k-1}i_{k-3}...i_1$ leads output terminal j to input terminal i in W_1. Consequently, W_1 must be doubly D-controllable. The same line of reasoning leads to the double D-controllability of W_2. □

Theorem 5.4: If W is a doubly D-controllable network, then W is strongly equivalent to B(*N*,2).

Proof: By Lemma 5.12 we can reposition the switches of col 0 and col k-1 of W so that W becomes bare-ended and of the form $W = T(W_1,W_2)$. Note that such repositioning of switches does not alter the power or control of W, as mentioned earlier. By Lemma 5.14, the switches of the leftmost columns of W_1 and W_2 can be repositioned so that T becomes equal to R. Then, using Lemma 5.15, we conclude that W_1 and W_2 are doubly D-controllable. The proof can proceed from here by simple induction. □

Therefore, all doubly D-controllable networks have the same permutational power. It will be shown that all doubly FD-controllable networks are weakly equivalent to B(*N*,2), allowing B(*N*,2) to simulate them all by renaming its input and output terminals.

Lemma 5.16: If $W = g\text{B}(N,2)f$, then $CT_{L\text{-to-}R}(W,i,j) = f^{-1}(j)$ and $CT_{R\text{-to-}L}(W,i,j) = g(i)$.

Proof: gB(*N*,2) is D-controllable ⇒ $CT_{L\text{-to-}R}(W,i,j) = f^{-1}(j)$ by Lemma 5.8. $W^{-1} = f^{-1}B^{-1}(N,2)g^{-1}$ and $B^{-1}(N,2) \equiv B(N,2) \Rightarrow CT_{R\text{-to-}L}(W,i,j) = CT_{L\text{-to-}R}(W^{-1},j,i) = g(i)$ by Lemma 5.8 too. □

Lemma 5.17: Let W be a doubly FD-controllable network and assume $CT_{L\text{-to-}R}(i,j) = f(j)$ and $CT_{R\text{-to-}L}(i,j) = g(i)$. Then, W is strongly equivalent $g\text{B}(N,2)f^{-1}$.

Proof: W is FD-controllable and $CT_{L\text{-to-}R}(i,j) = f(j) \Rightarrow Wf$ is D-controllable (Lemma 5.9). W^{-1} is FD-controllable and $CT_{L\text{-to-}R}(W^{-1},i,j) = CT_{R\text{-to-}L}(W,j,i) = g(j) \Rightarrow W^{-1}g$ is D-controllable.

Since Wf is D-controllable, $g^{-1}Wf$ must be D-controllable too. Similarly, $f^{-1}W^{-1}g$ must be D-controllable. As $f^{-1}W^{-1}g = (g^{-1}Wf)^{-1}$, we conclude that $g^{-1}Wf$ is doubly D-controllable. Thus, $g^{-1}Wf \equiv B(N,2)$ (Theorem 5.4) and therefore $W \equiv gB(N,2)f^{-1}$. □

Theorem 5.5: W is doubly FD-controllable if and only if W is weakly equivalent to $B(N,2)$.

Proof: Lemmas 5.16 and 5.17. □

The following result has been proved elsewhere [WU80a]. However, it follows here from Lemma 5.17.

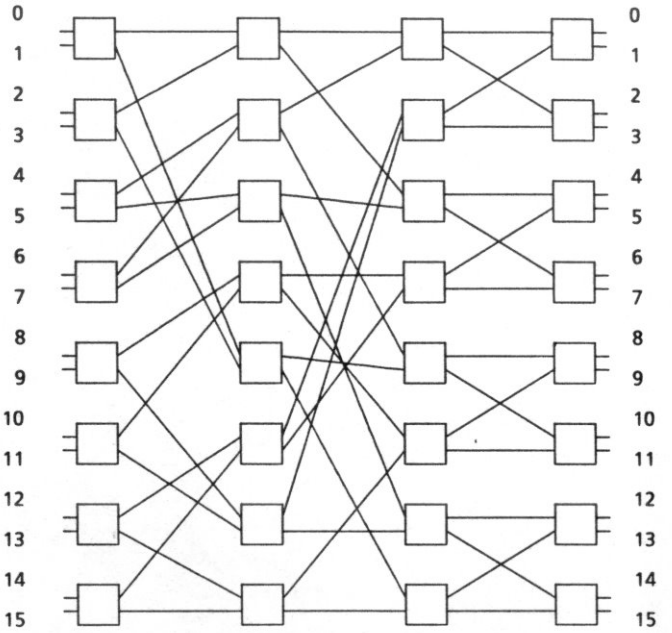
Theorem 5.6: $\Omega(N,2) \equiv \rho B(N,2)$ and $\Omega^{-1}(N,2) \equiv B(N,2)\rho$ where ρ is the bit reversal $(\rho(a_{k-1}...a_0) = a_0...a_{k-1})$.

Proof: $\Omega(N,2)$ is doubly FD-controllable, its left-to-right control function is the identity and its R-to-L control function is ρ [LAW75]. After Lemma 5.17, $\Omega(N,2) \equiv \rho B(N,2)$.

$\Omega^{-1}(N,2) \equiv B(N,2)\rho$ follows because $B^{-1}(N,2) \equiv B(N,2)$ and $\rho^{-1} = \rho$. □

The above results are easily generalizable to the case of $r \times r$ switches as building blocks, for arbitrary r. That is, all doubly D-controllable networks in $MIN(r,k)$ are strongly equivalent to $B(r,k)$, and all FD-controllable networks in $MIN(r,k)$ are weakly equivalent to $B(r,k)$.

The equivalence of all doubly FD-controllability will be usefull in the next chapter where a new subclass of networks, called digit-permutation networks, is studied. These networks will be shown doubly FD-controllable and hence topologically and functionally equivalent.



A Network that is not in extended GRN
Fig. 5.5

Chapter 6

Digit-Permutation Networks

6.1 Introduction

6.1.1 Motivation

One common feature in the definitions of the existing multistage interconnection networks is that the interconnections between columns are bit permutations, that is, operations that permute the bits of binary labels in a specified manner. The well known shuffle interconnection is an example. Among the reasons for using these permutations as interconnections are their regularity, rich structure and ease of analysis.

These same reasons may tempt one to propose new MIN's that have as interconnections bit permutations or, in the general case where the switches are $r \times r$, digit permutations that permute digits of r-ary labels. Therefore, there is a clear need to study the *class* of networks that are made out of digit permutations. This study focuses on three main considerations.

The first is to investigate the possibility of having just one structure underlying all these digit-permutation networks, that is, the possibility that all these networks are topologically and functionally equivalent. One reason to suspect this possibility is the fact that all existing MIN's turned out to have the same underlying structure [WU80a].

The second is to find necessary and sufficient conditions for $k+1$ digit permutations $f_0, f_1, \ldots, f_k$ to construct a network (as in Fig. 6.4) that is complete and single path. Such conditions would be useful in the design of digit-permutation networks and the verification of the essential preperties of completeness and single path.

The third is the development of efficient routing control for digit-permutation networks. The importance of efficient control is clear and has been emphasized throughout.

Before proceeding to the study of the three stated focuses of this chapter, the related work that has been done, mainly by Abidi [ABI80], is surveyed and discussed.

6.1.2 Previous Work

Abidi examined the second and third considerations concerning a class of MIN's called GEMINE (GEneralized Multistage Interconnection NEtworks) where the switches are 2×2 and the interconnections between columns are a special case of a linear transformation defined on bit vectors, where addition and multiplication is modulo 2. But he did not study any equivalence relationships between these networks.

A linear transformation f is a special mapping from S_N to S_N, where $S_N = \{0, 1, \ldots, N-1\}$, $N = 2^k$ and the integers of S_N are expressed as k-bit binary numbers. The mapping f is characterized by a $k \times k$ matrix A of 0/1 entries such that $f(x_{k-1} \ldots x_1 x_0) = A(x_0\ x_1\ \ldots\ x_{k-1})^T$, where $x_{k-1} \ldots x_1 x_0$ is an arbitrary binary number in S_N and $(x_0\ x_1 \ldots x_{k-1})^T$ its column matrix representation, and the arithmetic is modulo 2. With this column vector representation of the elements of S_N, and with modulo 2 arithmetic, S_N is a vector sapce over the field ($\{0,1\}$, $+$, $\times$). Thus, the name linear transformation is a legitimate here.

An example of a linear transformation: Let $N = 2^2 = 4$. $S_4 = \{0, 1, 2, 3\}$, or in binary, $\{00, 01, 10, 11\}$. Let f be the linear transformation of characteristic matrix

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \quad Then, f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, f\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$f\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1&1\\0&1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}, f\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}1&1\\0&1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}.$$

In decimal terms, f maps 0 to 0, 1 to 1, 2 to 3, and 3 to 2.

Note that a linear transformation f is a permutation of S_N if and only if its characteristic matrix is non-singular, that is, has a non-zero determinant..

Therefore, the interconnections between columns of GEMINE networks are linear transformations whose characteristic matrices are non-singular because these linear transformations have to be one-to-one and onto (i.e., permutations).

GEMINE will be shown later to include digit-permutation networks of 2×2 switches (these networks can also be called bit-permutation networks).

Abidi studied the necessary and sufficient conditions for $k+1$ matrices $A_0, A_1, \ldots, A_k$ to lead to a complete MIN, where the interconnection between column i-1 and column i is the linear transformation whose characteristic matrix is A_i, for $i = 1, 2, \ldots, k$-1, and the rightmost interconnection is the linear transformation of characteristic matrix A_k (refer to Fig. 6.4 for the general structure of GEMINE networks). These conditions are stated next.

Let $B_{k-i} = A_k A_{k-1} \ldots A_{i+1} A_i$ and $\underline{b}_{k-i}$ = the first column of B_{k-i}. Form the matrix $H = [\underline{b}_{k-1} \ \ldots \ \underline{b}_1 \ \underline{b}_0]$. The underline notation indicates a column vector in this chapter. The necessary and sufficient conditions found in [ABI80] are that $A_0, A_1, \ldots, A_k$ and H are non-singular. Note that the non-singularity of the A's is necessitated by the requirement that the interconnections between columns must be one-to-one, and the non-singularity of the matrix H is necessitated by the requirement that the network is complete and single path.

It takes $O((k+2)k^3) = O(k^4)$ to check these conditions because the complexity of computing the determinant is $O(k^3)$.

Abidi studied also the routing control of GEMINE networks and showed that the control tag c for the source-destination pair (s,d) is derived from the following equation: $d = B_k s + Hc$. This equation requires the storage of H^{-1} in every input terminal. Consequently, the space complexity of this control is $O(Nk^2\log_2 k) = O(N\log_2{}^2 N \log_2\log_2 N)$, and the time complexity to determine c is $O(k^2)$ for the multiplication of a matrix by a vector. This is costly in time and space if N is large.

6.1.3 The Contributions of this Chapter

This chapter generalizes the definitions and results related to linear transformations and GEMINE to the r-ary case for arbitrary r, shows that r-ary digit-permutation networks are included in the generalized GEMINE, and gives "optimal-to-check" necessary and sufficient conditions for a set of digit permutations, presented in a specified order, to construct a complete, single path network (as Fig. 6.4). The complexity of checking these conditions will be shown to achieve the lower bound $O(k^2)$.

Also, a better control scheme is developed for digit-permutation networks, which requires storing a digit permutation (of $O(k\log_2 k)$ space) in every input terminal s, and where the control tag c for the pair (s,d) is derived in $O(\log_2 N)$ time, providing thus a $\log_2 N$ factor in space saving and in speedup over the GEMINE control scheme.

This chapter also shows that all digit-permutation networks are doubly FD-controllable and therefore weakly (functionally) equivalent. This affirms what we suspected earlier, and supersedes the results in [SIE79] and [WU80a] which show that the existing MIN's are functionally equivalent.

One implication of this equivalence is that no digit-permutation network brings any new functional power, for it can be simulated by the baseline $B(r,k)$ by renaming the latter's terminals appropriately. Another implication is that when a

digit-permutation network is sought to realize a given set of permutations, the problem is reduced to finding a relabeling of the terminals of the baseline to realize these permutations if such a relabeling exists.

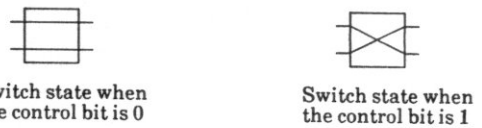
Finally, De Bruijn networks are generalized to other banyan networks, called *word networks*, which are contructed via digit manipulation . These networks are related to digit-permutation networks and shown to be topologically equivalent.

The chapter is organized as follows. Section 6.2 generalizes Abidi's work. Section 6.3 deals with digit-permutation networks. Section 6.4 studies the control ofdigit-permutation networks . Section 6.5 shows the double controllability and functional equivalence of all digit-permutation networks. De Bruijn and word networks are dealt with in Section 6.6. Section 6.7 summarizes the inclusion properties of all the different classes of networks dealt with so far. Section 6.8 concludes the chapter.

6.2 Linear Transformations and GEMINE$_r$

This section generalizes Abidi's work. The definitions of linear transformations and GEMINE are extended to accommodate networks whose building blocks are $r\times r$ switches, for arbitrary r.

Abidi's results are for the $r = 2$ case. To be able to generlize to arbitrary r, certain changes will be made to Abidi's use of control tags. For 2×2 switches, Abidi sets the switch to the straightthrough state if the control bit is 0, and to the cross state if the control bit is 1. This is hard to generalize to the general $r\times r$ switch. Our use of control digits is as defined in Chapter 2, and is straightforward for any r. It is as follows. If the control digit coming to an $r\times r$ switch is the r-ary digit c, then the switch is (partially) set to link the incoming input to the c-th output port of the switch. Fig. 6.1 illustrates the difference between Abidi's use and our use of control digits (or bits) for the $r = 2$ case. Fig. 6.2 illustrates our use of control digits for the



Switch state when the control bit is 0

Switch state when the control bit is 1

Abidi's Use of Control Bits

(a)

Switch state when the control bit is 0 and the input is at the upper input port

Switch state when the control bit is 0 and the input is at the lower input port

Switch state when the control bit is 1 and the input is at the upper input port

Switch state when the control bit is 1 and the input is at the lower input port

Our Use of Control Bits

(b)

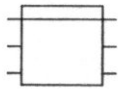
Fig. 6.1

$r=3$ case.

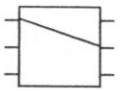
Now we are in a position to generalize Abidi's work.

Let $N = r^k$ throughout, and $S_N = \{0, 1, \ldots, N\text{-}1\}$. S_N represents the set of the labels of the network terminals as well as the labels of the input and output ports of the switch columns of the networks. Whenever convenient, these labels are expressed as r-ary labels of k r-ary digits of the form $x_{k-1}x_{k-2}\ldots x_0$ (each of the digits x_i is an element of the set $\{0, 1, \ldots, r\text{-}1\}$). An arbitrary such label $x_{k-1}x_{k-2}\ldots x_0$ will at times be represented by the column vector $(x_0\, x_1 \ldots\, x_{k-1})^T$.

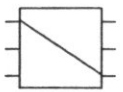
A matrix is said to be r-ary if its entries are r-ary digits.



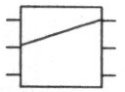
When 3-ary control digit is 0 and incoming input is at the top input port.



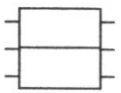
When 3-ary control digit is 1 and incoming input is at the top input port.



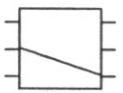
When 3-ary control digit is 2 and incoming input is at the top input port.



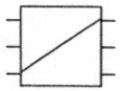
When 3-ary control digit is 0 and incoming input is at the middle input port.



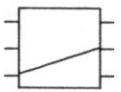
When 3-ary control digit is 1 and incoming input is at the middle input port.



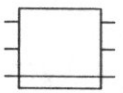
When 3-ary control digit is 2 and incoming input is at the middle input port.



When 3-ary control digit is 0 and incoming input is at the bottom input port.



When 3-ary control digit is 1 and incoming input is at the bottom input port.



When 3-ary control digit is 2 and incoming input is at the bottom input port.

Switch States for the $r = 3$ Case

Fig. 6.2

Definition 6.1: An r-ary linear transformation (denoted LT_r) of S_N is a mapping f from S_N to S_N where there is a $k \times k$ r-ary matrix A such that an arbitrary label in column vector representation $(x_0\ x_1\ \ldots\ x_{k-1})^T$ is mapped to $f((x_0\ x_1\ \ldots\ x_{k-1})^T) = A(x_0\ x_1\ \ldots\ x_{k-1})^T$. The arithmetic is modulo r.

Example:

Let $r = 3$, $k = 2$, and thus $N = 3^2 = 9$. The 9 lables of S_9 are (in 3-ary): 00, 01, 02, 10, 11, 12, 20, 21, 22. In column vector representation, the elements of S_9 are:

$$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}2\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}0\\2\end{bmatrix}, \begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}2\\2\end{bmatrix}.$$

Let f be the 3-ary linear transformation of characteristic matrix A, where

$$A = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}.$$

Then,

$$f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}, f\left(\begin{bmatrix}1\\0\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}2\\0\end{bmatrix}, f\left(\begin{bmatrix}2\\0\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}2\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix},$$

$$f\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\2\end{bmatrix}, f\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}0\\2\end{bmatrix}, f\left(\begin{bmatrix}2\\1\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix} = \begin{bmatrix}2\\2\end{bmatrix},$$

$$f\left(\begin{bmatrix}0\\2\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}0\\2\end{bmatrix} = \begin{bmatrix}2\\1\end{bmatrix}, f\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}, f\left(\begin{bmatrix}2\\2\end{bmatrix}\right) = \begin{bmatrix}2 & 1\\ 0 & 2\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}.$$

In more direct terms, f maps 3-ary label 00 to 3-ary label 00, 01 to 02, 02 to 01, 10 to 21, 11 to 20, 12 to 22, 20 to 12, 21 to 11, and 22 to 10. Expressed in decimal terms, f maps 0 to 0, 1 to 2, 2 to 1, 3 to 7, 4 to 6, 5 to 8, 6 to 5, 7 to 4, and 8 to 3.

Note that when $r = 2$, the modulo 2 arithmetic is similar to boolean operations ('+' is *xor* and '×' is *and*). For larger r, this similarity cannot be generalized. This will not cause any problems in this chapter, however, because no boolean operations are needed in the analysis.

Definition 6.2: Let $GEMINE_r$ be the class of networks of $MIN(r,k)$ where the interconnections between columns as well as the rightmost and leftmost interconnections are r-ary linear transformations.

Note then that GEMINE is $GEMINE_2$.

The following is an example of a network in $GEMINE_2$ (see Fig. 6.3). Let $r = 2$, $k = 3$ and thus $N = 2^3 = 8$. The network is bare-ended. The interconnection between column 0 and column 1, and that between column 1 and column 2 are two binary linear transformations of S_8, f_1 and f_2, with characteristic matrices A_1 and A_2, respectively, where

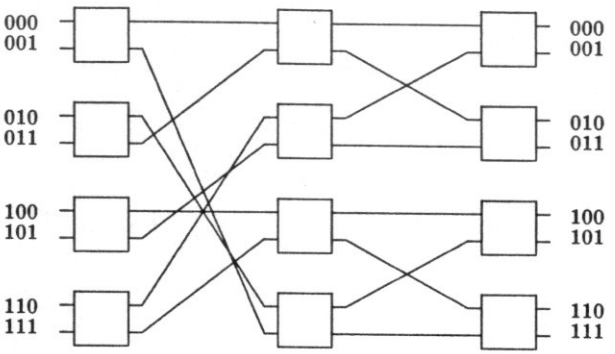
$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \textit{and} \quad A_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

It can be seen that f_1 maps (or connects) each binary label $x_2x_1x_0$ to the binary label $y_2y_1y_0$, where $y_0 = x_0$, $y_1 = x_0+x_1$ mod 2, and $y_2 = x_0+x_1+x_2$ mod 2. It thus maps 000 to 000, 001 to 111, 010 to 110, 011 to 001, 100 to 100, 101 to 011, 110 to 010, and 111 to 101. Similarly, f_2 maps each binary label $x_2x_1x_0$ to the binary label $x_2x_0x_1$. It thus maps 000 to 000, 001 to 010, 010 to 001, 011 to 011, 100 to 100, 101 to 110, 110 to 101, and 111 to 111. Hence the network of Fig. 6.3. This network can be clearly seen to be complete and single path.

Next, we will derive the necessary and sufficient conditions for LT_r's f_0, f_1, f_2, ..., f_k, of characteristic matrices $A_0, A_1, A_2, \ldots, A_k$, to form a MIN (as in Fig. 6.4) that is complete and single path. The proof is very similar to the proof in [ABI80] except for minor changes due to the different way of using control digits.

Let (s,d) be an arbitrary source-destination pair and $c = c_0c_1...c_{k-1}$ the r-ary control tag that establishes the path between s and d (in a way described in Chapter 2). The relationship between $s, d, c, A_0, A_1, A_2, \ldots$, and A_k will be found and expressed in matrix form. This relationship will help find the aforementioned conditions.

Denote by $\underline{m}^i$ (resp., $\underline{l}^{i+1}$) the input port r-ary label (resp., output port label) of column i through which the path between s and d goes, for $i = 0, 1, \ldots, k-1$. Digit c_i



A Network in $GEMINE_2$
Fig. 6.3

controls column i in such a way that the input port of the involved switch of column i is linked to the output port whose label differs from the input port only in the lowest order digit. Specifically, $\underline{m}^i$ and $\underline{l}^{i+1}$, when expressed in r-ary, are related as follows. Let $\underline{m}^i = m^i_{k-1} \ldots m^i_1 m^i_0$ (in r-ary). Then, $\underline{l}^{i+1} = m^i_{k-1} \ldots m^i_1 c_i$. This can be put as $\underline{l}^{i+1} = m^i_{k-1} \ldots m^i_1 c_i = m^i_{k-1} \ldots m^i_1 0 + 00 \ldots 0c_i$. It will also be expressed in matrix form, which will help in later analysis.

The column vector representation of $\underline{m}^i$ is $(m^i_0\ m^i_1 \ldots\ m^i_{k-1})^T$, and the representation of $\underline{l}^{i+1}$ is $(c_i\ m^i_1 \ldots\ m^i_{k-1})^T$, which is equal to $(0\ m^i_1 \ldots\ m^i_{k-1})^T + (c_i\ 0\ 0 \ldots\ 0)^T$.

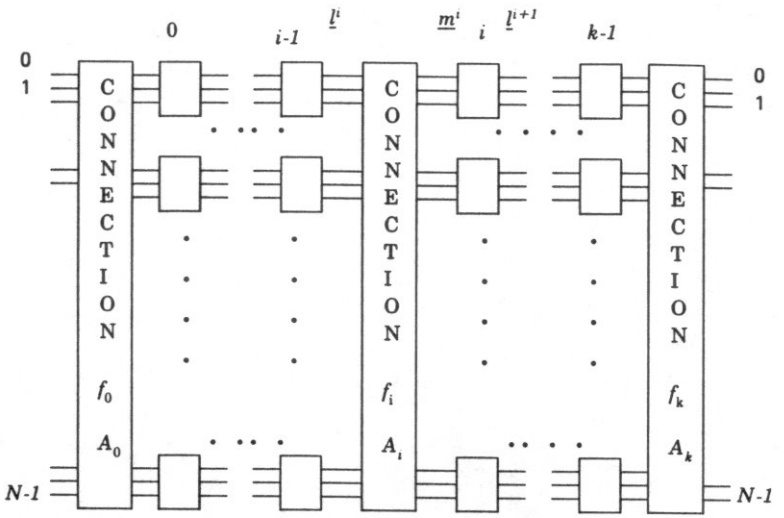
We can express $(0\ m^i_1 \ldots\ m^i_{k-1})^T$ better still:

$$(0\ m^i_1 \ldots\ m^i_{k-1})^T = Q(m^i_0\ m^i_1 \ldots\ m^i_{k-1})^T$$

where Q is the $k \times k$ r-ary matrix derived from the identity matrix by setting the upper left corner to 0.

Let $\underline{c}_i = (c_i\ 0\ 0 \ldots 0)^T$, we have then

$$\underline{l}^{i+1} = Q\underline{m}^i + \underline{c}_i.$$



The labels of the output ports of column i-1 and the labels of input ports of column i are shown. f_i connects the output ports of column i-1 to the input ports of column i (output port $\underline{l}^i$ is linked to input port $\underline{m}^i = f_i(\underline{l}^i) = A_i\underline{l}^i$)

Structure of GEMINE$_r$ Networks
Fig. 6.4

Also, by construction of the GEMINE networks, $\underline{m}^i$ and $\underline{l}^i$ are related by $\underline{m}^i = A_i\underline{l}^i$, for $i = 0,1, ..., k$ (note that $\underline{l}^0 = s$ and $\underline{m}^k = d$).

Note that in the $r = 2$ case and when control bits are used as in Fig. 6.1(a) (i.e., as used by Abidi), there would be no need to introduce the matrix Q. We would have $\underline{l}^{i+1} = \underline{m}^i + \underline{c}_i$. The matrix Q is needed because of our way of using control digits, which in turn is needed to generalize to the r-ary case. This will not complicate the analysis much, however.

Using the above matrix forms, a matrix relation between sources, destinations and control tags will be derived, and necessary and sufficient conditions for $A_0, A_1, A_2, \ldots, A_k$ to form a MIN that is complete and single path will be found in the following two theorems.

Theorem 6.1: In a GEMINE$_r$ network an arbitrary destination d is related to an arbitrary source s by the following matrix equation

$$d = B_k s + H\underline{c} \tag{6.1}$$

where

$$B_{k-i} = A_k Q A_{k-1} Q \ldots A_{i+1} Q A_i, \quad (B_0 = A_k) \tag{6.2a}$$

$$\underline{b}_{k-i} = \text{the first column of } B_{k-i} = B_{k-i}(1\,0\,0 \ldots 0)^T \tag{6.2b}$$

$$H = [\underline{b}_{k-1} \ldots \underline{b}_1\ \underline{b}_0] \tag{6.2c}$$

$$\underline{c} = (c_0\, c_1 \ldots c_{k-1})^T \tag{6.2d}$$

Proof: Using the recurrence equations $\underline{l}^{i+1} = Q\underline{m}^i + \underline{c}_i$ for $i = 0, 1, \ldots, k\text{-}1$, and $\underline{m}^i = A_i\underline{l}^i$ for $i = 0,1, \ldots, k$ where $\underline{l}^0 = s$ and $\underline{m}^k = d$, it follows that

$$d = \underline{m}^k = A_k\underline{l}^k = A_k(Q\underline{m}^{k-1} + \underline{c}_{k-1}) = A_k(Q(A_{k-1}(Q(A_{k-2}(\ldots(Q(A_2(Q(A_1(Q(A_0 s) + \underline{c}_0) + \underline{c}_1) + \underline{c}_2) + \ldots + \underline{c}_{k-3}) + \underline{c}_{k-2}) + \underline{c}_{k-1})$$

$$d = A_k Q A_{k-1} Q \ldots A_1 Q A_0 s + A_k Q A_{k-1} Q \ldots A_2 Q A_1 \underline{c}_0 + A_k Q A_{k-1} Q \ldots A_3 Q A_2 \underline{c}_1 + \ldots + A_k \underline{c}_{k-1}$$

$$d = B_k s + B_{k-1}\underline{c}_0 + B_{k-2}\underline{c}_1 + \ldots + B_0\underline{c}_{k-1}$$

$$d = B_k s + B_{k-1}(1\,0\,0 \ldots 0)^T c_0 + B_{k-2}(1\,0\,0 \ldots 0)^T c_1 + \ldots + B_0(1\,0\,0 \ldots 0)^T c_{k-1}$$

$$d = B_k s + \underline{b}_{k-1}c_0 + \underline{b}_{k-2}c_1 + \ldots + \underline{b}_k c_{k-1}$$

$$d = B_k s + [\underline{b}_{k-1} \ldots \underline{b}_1\ \underline{b}_0]\underline{c}$$

$$d = B_k s + H\underline{c}$$

□

For a fixed source s to reach all r^k destinations in a GEMINE$_r$, the vector $d = B_k s + H\underline{c}$ should assume r^k distinct values as the control tag $\underline{c}$ takes on all the r^k

values. Since s is fixed, $B_k s$ is a constant vector and thus d assumes r^k distinct values if and only if H is non-singular.

The Matrices A_i should also be non-singular because otherwise the corresponding r-ary linear transformation is not one-to-one and hence not a legitimate interconnection between two columns. The following theorem has then been proved.

Theorem 6.2: Let $A_0, A_1, \ldots, A_k$ be $k+1$ $k \times k$ matrices of r-ary entries and H as defined in Theorem 6.1. The network constructed from these matrices as shown in Fig. 6.4 is a complete, single path network if and only if $A_0, A_1, \ldots, A_k$ and H are non-singular.

As pointed out earlier, it takes $O(k^4)$ time to check the satisfiability of these conditions. As also done in Abidi, the relation $d = B_k s + H\underline{c}$, which is equivalent to $\underline{c} = H^{-1}(d - B_k s)$ (arithmetic is modulo r), leads to a routing control scheme that would require the storage of the matrices $B_k s$ and H^{-1} at every input terminal. That requires $O(Nk^2 \log_2 k)$ overall memory, and $O(k^2 \log_2 k)$ time to compute the control tag $\underline{c}$ every time source s needs to communicate with destination d.

These cost figures are relatively good in comparison with less structured networks, yet better costs are still called for when N is large.

A special subclass of GEMINE$_r$ will be defined in the next section for which the control cost is much lower. It is the subclass of digit-permutation networks.

6.3 Digit-Permutation Networks

This section defines Digit-Permutation Networks (DPN's), shows that the existing networks (defined in Section 2.2) are digit-permutation networks, and proves that digit-permutation networks are a subclass of GEMINE$_r$. We also derive

from Theorem 6.2 optimal-to-check necessary and sufficient conditions for a set of digit permutations to construct a digit-permutation network that is complete and single path.

Digit permutations are defined next, and then shown to be particular linear transformations. This will establish that digit-permutation networks are a subclass of GEMINE$_r$.

Definition 6.3: A permutation f of $S_N = \{0, 1, \ldots, N-1\}$, where $N = r^k$, is said to be an r-ary digit permutation if there exists a permutation π of $\{0,1, \ldots, k-1\}$ such that $f(x_{k-1}x_{k-2}\ldots x_0) = x_{\pi(k-1)}x_{\pi(k-2)}\ldots x_{\pi(0)}$, where $x_{k-1}x_{k-2}\ldots x_0$ is an arbitrary r-ary label. f is said to correspond to π and is denoted f_π.

Note that a digit permutation of S_N is totally specified by another permutation π of $\{0,1, \ldots, k-1\}$.

In the $r = 2$ case, digit permutations are also referred to as bit permutations.

Examples: Let $r = 2$, $k = 3$ and thus $N = 2^3 = 8$. The permutation f of S_8 such that $f(x_2x_1x_0) = x_1x_0x_2$ for any 3-bit binary label $x_2x_1x_0$ is a bit permutation. Its corresponding π is a permutation of $\{0, 1, 2\}$ where $\pi(2) = 1$, $\pi(1) = 0$ and $\pi(0) = 2$. It maps the binary label 000 to 000, 001 to 010, 010 to 100, and so on.

As another example, let $r = 3$, $k = 4$ and thus $N = 3^4 = 81$. The permutation f of S_{81} such that $f(x_3x_2x_1x_0) = x_0x_2x_3x_1$ for any 3-ary label $x_3x_2x_1x_0$ is a 3-ary digit permutation. Its corresponding π is a permutation of $\{0, 1, 2, 3\}$ where $\pi(3) = 0$, $\pi(2) = 2$, $\pi(1) = 3$ and $\pi(0) = 1$.

Shuffle, unshuffle, shuffle within segments, unshuffle within segments, and digit reversal are clearly digit permutations. In fact, all the permutations of Definition 2.4 are digit permutations.

Definition 6.4: A network of MIN(r,k) is called a digit-permutation network if the interconnections between the columns as well as the end interconnections are r-ary digit permutations of S_N, where $N = r^k$.

As the interconnections of the networks of Section 2.2 (Ω, Ω^{-1}, G, B and I) are permutations defined in Definition 2.4, it follows that these networks are digit-permutation networks.

The following theorem will show that r-ary digit permutations are linear transformations. This makes digit-permutation networks a subclass of GEMINE$_r$.

Theorem 6.3: Every r-ary digit permutation f_π of S_N is an r-ary linear transformation of S_N, where $N = r^k$. Moreover, the characteristic matrix of f_π is the permutation matrix of π.

Proof: Let f_π be an r-ary digit permutation of S_N. Note that π is a permutation of $\{0, 1, \ldots, k\text{-}1\}$. For every r-ary label $x_{k-1}x_{k-2}\ldots x_0$, we have

$$f(x_{k-1}x_{k-2}\ldots x_0) = x_{\pi(k-1)}x_{\pi(k-2)}\ldots x_{\pi(0)}.$$

Let A be the permutation matrix of π (not f), that is, A is a $k\times k$ matrix which has one 1 in each row and each column, and the position of 1 in the i-th row is in column $\pi(i)$ of the matrix. In other terms, $A = (a_{ij})$ where $a_{i\pi(i)} = 1$ for all $i = 0, 1, \ldots, k\text{-}1$, while all the other entries are 0.

We claim that $f(x_0 x_1 \ldots x_{k-1})^T) = A(x_0 x_1 \ldots x_{k-1})^T$. Recall that $(x_0 x_1 \ldots x_{k-1})^T$ is the vector column representation of the label $x_{k-1}x_{k-2}\ldots x_0$.

To show this, let $A(x_0 x_1 \ldots x_{k-1})^T = (y_0 y_1 \ldots y_{k-1})^T$. We must then have

$y_i = a_{i0}x_0 + a_{i1}x_1 + \ldots + a_{i,k-1}x_{k-1}$. As the coefficients a_{ij} are 0 for all j except for $j = \pi(i)$, it follows that $y_i = a_{i\pi(i)}x_{\pi(i)} = 1\times x_{\pi(i)} = x_{\pi(i)}$.

Thus, $A(x_0 x_1 \ldots x_{k-1})^T = (x_{\pi(0)} x_{\pi(1)} \ldots x_{\pi(k-1)})^T$, which is the column vector representation of $x_{\pi(k-1)}x_{\pi(k-2)}\ldots x_{\pi(0)}$. It follows then that

$$f(x_0 x_1 \ldots x_{k-1})^T) = A(x_0 x_1 \ldots x_{k-1})^T$$

and the theorem is proved. □

The permutation matrix of permutation π will be denoted here A_π.

Examples: Let $r = 3$, $k = 2$, and thus $N = 3^2 = 9$. Let f_π be the 3-ary digit permutation of S_9 such that for every 3-ary label x_1x_0, $f_\pi(x_1x_0) = x_0x_1$. So π is a permutation of $\{0, 1\}$ that maps 0 to 1 and 1 to 0. The permutation matrix A_π of π is

$$A_\pi = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Clearly, $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_0 \end{bmatrix}$. *Since* $f_\pi(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) = \begin{bmatrix} x_1 \\ x_0 \end{bmatrix}$, *it follows that* $f_\pi(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) = A_\pi\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$.

f_π is then an 3-ary linear transformation of charateristic matrix A_π.

One direct observation is that the inverse of a digit permutation is a digit permutation because $(f_\pi)^{-1} = f_{\pi^{-1}}$. Consequently, the inverse of a digit-permutation network is a digit-permutation network.

The necessary and sufficient conditions of Theorem 6.2 will be refined next for the case of digit-permutation networks.

Let the digit permutations f_{π_0}, f_{π_1}, ... , f_{π_k} be the interconnections of an arbitrary digit-permutation network. That is, the leftmost interconnection is f_{π_0}, the interconnection between column 0 and column 1 is f_{π_1}, and so on. After Theorem 6.3, these permutations are linear transformations of characteristic $k\times k$ matrices A_{π_0}, A_{π_1}, ... , and A_{π_k}, respectively. We can then apply Theorems 6.1 and 6.2 on these matrices. The matrix B_{k-i} will now be shown to have a special structure.

$B_{k-i} = A_{\pi_k}QA_{\pi_{k-1}}Q \ldots A_{\pi_{i-1}}QA_{\pi_i}$ (as defined in Theorem 6.1). The Q's will be "filtered out" to the leftmost end of the A's.

Let Q_i be the $k \times k$ matrix derived from the identity matrix by making the i-th diagonal entry 0. Q is then Q_0.

It is straightforward to show that $A_n Q_i = Q_{\pi^{-1}(i)} A_n$.

Now we are in a position to filter out the Q's in B_{k-i} to the left end.

Let $\beta_i = \pi_i \ldots \pi_{k-1} \pi_k$.

$B_{k-i} = A_{\pi_k} Q_0 A_{\pi_{k-1}} Q_0 \ldots A_{\pi_{i-1}} Q_0 A_{\pi_i} = Q_{\pi^{-1}{}_k(0)} A_{\pi_k} A_{\pi_{k-1}} Q_0 \ldots A_{\pi_{i-1}} Q_0 A_{\pi_i}$

$B_{k-i} = Q_{\pi^{-1}{}_k(0)} A_{\pi_{k-1} \pi_k} Q_0 \ldots A_{\pi_{i-1}} Q_0 A_{\pi_i}$ because $A_\alpha A_\beta = A_{\beta\alpha}$

$B_{k-i} = Q_{\beta^{-1}{}_k(0)} A_{\beta_{k-1}} Q_0 \ldots A_{\pi_{i-1}} Q_0 A_{\pi_i}$ because $\beta^{-1}{}_k = \pi^{-1}{}_k$ and $\beta_{k-1} = \pi_{k-1} \pi_k$.

$B_{k-i} = Q_{\beta^{-1}{}_k(0)} Q_{\beta^{-1}{}_{k-1}(0)} A_{\beta_{k-1}} A_{\pi_{k-2}} Q_0 \ldots A_{\pi_{i-1}} Q_0 A_{\pi_i}$

$B_{k-i} = Q_{\beta^{-1}{}_k(0)} Q_{\beta^{-1}{}_{k-1}(0)} A_{\beta_{k-2}} Q_0 \ldots A_{\pi_{i-1}} Q_0 A_{\pi_i}$

. . . . . .

$B_{k-i} = Q_{\beta^{-1}{}_k(0)} Q_{\beta^{-1}{}_{k-1}(0)} \ldots Q_{\beta^{-1}{}_{i+1}(0)} A_{\beta_i}$

The effect of multiplying $Q_{\beta^{-1}{}_k(0)} Q_{\beta^{-1}{}_{k-1}(0)} \ldots Q_{\beta^{-1}{}_{i+1}(0)}$ by A_{β_i} is to turn the entries of rows $\beta^{-1}{}_k(0)$, $\beta^{-1}{}_{k-1}(0)$ ... $\beta^{-1}{}_{i+1}(0)$ of A_{β_i} to 0. Consequently, the entries of rows $\beta^{-1}{}_k(0)$, $\beta^{-1}{}_{k-1}(0)$, ..., $\beta^{-1}{}_{i+1}(0)$ of B_{k-i} are 0.

The only "1" in the first column of A_{β_i} is in row $\beta^{-1}{}_i(0)$. Therefore, the first column $\underline{b}_{k-i}$ of B_{k-i} has all zero's except for "1" in the $\beta^{-1}{}_i(0)$-th entry if $\beta^{-1}{}_i(0)$ is different from $\beta^{-1}{}_k(0)$, $\beta^{-1}{}_{k-1}(0)$, ..., and $\beta^{-1}{}_{i+1}(0)$. Otherwise, $\underline{b}_{k-i}$ is all 0.

We can now simplify the necessary and sufficient conditions of Theorem 6.2 for the digit-permutation network case in the following theorem.

Theorem 6.4: Let $\pi_0, \ldots, \pi_{k-1}, \pi_k$ be $k+1$ permutations of $\{0, 1, \ldots, k-1\}$. The network constructed from f_{π_0}, f_{π_1}, f_{π_2}, ..., f_{π_k} is complete and single path if and only if $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ are pairwise distinct, where $\beta_i = \pi_i \ldots \pi_{k-1} \pi_k$.

Proof: As defined in Theorem 6.1, $H = [\underline{b}_{k-1} \ldots \underline{b}_1 \ \underline{b}_0]$. By Theorem 6.2, it is enough to show that the matrix H and the A_{π_i}'s are nonsingular iff $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ are pairwise distinct. As the matrices A_{π_i}'s are permutation matrices, they are non-singular. Thus, we only have to show that H is nonsingular iff $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ..., $\beta_k{}^{-1}(0)$ are pairwise distinct .

If $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ are pairwise distinct, then for all $i = 0, 1, \ldots, k-1$, $\underline{b}_{k-i}$ has all 0's except for "1" in the $\beta^{-1}{}_i(0)$-th entry, and hence H has only one "1" in each column. These 1's are in rows $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ which are pairwise distinct. Consequently, H must be a permutation matrix A_π for some π and is thus non-singular. One can compute π as follows. For $i = 0, 1, \ldots, k-1$, $\pi^{-1}(i) = \beta^{-1}{}_{i+1}(0)$ because $\pi^{-1}(i)$ is the row position where the "1" is in column i of H, which is $\underline{b}_{k-i-1}$.

To show the only if part, we reason by contradiction. Assume that $\beta_i{}^{-1}(0) = \beta_j{}^{-1}(0)$ for some $i < j$. Then $\beta_i{}^{-1}(0)$ occurs in $\beta^{-1}{}_k(0)$, $\beta^{-1}{}_{k-1}(0)$, ... , $\beta_j{}^{-1}(0)$, ... , $\beta^{-1}{}_{i+1}(0)$ and hence the column $\underline{b}_{k-i}$ is all 0, making H singular, and thus contradicting the hypothesis. Therefore, $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ are pairwise distinct. □

A special case is when all the π_i's are equal to the same permutation α. All the stages are then identical. In this case, $\beta_i = \alpha^{k-i+1}$. Then, $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ are pairwise distinct iff $\alpha^{-1}(0)$, $\alpha^{-2}(0)$, ... , $\alpha^{-k}(0)$ are pairwise distinct. That is equivalent to saying that α^{-1}, written in a cyclic form, has only one cycle ($\alpha^{-1}(0)$ $\alpha^{-2}(0)$ ... $\alpha^{-k}(0)$). But that is also equivalent to saying that α has only one cycle. Thus, the necessary and sufficient conditions of the preceding theorem reduce to α having only one cycle. The number of such networks is then the number of one-cycle permutations. That number is $(k-1)!$ because the cycle can always start with 0, and the remaining numbers are any permutation of $\{1, 2, \ldots, k-1\}$.

The complexity of checking whether $\beta_1{}^{-1}(0)$, $\beta_2{}^{-1}(0)$, ... , $\beta_k{}^{-1}(0)$ are pairwise distinct is $O(k^2) = O(\log_r{}^2 N)$ for the following reasons. To compute $\pi^{-1}{}_0, \ldots, \pi^{-1}{}_{k-1}, \pi^{-1}{}_k$,

it takes $O(k^2)$ because inverting a permutation takes linear time. Afterwards, to compute $\beta_i^{-1}(0)$ takes $O(k\text{-}i)$, and consequently, $\beta_1^{-1}(0), \beta_2^{-1}(0), \ldots, \beta_k^{-1}(0)$ take the sum of $O(k\text{-}i)$, which is $O(k^2)$. Checking them to see if they are pairwise distinct takes $O(k)$ by taking an array of size k, and putting $\beta_i^{-1}(0)$ in the $\beta_i^{-1}(0)$-th bucket. If any bucket holds more than one number, the condition is violated. Thus, the overall complexity is $O(k^2) = O(\log^2 N)$. This is the lower bound (up to a constant factor) because the size of the input $\pi_0, \ldots, \pi_{k-1}, \pi_k$ is $O(k^2)$.

6.4 Control of Digit-Permutation Networks

Abidi did not deal with digit-permutation networks, even for the $r = 2$ case. In this section, digit-permutation networks are shown to be efficiently controllable.

From the expression of B_{k-i}, it follows that $B_k = Q_{\beta\text{-}1_k(0)} Q_{\beta\text{-}1_{k-1}(0)} \cdots Q_{\beta\text{-}1_1(0)} A_{\beta_0}$. As in a complete, single path DPN the integers $\beta_1^{-1}(0), \beta_2^{-1}(0), \ldots, \beta_k^{-1}(0)$ are pairwise distinct and hence span 0, 1, ... , k-1, it follows that B_k is the zero matrix. Consequently, the relation between the source s, the destination d, and the control tag $\underline{c}$ in a DPN is: $d = B_k s + H\underline{c} = H\underline{c}$. In the proof of Theorem 6.3, H was shown to be some permutation matrix A_π. H^{-1} is then $A_{\pi\text{-}1}$, and $\underline{c} = A_{\pi\text{-}1} d = f_{\pi\text{-}1}(d)$.

Consequently, if we store π^{-1} in source s for all s, then it takes $O(k) = O(\log_r N)$ time to compute the control tag $\underline{c} = f_{\pi\text{-}1}(d)$, assuming the uniform model of time complexity, that is, the time to fetch $\pi^{-1}(i)$ is a constant independent of k, and the time to move the r-*ary* digits around is also a constant independent of r.

As a result, the time complexity of the routing control of a digit-permutation network is $O(\log_2 N)$, and the space complexity is $O(Nk\log_2 k)$ because the storage of π^{-1} in each source requires $O(k\log_2 k)$ and this amount has to be stored in all the N sources. This shows a speedup in time and a saving in space by a factor of $\log_2 N$ over the control complexity of general GEMINE networks.

As an example, assume $r = 16 = 2^4$ (i.e, 16×16 switches) and $k = 5$, so that the number of terminals on each side of the network is $N = r^k = 2^{20} = 1\text{M}$. The number of computer operations to compute the control tag is $\log_2 N = 20$ operations (of Fetches and data movement), which takes 20 μs if each of these operations takes one μs. The space required at each input terminal is $k\lceil \log_2 k \rceil = 15$ bits, or a 16-bit computer word. That shows the time and space efficiency of the routing control of reasonably large systems.

6.5 Functional Equivalence of Digit-Permutation Networks

As the control tag $\underline{c}$ that establishes the path between an arbitrary source s and an arbitrary destination d in a digit-permutation network is related to d by $\underline{c} = H^{-1}d$ and hence a function of the destination tag alone (i.e., independent of the source s), it follows that every digit-permutation network is FD-controllable (from last chapter).

Since the inverse of a digit-permutation network is a digit-permutation network, we conclude that every digit-permutation network is doubly FD-controllable.

The double FD-controllability of digit-permutation networks yields that all they are topologically and hence weakly (functionally) equivalent to the baseline network, by Theorem 5.5. This reveals the structure of digit-permutation networks and shows that these networks bring no new communication power because this functional equivalence with the baseline makes the latter able to simulate any digit-permutation network in one pass by relabeling the end terminals of the baseline appropriately. This result is a generalization of the functional equivalence results of the existing networks (of Section 2.2) by [WU80a].

To find the new labels of the baseline to simulate a given digit-permutation network W, Procedure *control-function* of section 5.3 is called on W and then on W^{-1}

to get the left-to-right control function f and the right-to-left control function g of W. This procedure can be called because W is doubly FD-controllable. By Lemma 5.17, W is strongly equivalent to $gB(r,k)f^{-1}$.

Therefore, the output terminals of $B(r,k)$ should be relabeled by f^{-1} and the input terminals by g^{-1}.

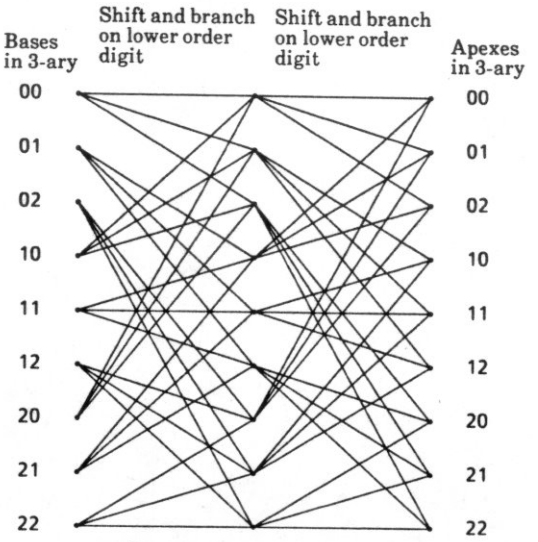
After such relabeling, the network can be controlled in the same way as network W, as explained in Section 6.4.

6.6 De Bruijn Networks

De Bruijn networks [SCH74] are regular rectangular banyan networks. For node degree r and number of bases r^{k-1}, the De Bruijn network has k columns of r^{k-1} nodes each. Assuming the nodes in each column are labeled in r-ary, from 0 to $r^{k-1}-1$, the successive columns are interconnected as follows. Node $x_{k-1}x_{k-2}\cdots x_1$ in each column is linked to the r nodes labeled $x_{k-2}\cdots x_1 0$, $x_{k-2}\cdots x_1 1$, ... , and $x_{k-2}\cdots x_1(r-1)$ in the succeeding column. That is, each node label is circularly shifted left and then branched to the r nodes having the r distinct digits in the low order position. Fig. 6.5 shows a De Bruijn network.

It can be seen that any base label can be transformed to any apex label by $k-1$ operations of the kind described above. This leads to the completeness of these networks. This transformation by those operations is unique, and that shows that De Bruijn networks are single path.

De Bruijn shifting of labels then branching on the lowest order digit is not the only way to construct regular rectangular banyan networks via digit manipulation. Other ways of construction will be presented and the resulting banyans will be called word banyans. These banyans and De Bruijn networks will be related to digit permutation networks. In particular, word banyans and De Bruijn networks are



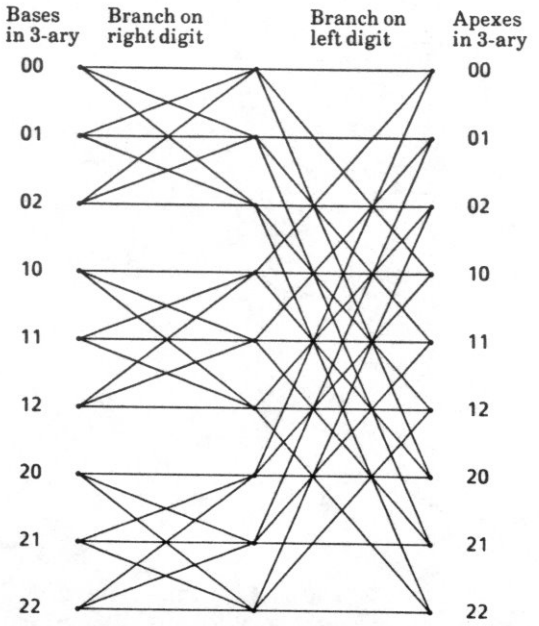
DE Bruijn Network of node degree $r = 3$ and of $3^2 = 9$ nodes in each column.

Fig. 6.5

shown to be representing banyans of DPN. And consequently, one can conclude that all word banyans and De Bruijn networks are topologically equivalent.

Let π be a permutation of $\{1, \ldots, k-1\}$ and r an arbitrary integer greater than or equal to 2. The regular rectangular banyan of k columns of r^{k-1} nodes each, denoted $WB(r,k,\pi)$ and called a *word banyan*, is defined as follows. Assume the columns are numbered 0, 1, ... , $k-1$ from base to apex, and the nodes in each column are labeled 0, 1, ..., $r^{k-1}-1$ in r-ary. Each node $x_{k-1}x_{k-2}\cdots x_1$ in column i is linked to the r

nodes that may differ from $x_{k-1}x_{k-2}...x_1$ in the $\pi(i)$-th digit position only. Those nodes are in column $i+1$. Fig. 6.6 shows WB(3,3, π = the identity).



WB(3,3, π = the identity)

Fig. 6.6

It can be seen that there is a path between each base and each apex because each r-ary label can be transformed to any other label by first changing the former's $\pi(1)$-th digit to the latter's $\pi(1)$-th digit, then by doing the same to all the remaining digits. Note that the fact that π is a permutation allows us to access each digit

position once. Thus, WB(r,k,π) is single path and complete. The outdegree of each node (except for the apexes) is clearly r. As the number of nodes is the same in each columns, the indegree of each node (except the bases) must also be r. Hence the regularity of word banyans.

Note also that π has to be a permutation if the network is to be complete.

The number of word banyans of the same degree r and same number of bases r^{k-1} is the number of the permutations of $\{1, 2, ..., k-1\}$, and that is $(k-1)!$.

Next, each word banyan is shown to be the representing banyan of a digit permutation network. Recall that the representing banyan of a MIN is derived from the MIN by replacing the switches by nodes, and by dropping the leftmost and rightmost interconnections.

Let WB(r,k,π) be an arbitrary word banyan. A network W in MIN(r,k) will be constructed from WB(r,k,π) by replacing each node by an $r \times r$ permutation switch labeled with the node's label. Each edge between two nodes in WB(r,k,π) will connect the corresponding two switches. What is required is to specify which ports of the two switches the edge will link. The ports on each side of switch $x_{k-1}x_{k-2}...x_1$ are labeled $x_{k-1}x_{k-2}...x_1 0$, $x_{k-1}x_{k-2}...x_1 1$, ..., $x_{k-1}x_{k-2}...x_1(r-1)$ from top to bottom.

The edge between node $x_{k-1}x_{k-2}...x_1$ of column i and node $x_{k-1}x_{k-2}...x_{j+1}y_jx_{j-1}...x_1$ (where $j = \pi(i)$) of column $i+1$ of WB(r,k,π) goes from output port $x_{k-1}x_{k-2}...x_1\ y_j$ of switch $x_{k-1}x_{k-2}...x_1$ of column i to input port $x_{k-1}x_{k-2}...x_{j+1}y_jx_{j-1}...x_1x_j$ of switch $x_{k-1}x_{k-2}...x_{j+1}y_jx_{j-1}...x_1$ of column $i+1$ of W.

It can be easily shown that the resulting network is a complete, single path network in MIN(r, k). It remains to show that W is a digit permutation network.

The interconnection between column i and column $i+1$ of W is the permutation that maps $x_{k-1}x_{k-2}...x_1y_j$ to $x_{k-1}x_{k-2}...x_{j+1}y_jx_{j-1}...x_1x_j$, which obviously swaps the digits in digit position 0 and $j = \pi(i)$. Thus, this interconnection is a digit

permutation. As i was chosen arbitrarily, it follows that all the interconnections of W are digit permutations, and hence W is a digit permutation network.

From the construction of W, it is clear that WB(r,k,π) is the representing banyan of W.

Therefore, each word banyan is the representing banyan of a digit permutation network.

In a similar fashion, each De Bruijn network (of parameters r and k) can be shown to be the representing banyan of a digit permutation network, namely $\Omega(r,k)$.

As all r-ary digit-permutation networks of the same number of terminals are topologically equivalent, it follows that all $(k-1)!$ word banyans with the same parameters r and k but with varying π (as well as the De Bruijn network of parameters r and k,) are topologically equivalent. That is, any of these networks can transformed to any other network by relabeling their nodes.

De Bruin networks and word banyans are not the only regular rectangular banyans that can be constructed via digit manipulations. The stages between columns of r-ary labeled nodes can be constructed by first permuting the digits of the labels by some digit permutation (not necessarily shifting them), then branch to the nodes that differ only in one specified digit position. Networks constructed in this fashion can also be shown to be graph representations of digit permutation networks. Therefore, similar necessary and sufficient conditions (for the digit permutations and digit position choices for branching employed in the construction of stages to yield complete, single path regular rectangular banyans) can be derived from the necessary and suffucient conditions given in Theorem 6.2. In addition, these "digit manipulation" banyans, being representing banyans of digit-permutation networks, are all topologically equivalent.

6.7 Summary of Equivalence and Inclusion Properties

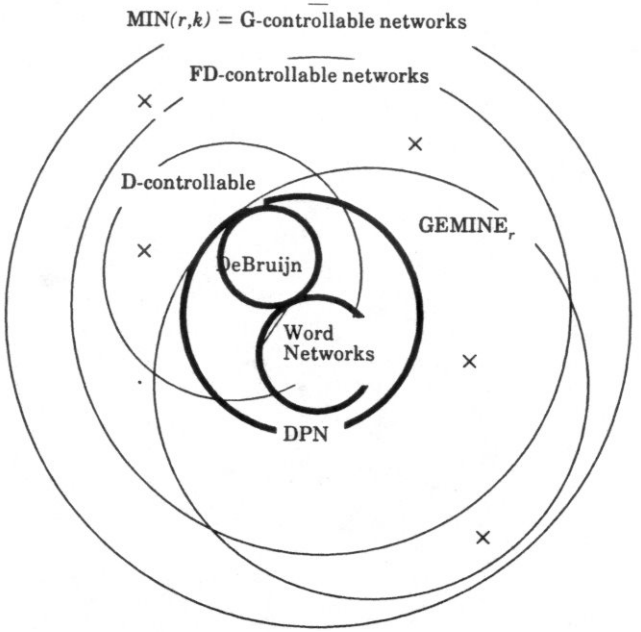
Several subclasses of networks have been introduced thus far, and equivalence as well as inclusion properties have been shown. The following diagram in Fig. 6.7 summarizes these properties. The networks in the dark-circled regions are functionally equivalent.

The largest class of interest is MIN(r,k), which is the class of G-controllable networks. This then properly includes the subclass of FD-controllable networks. Fig. 5.5 shows a network in MIN(r,k) that is not FD-controllable.

GEMINE$_r$ includes the class of digit-permutation networks (DPN) as pointed out in Section 6.3. Section 6.6 implies that the class of digit-permutation networks includes De Bruijn networks and word networks. All the digit-permutation networks are functionally equivalent after Section 6.5.

The class of D-controllable networks incudes De Bruijn networks and some of the word networks as the latter can be D-controllable or FD-controllable. Omega inverse for example is a word network that is not D-controllable. If an interconnection that is not a linear transformation is appended to the left of the baseline B(r,k), the resulting network is clearly D-controllable but not in GEMINE$_r$. Also, if an LT that is not a digit permutation is appended to the right of the baseline B(r,k), the resulting network is in GEMINE$_r$ but is neither a digit-permutation network nor a D-controllable network. Hence, the class of digit-permutation networks is properly contained in GEMINE$_r$, and the class of D-controllable networks overlaps with GEMINE$_r$ but neither includes it nor is included by it. We can also show, though tediously, that there are networks in GEMINE$_r$ that are not FD-controllable. The proof is beyond the scope of this dissertation.

6.8 Conclusion



Inclusion Diagram. The × indicates that the corresponding part of the set is not empty.

Fig 6.7

It has been shown that all digit-permutation networks are functionally equivalent to the baseline network, and "optimal-to-check" necessary and sufficient conditions for a set of digit permutations, presented in a specified order, to construct a complete, single path network have been derived. Efficient routing control was also developed for these networks.

The functional equivalence of these networks with the baseline $B(r,k)$ enables the latter to simulate any DPN by relabeling the terminals of the baseline and controlling it in the same way as the simulated network. The relabeling can be done in linear time, and the routing control of the relabeled baseline takes $O(k\log_2 k)$ space (in every input terminal) and $O(\log_2 N)$ time.

De Bruijn networks were shown to be some of many regular rectangular banyans constructed via digit transformations. Other such networks were defined, particularly, word networks, and were shown to be representing banyans of digit permutation networks. As all digit-permutation networks are topologically and functionally equivalent, it was concluded that De Bruijn networks, word banyans and all other regular rectangular banyans of node degree r and r^{k-1} bases, constructed via digit transformations, are topologically equivalent.

Chapter 7

Modular Networks

Modular multistage networks consist of identical stages and have therefore an important advantage over the non-modular ones. Besides manufacturing efficiency, each of these networks can be simulated by one single stage using multiple passes. This warrants the study of the class of modular networks, and particularly the ones that are FD-controllable because of the control efficiency considerations.

Among existing multistage interconnection networks, omega and its inverse are modular.

As with digit-permutation networks, the main concern is whether FD-controllable modular networks share the same underlying structure, for that would reduce the quest of an FD-controllable modular network that realizes a given set of permutations to a search for a relabeling of the terminals of omega. It would also enable omega to simulate any FD-controllable modular networks by relabeling the former's terminals.

The contribution of this chapter is showing that D-controllable modular networks are all strongly equivalent to Ω, and that modular FD-controllable networks are all conjugately equivalent to Ω. The implications stated in the preceding paragraph then follow.

The analysis here is carried out on networks in MIN($2,k$), but can be generalized to MIN(r,k) for arbitrary r.

For convenience of analysis, we assume that modular networks are right bare ended and the leftmost interconnection (-, col 0) is the same as all the other interconnections between the columns. This assumption does not affect weak

equivalence. At the end of the chapter, it is concluded that all the networks that have all the inner interconnections identical but the left and right interconnections arbitrary are weakly equivalent to omega.

The chapter is organized as follows. Modular recursive banyan networks are defined and characterized in Section 7.1. Section 7.2 studies modular recursive banyans and their synthesis, and shows that they all have the same underlying structure. Section 7.3 deals with the strong equivalence and conjugate equivalence among all modular D- and FD-controllable networks. Some concluding remarks are presented in Section 7.4.

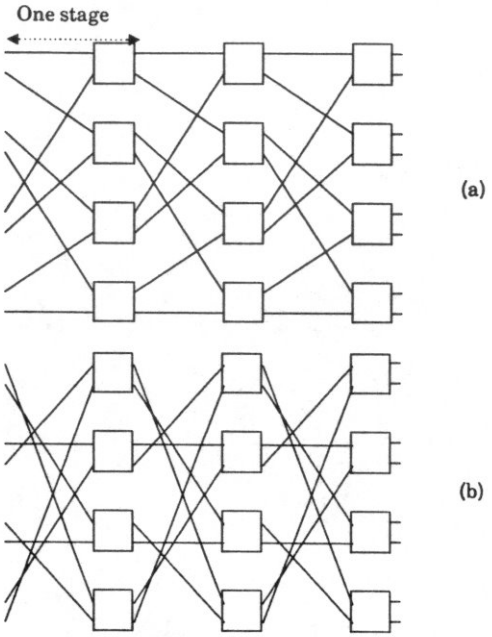
7.1 Modular Banyan Networks

To prove modular D-controllable log$N$-stage networks are all strongly equivalent to Ω, it is easier to do the analysis on the representing banyans of modular networks.

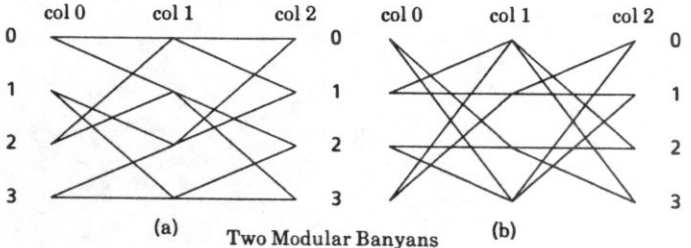
Definition 7.1: The class of representing banyans of the "generalized recursive networks" is called the class of *generalized recursive banyans*.

A generalized recursive banyan G(W) is in *canonical* form if W is in canonical; otherwise, it is in *non-canonical* form. Fig. 7.3 shows two GRB-banyans in canonical form, and Fig. 7.2-a shows a GRB-banyan in non-canonical form.

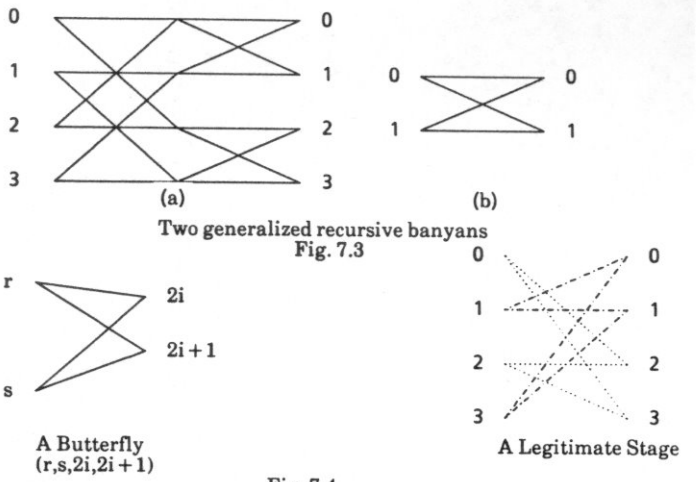
The following theorem characterizes generalized recursive banyans and is easily derived from Theorem 5.3.



Two Modular Networks
Fig. 7.1



Two Modular Banyans
Fig. 7.2



Two generalized recursive banyans
Fig. 7.3

Fig. 7.4

Theorem 7.1: Let Q be a representing banyan of size $N = 2^k$, $G_i(x,Q)$ the set of nodes of col k-i of Q that are reachable from apex x, and $R(i,j) = \{j2^i, j2^i+1, j2^i+2, \ldots, j2^i + 2^i-1\}$.

Q is generalized recursive banyan if and only if $G_i(x,Q) = G_i(y,Q)$ for all i, x and y where $1 \le i \le k-1$ and x and y are in $R(i,j)$ for some j, $0 \le j \le 2^{k-i-1}$.

□

Definition 7.2: A *modular banyan* is a banyan whose stages are identical.

Fig. 7.2 shows two modular banyans. Clearly, if W represents a modular MIN, then $G(W)$ is a modular banyan.

7.2 Modular Recursive Banyans

This section deals with modular banyans that are generalized recursive banyans, which will be referred to as *modular recursive banyans* (MRB). Each MRB, of size $N = 2^k$, can be denoted by T^k where T is one of its identical stages. Note that T is a bipartite graph of fixed degree 2 and (col i, col $i+1$) = T for all i.

The purpose of this section is to show that all MRB's can be derived from G(Ω) by column permutation. By permuting a column by a permutation g it is meant that node i is relabeled (or moved to position) $g(i)$ for all i.

This fact is shown by first establishing that all the different MRB's of size N can be synthesized recursively from smaller MRB's in a specified manner, yielding that the number of these MRB's is $2^{N\cdot 2}$. Next it is shown that $2^{N\cdot 2}$ different MRB's can be derived from G(Ω) by permuting its columns by some $2^{N\cdot 2}$ different permutations, each MRB corresponding to one permutation by which all the columns of G(Ω) are permuted. It is then concluded that all the MRB's are derivable from G(Ω) by column permutation.

The major steps to show the recursive synthesis of MRB's involve observing first that a stage of an MRB is a collection of butterflies (as in Fig. 7.4) on the one hand, and a collection of double butterflies on the other hand (as in Fig. 7.5). Then, the stage T of any MRB of size N is shown to reduce to a stage T' of height $N/2$ by collapsing each double butterfly of T to a butterfly. $T'^{k\cdot 1}$ is proved to be an MRB of size $N/2$. The converse is shown afterwards. Each stage T of height N is shown to expand to one of many possible stages of height $2N$ by expanding each butterfly of T to a double butterfly of any of the forms in Fig. 7.5. Each of these stages if duplicated $k+1$ times is proved to produce an MRB of size $2N$. This stage reduction and expansion yields that any MRB can be expanded from one single butterfly.

These steps are elaborated next.

Definition 7.3: If T is such that T^k is MRB of size 2^k, then T is said to be *legitimate*.

Three special transformations will be used often, namely the perfect shuffle S_N (or simply S when no ambiguity arises), where $N = 2^k$, the exchange function (E) and the identity permutation (e).

$E(a_{k\cdot 1}a_{k\cdot 2}\ldots a_0) = a_{k\cdot 1}a_{k\cdot 2}\ldots a_1 b_0$ where $a_{k\cdot 1}a_{k\cdot 2}\ldots a_0$ is a binary number and b_0 is the complement of a_0. Alternately, $E(2i) = 2i+1$ and $E(2i+1) = 2i$.

7.2.1 Structure of legitimate stages

This subsection studies the stage structure of MRB's, derives a permutational representation for it, and shows that a legitimate stage consists of *butterflies* (Fig. 7.4) on the one hand, and *double butterflies* (Fig. 7.5) on the other hand.

Lemma 7.1: If Q is a generalized recursive banyan of size $N = 2^k$ and T its last stage (col $k\text{-}1$, col k), and if (x,y) is in T, then $(x, E(y))$ is in T too.

Proof: Follows immediately from the definition of GRN and generalized recursive banyans. □

Consequently, T is a collection of *butterflies* $(r,s,2i,2i+1)$, as shown in Fig. 7.4.

Lemma 7.2: If T is a legitimate stage of size $N = 2^k$, then T can be represented by two permutations f and $g = fE$ such that $(r, f(r))$ and $(r, g(r))$ are in T, and $f(r) \neq g(r)$ for all $r = 0, 1, \ldots, N\text{-}1$.

Proof: Since T is a collection of butterflies of the form $(r,s,2i,2i+1)$, let $f(r) = 2i, f(s) = 2i+1, g(r) = 2i+1$ and $g(s) = 2i$. □

T can now be identified by $\{f,g\}$, and even by f only because $g = fE$. We write $T = \{f,g\}$ or $T = (f)$.

Lemma 7.3: If $T = \{f,g\}$ and if the left column of T is permuted by h and its right column by l, then the resulting stage $T' = \{h^{-1}fl, h^{-1}gl\}$.

The proof is straightforward.

Lemma 7.4: Let $T = \{f, g = fE\}$ be a legitimate stage of size $N = 2^k$, and let $Q = T^k$. Permute col i of Q by f^{k-i} for $i = 1, 2, \ldots, k-1$ and let P be the resulting banyan. Then, $(\text{col } k\text{-}1, \text{col } k)_P = \{e, E\}$ and $(\text{col } k\text{-}2, \text{col } k\text{-}1)_P = \{e, f^{-1}Ef\}$.

Proof: Since col k-1 of Q is permuted by f and $(\text{col } k\text{-}1, \text{col } k)_Q = \{f,g\}$, $(\text{col } k\text{-}1, \text{col } k)_P = \{f^{-1}f, f^{-1}fE\} = \{e, E\}$, after Lemma 7.3. As col k-2 of Q is permuted by f^2 and $(\text{col } k\text{-}2, \text{col } k\text{-}1)_Q = \{f,g\}$, $(\text{col } k\text{-}2, \text{col } k\text{-}1)_P = \{f^2f, f^2fE\} = \{e, f^{-1}Ef\}$. □

Lemma 7.5: If $T = (f)$ is a legitimate stage of size N, then $(\{4i,4i+1\})\ f^{-1}Ef = \{4i+2,4i+3\}$ and $(\{4i+2,4i+3\})\ f^{-1}Ef = \{4i,4i+1\}$.

Proof: Let $Q = T^k$ and P as in Lemma 7.4. As Q is a generalized recursive banyan, so must P be. By Theorem 7.1, $G_2(4i,P) = G_2(4i+1,P) = G_2(4i+2,P) = G_2(4i+3,P)$ because $R(2,i) = \{4i, 4i+1, 4i+2, 4i+3\}$ (refer to Theorem 7.1 for notation). Moreover, $|G_2(4i,P)| = 4$. As $(\text{col } k\text{-}1, \text{col } k)_P = \{e,E\}$ and $(\text{col } k\text{-}2, \text{col } k\text{-}1)_P = \{e, f^{-1}Ef\}$, x is in $G_2(x,P)$ for all x due to the identity permutation e. Consequently, $G_2(4i,P) = \{4i, 4i+1, 4i+2, 4i+3\}$.

On the other hand, $G_2(4i,P) = e^{-1}(G_1(4i,P)) \cup (f^{-1}Ef)^{-1}(G_1(4i,P))$ because $(\text{col } k\text{-}2, \text{col } k\text{-}1)_P = \{e, f^{-1}Ef\}$. By virtue of the fact that $G_1(4i,P) = e^{-1}(4i) \cup E^{-1}(4i) = \{4i, 4i+1\}$ and $(f^{-1}Ef)^{-1} = f^{-1}Ef$, we conclude that $G_2(4i,P) = \{4i, 4i+1, (f^{-1}Ef)^{-1}(4i), (f^{-1}Ef)^{-1}(4i+1)\}$. Therefore, $\{4i, 4i+1, 4i+2, 4i+3\} = \{4i, 4i+1, (f^{-1}Ef)^{-1}(4i), (f^{-1}Ef)^{-1}(4i+1)\}$.

Hence, $(\{4i,4i+1\})\ f^{-1}Ef = \{4i+2,4i+3\}$. Since $(f^{-1}Ef)^{-1} = f^{-1}Ef$, we conclude that $(\{4i+2,4i+3\})\ f^{-1}Ef = \{4i,4i+1\}$. □

Hence, if $f(2r)$ is in $\{4i,4i+1\}$, then $f(2r+1)$ which is equal to $(f(2r))f^{-1}Ef$ must be in $\{4i+2,4i+3\}$ and conversely. This leads to the following proposition.

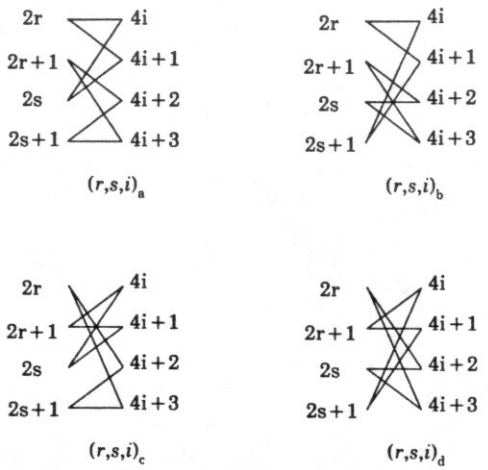
Proposition 7.1: If T is a legitimate stage, then T is a collection of double butterflies of one of the four forms $(r,s,i)_a$, $(r,s,i)_b$, $(r,s,i)_c$, and $(r,s,i)_d$ shown in Fig. 7.5.

This structure of legitimate stages T will be used to define two operations on T, *reduction* and *expansion*. The reduction operation reduces T into a stage T' of half the size of T, and the expansion operation expands T to a stage T'' of double the size of T.

7.2.2 Legitimate stage synthesis

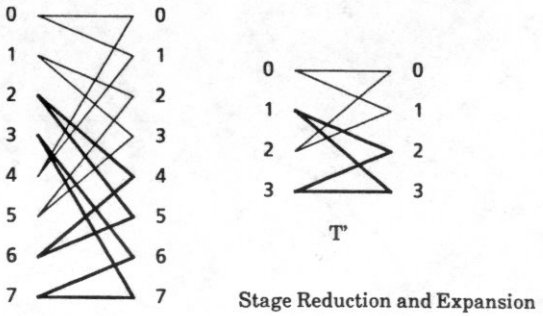
Stage Reduction: Let T be a legitimate stage of size N. T is a collection of double butterflies of any of the four forms a, b, c or d, of different parameters r, s, and i. Reduce each double butterfly (r,s,i) of T to a butterfly $(2r, 2s, 2i, 2i+1)$. Let T' be the collection of the resulting butterflies. T' is of size $N/2$. Fig. 7.6 shows a stage T and its reduced stage T'.

Stage Expansion: Let T be a legitimate stage of size N. T is a collection of butterflies $(2r, 2s, 2i, 2i+1)$ of different parameters r, s and i. Expand each butterfly $(2r, 2s, 2i, 2i+1)$ of T into a DB (r,s,i) of any of the four forms a, b, c or d. Let T'' be the collection of the resulting DB's. T'' is clearly a bipartite graph of fixed degree 2, and each column of T'' has $2N$ nodes. In Fig. 7.6, T is one of the expansions of T'.



Double Butterflies

Fig. 7.5



Stage Reduction and Expansion

Fig. 7.6

Note that many different stages T'' can be expanded from T. Precisely, $4^{N/2}$ different T'' can be derived if T is of size N because T has $N/2$ butterflies and each butterfly can be expanded to four DB's.

It will be shown later that if T is a legitimate stage of size $N = 2^k$ (i.e., T^k is an MRB), and if T' (resp. T'') is the reduced stage (resp. an expansion) of T, then, T'^{k-1} and T''^{k+1} are two MRB's of size $N/2$ and $2N$, respectively. We can then conclude that legitimate stage T can be expanded from the one-butterfly stage shown in Fig. 7.3-b, by applying the expansion operation a number of times.

Denote by Σ_N the number of all the different MRB's of size N. It is obvious that $\Sigma_2 = 1$ because the only MRB of size 2 is that shown in Fig. 7.3-b. Since the MRB's of size N can be expanded from the MRB's of size N/2 and since $4^{N/4}$ stages can be expanded from each legitimate stage of size $N/2$, it can be concluded that $\Sigma_N \leq 4^{N/4}\,\Sigma_{N/2}$. It is "$\leq$" rather than "$=$" because it is not yet evident that no MRB can be expanded using two different expansion paths.

Using this recurrence inequality, it can be concluded that $\Sigma_N \leq 2^N/4 = 2^{N-2}$.

Denote by $T_s = (S)$, where S is the perfect shuffle. Note that $G(\Omega) = T_s^k$. It will be shown later that 2^{N-2} different MRB's can be derived from $T_s^k = (S)^k$ by permuting the columns of T_s^k by the same permutation, generating different MRB's from different permutations.

7.2.3 Legitimacy of reduced stages

This subsection shows that the reduced stage T' of a legitimate stage T is legitimate. It proceeds by proving first that T'^{k-1} is a banyan network, then it shows that T'^{k-1} is an MRB relying on the characterization theorem of GRB's (Theorem 7.1).

Lemma 7.6: If T is a stage of size $N = 2^k$ and if there is at least one path between any base and any apex of T^k, then T^k is a banyan.

Proof: The number of apexes reachable from any base is N. Therefore, the nodes reachable from any base x of T^k should form a full binary tree, for otherwise, the number of nodes at depth k (i.e., apexes) would be less than 2^k. Hence, T^k consists of superposed full binary trees, and is thus a banyan. □

Lemma 7.7: If T is a legitimate stage and T' is the reduced stage from T, and if (x,y) is in T, then $(\lfloor x/2 \rfloor, \lfloor y/2 \rfloor)$is in T'.

Proof: It follows directly from the rule of reduction of double butterflies to butterflies. □

Lemma 7.8: If T is a legitimate stage and T'' is an expansion of T, and if (x,y) is in T, then $(2x, 2y)$ or $(2x+1, 2y)$ is in T''.

Proof: It follows directly from the rule of expansion of butterflies to double butterflies. □

Theorem 7.2: If $T = (f)$ is a legitimate stage of size $N = 2^k$ and T' is the reduced stage from T, then T'^{k-1} is banyan.

Proof: By Lemma 7.6, it is enough to show that there is at least one path between each base and each apex of T'^{k-1}. Fix such a pair s and d, and let $x = f^{-1}(2s)$ and $y = 2d$ be a base-apex pair in T^k. As T^k is banyan, there is a path $x_0 = x, x_1, x_2, \ldots, x_k = y$ between x and y in T^k. Let $y_i = \lfloor x_i/2 \rfloor$. Since (x_i, x_{i+1}) is in T for all i, (y_i, y_{i+1}) must be in T' (by Lemma 7.7). Hence, $y_1, y_2, \ldots, y_k$ is a path in T'^{k-1}.

$2s = f(x_0)$ and $x_1 = f(x_0)$ or $e(f(x_0)$ because $T = \{f, fE\}$. This yields that $x_1 = 2s$ or $2s+1$. In either case, $\lfloor x_1/2 \rfloor = s$ and thus $y_1 = s$. $y_k = \lfloor x_k/2 \rfloor = \lfloor y/2 \rfloor = \lfloor 2d/2 \rfloor = d$. Therefore, $s = y_1, y_2, \ldots, y_k = d$ is a path between s and d in T'^{k-1}. □

Lemma 7.9: If T is a legitimate stage of size $N=2^k$ and T' its reduced stage, then for all x and i, $G_i(x,T'^{k-1}) = \{ \lfloor y/2 \rfloor \mid y \in G_i(2x,T^k) \}$.

Proof: If $y \in G_i(2x,T^k)$, then there is a path $y_{k-i} = y, y_{k-i+1}, \ldots, y_k = 2x$ in T^k. Hence, $\lfloor y_{k-i}/2 \rfloor = \lfloor y/2 \rfloor, \lfloor y_{k-i+1}/2 \rfloor, \ldots, \lfloor y_k/2 \rfloor = x$ is a path in T'^{k-1}. Thus, $\lfloor y/2 \rfloor \in G_i(x,T'^{k-1})$. On the other hand, if r is in $G_i(x,T'^{k-1})$, then there is a path $z_{k-i-1} = r, z_{k-1}, \ldots, z_{k-1} = x$ of length i in T'^{k-1}. Let $x_k = 2x$. As $(z_{k-2}, z_{k-1}=x) \in T'$, it follows that $(2z_{k-2}, 2x)$ or $(2z_{k-2}+1, 2x)$ is in T by Lemma 7.8. Let $x_{k-1} = 2z_{k-2}$ or $2z_{k-2}+1$ such that (x_{k-1}, x_k) is in T.

In a similar way, $x_{k-2}, x_{k-3}, \ldots, x_{k-i}$ can be inductively defined in such a way that the edge (x_l, x_{l+1}) is in T for $l = k-2, k-3, \ldots, k-i$, and $x_l = 2z_{l-1}$ or $2z_{l-1}+1$. Consequently, $x_{k-i}, x_{k-i+1}, \ldots, x_k = 2x$ is a path of length i in T^k. Thus, $x_{k-i} \in G_i(2x,T^k)$.

Now since $x_{k-i} = 2z_{k-i-1}$ or $2z_{k-i-1}+1$, we conclude that $x_{k-i} = 2r$ or $2r+1$ and therefore $r = \lfloor x_{k-i}/2 \rfloor$. Hence, $r \in \{ \lfloor y/2 \rfloor \mid y \in G_i(2x,T^k) \}$. □

Lemma 7.10: If $T = (f)$ is a legitimate stage of size $N=2^k$ and $x, y \in R(i,j)$ for some i and j, then $G_i(2x,T^k) \cup E(G_i(2x,T^k)) = G_i(2y,T^k) \cup E(G_i(2y,T^k))$.

Proof: Let $g = fE$. $G_{i+1}(2x,T^k) = f^{-1}(G_i(2x,T^k)) \cup g^{-1}(G_i(2x,T^k)) = f^{-1}\big(G_i(2x,T^k) \cup E(G_i(2x,T^k))\big)$. Similarly, $G_{i+1}(2y,T^k) = f^{-1}\big(G_i(2y,T^k) \cup E(G_i(2y,T^k))\big)$.

$x, y \in R(i,j) \Rightarrow 2x, 2y \in R(i+1,j)$. Thus, $G_{i+1}(2x,T^k) = G_{i+1}(2y,T^k)$ due to Theorem 7.1 and the fact that T^k is a generalized recursive banyan.

Hence, $f^{-1}\big(G_i(2x,T^k) \cup E(G_i(2x,T^k))\big) = f^{-1}\big(G_i(2y,T^k) \cup E(G_i(2y,T^k))\big)$ and consequently, $G_i(2x,T^k) \cup E(G_i(2x,T^k)) = G_i(2y,T^k) \cup E(G_i(2y,T^k))$ because f is a permutation. □

Theorem 7.3: If T is a legitimate stage of size $N=2^k$ and T' its reduced stage, then T'^{k-1} is an MRB.

Proof: By Theorem 7.2, T'^{k-1} is banyan. It is obviously modular. It remains to show it is a generalized recursive banyan. To this end, we need to show that $G_i(x,T'^{k-1}) = G_i(y,T'^{k-1})$ for x, y in $R(i,j)$. Let r be in $G_i(x,T'^{k-1})$. By Lemma 7.9, there exists z in $G_i(2x,T^k)$ such that $r = \lfloor z/2 \rfloor$.

Using Lemma 7.10, we get $G_i(2x,T^k) \cup E(G_i(2x,T^k)) = G_i(2y,T^k) \cup E(G_i(2y,T^k))$. Therefore, z is in $G_i(2y,T^k)$ or $E(G_i(2y,T^k))$ and thus z or $E(z)$ is in $G_i(2y,T^k)$. As $r = \lfloor z/2 \rfloor$ and $\lfloor E(z)/2 \rfloor = \lfloor z/2 \rfloor$, r must be in $\{ \lfloor m/2 \rfloor \mid m \in G_i(2y,T^k) \} = G_i(y,T'^{k-1})$. Hence, $G_i(x,T'^{k-1}) \subseteq G_i(y,T'^{k-1})$. One can similarly prove that $G_i(y,T'^{k-1}) \subseteq G_i(x,T'^{k-1})$. Thus, $G_i(x,T'^{k-1}) = G_i(y,T'^{k-1})$. □

7.2.4 Legitimacy of expanded stages

The converse of the previous theorem can be proved with the same line of reasoning. That is, if T is a legitimate stage of size $N = 2^k$ and if T''' is an expansion of T, then T'''^{k+1} is a modular recursive banyan of size $2N$. As the proof to this is very similar to that of the previous theorem, it will not be presented.

This ends the proof of an earlier statement that any MRB can be expanded from the one-stage banyan of size 2, shown in Fig. 7.3-b, and that $\Sigma_N \le 2^{N-2}$.

7.2.5 MRB derivation from G(Ω)

This subsection shows that $2^N/4$ different MRB's can be derived from G(Ω) by permuting the latter's columns identically by some $2^N/4$ different permutations. These permutations will be defined recursively, and will have the property that if any of them is used to permute the nodes of the rightmost column of a GRB-banyan, the network remains in GRB.

Definition 7.4: H_N is a subset of permutations of $\{0, 1, \ldots N-1\}$ defined recursively. $H_2 = \{e_2, E_2 = (0\ 1)\}$, E_2 is the exchange permutation of $\{0,1\}$.

$H_N = H_{N/2} \times H'_{N/2} \cup H''_{N/2} \times H'''_{N/2}$ where the permutations of $H'_{N/2}$, $H''_{N/2}$ and $H'''_{N/2}$ are the same as those of $H_{N/2}$ except that the permutations of $H'_{N/2}$ map $\{N/2, N/2+1, \ldots, N-1\}$ to itself, those of $H''_{N/2}$ map $\{0,1, \ldots, N/2\}$ to $\{N/2, N/2+1, \ldots, N-1\}$ and those of $H'''_{N/2}$ map $\{N/2, N/2+1, \ldots, N-1\}$ to $\{0,1, \ldots, N/2\}$. A permutation α in $H_{N/2} \times H'_{N/2}$ is of the form $\alpha = (\alpha_1,\alpha_2)$ such that α_1 is in $H_{N/2}$, α_2 is in $H'_{N/2}$, and $\alpha(i) = \alpha_1(i)$ or $\alpha_2(i)$ depending on whether i is in $\{0,1, \ldots, N/2\}$ or $\{N/2, N/2+1, \ldots, N-1\}$. See Fig. 7.7 for a pictorial representation of H_N. Fig. 7.8 shows a few permutations in H_4 and H_8.

The main feature of H_N is that if the rightmost column of a generalized recursive banyan is permuted by a permutation in H_N, the network remains a generalized recursive banyan. This can be shown by simple induction.

It is clear that $|H_{N/2}| = |H'_{N/2}| = |H''_{N/2}| = |H'''_{N/2}|$ and that $|H_N| = 2|H_{N/2}|^2$. Let $R_N = H_{N/2} \times H'_{N/2}$. $|R_N| = |H_N|/2$.

The following two lemmas are straightforward to prove by induction.

Lemma 7.11: $|H_N| = 2^N/2$ and $|R_N| = 2^N/4$.

Lemma 7.12: If g is in H_N, then $gE = Eg$.

Lemma 7.13: If g is in R_N and $g^{-1}S_Ng = S_N$, then $g = e$.

Proof: S is used in place of S_N to simplify the notation. Let $i \le N/2 - 1$.

Then $S(i) = 2i$ and $g(i) \le N/2 - 1$. $g^{-1}Sg = S \Rightarrow gS = Sg \Rightarrow (i)Sg = (i)gS \Rightarrow$

$g(S(i)) = S(g(i)) \Rightarrow g(2i) = 2g(i)$.

$gE = Eg \Rightarrow g(2i+1) = 2g(i) + 1$.

By simple induction on $l \le N-1$, it can now be shown that $g(l) = l$. □

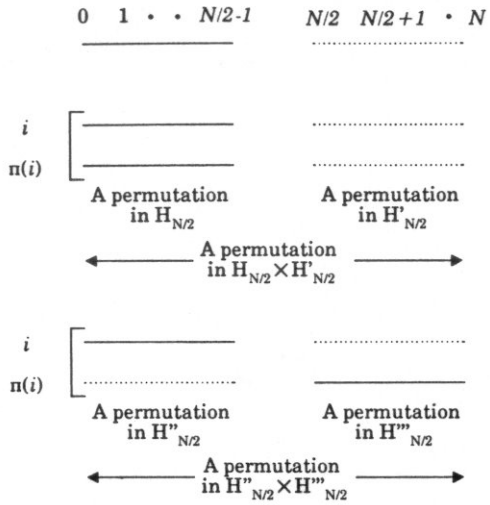


Fig. 7.7
General Recursive Structure of H_N

$$\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}\begin{pmatrix}0&1&2&3\\3&2&0&1\end{pmatrix}\begin{pmatrix}0&1&2&3\\3&2&1&0\end{pmatrix}$$

Permutations in H_4
(a)

$$\begin{pmatrix}0&1&2&3&4&5&6&7\\1&0&3&2&5&4&6&7\end{pmatrix}\begin{pmatrix}0&1&2&3&4&5&6&7\\5&4&6&7&0&1&2&3\end{pmatrix}\begin{pmatrix}0&1&2&3&4&5&6&7\\5&4&6&7&2&3&0&1\end{pmatrix}\begin{pmatrix}0&1&2&3&4&5&6&7\\5&4&7&6&2&3&1&0\end{pmatrix}$$

In H_4 In H'_4 In H''_4 In H'''_4 In H''_4 In H'''_4 In H''_4 In H'''_4

In $H_4 \times H'_4$ In $H''_4 \times H'''_4$ In $H''_4 \times H'''_4$ In $H''_4 \times H'''_4$

Permutations in H_8
(b)
Fig. 7.8

Lemma 7.14: If g_1 and g_2 are in R_N and $g_1^{-1}S_N g_1 = g_2^{-1}S_N g_2$, then $g_1 = g_2$.

Proof: $g_1^{-1}S_N g_1 = g_2^{-1}S_N g_2 \Rightarrow (g_1 g_2^{-1})^{-1}S_N(g_1 g_2^{-1}) = S_N \Rightarrow g_1 g_2^{-1} = e \Rightarrow g_1 = g_2$. □

Let $E = \{ T^k \mid T = (g^{-1}S_N g)$ for some g in $R_N\}$.

Note that each T^k in E is a modular banyan because it can be derived from T_s^k, where $T_s = (S_N)$, by permuting the latter's columns by g. Moreover, T^k is a generalized recursive banyan because T_s^k is a generalized recursive banyan and g is in H_N. Therefore, the networks of E are all modular recursive banyans.

Lemma 7.15: $|E| = |R_N| = 2^N/4$.

Proof: Follows immediately from Lemma 7.14 and 7.11. □

Theorem 7.4: Every MRB of size $N = 2^k$ can be derived from T_s^k by permuting all the latter's columns by some permutation g in R_N.

Proof: $|E| = 2^N/4 \geq \Sigma_N \Rightarrow |E| = \Sigma_N$ because Σ_N is the number of all MRB's. Hence, every MRB is in E and therefore can be derived T_s^k by permuting all the latter's columns by some permutation g in R_N. □

Corollary 7.1: If T^k is an MRB, then $T = (f)$ for some f such that $f^k = e$.

Proof: T^k is in E. Therefore, $T = (g^{-1}S_N g)$ for some g in R_N. Let $f = g^{-1}S_N g$.
$f^k = (g^{-1}S_N g)^k = g^{-1}(S_N)^k g = g^{-1}eg = e$. □

Henceforth, when a legitimate stage T is represented by (f), f is assumed to satisfy $f^k = e$.

Theorem 7.4 will lead to the main results of this chapter, namely that all modular D-controllable networks are strongly equivalent to Ω, and all modular FD-controllable networks are conjugately equivalent to Ω.

7.3 D- & FD-controllable Modular Networks and their Equivalence to Ω

The goal of this section is to show that all D-controllable modular networks are strongly equivalent to Ω and all FD-controllable modular networks are conjugately equivalent to Ω. First we show in Lemma 7.17 that any MRB Q can be put in the form $X(Q_1,Q_2)$ by permuting its internal columns, where X is the first stage and Q_1 and Q_2 are both MRB's of half the size of Q. Using this "second" recursive form $X(Q_1,Q_2)$ of MRB's, we show inductively that any D-controllable modular network is strongly equivalent to Ω. These steps are presented below with some supporting lemmas.

Lemma 7.16: If $T = (f)$ is a legitimate stage of size $N = 2^k$ such that $f^k = e$, then $f(i)$ is even (resp., odd) for all $i \le N/2 - 1$ (resp., $i \ge N/2$).

Proof: There exists a modular D-controllable network W in GRN_{2N} such that $G(W) = T^k$. The path $i, f(i), f^2(i), \ldots, f^k(i)$ goes from switch i of col 0 of W to switch $f^k(i) = i$ of col k.

Since W is in GRN, the right upper (resp., lower) links of the switches of col 0 of W lead to switches of labels $\le N/2 - 1$ (resp., $i \ge N/2$) in col k. Hence, $(i,f(i))$ is an upper (resp., lower) right link of a switch in col 0 of W if $i \le N/2 - 1$ (resp., $i \ge N/2$).

On the other hand, in the last stage (col $k-1$, col k) of W, the right upper (resp., lower) links go to even-labeled (resp., odd-labeled) switches because W is in GRN.

Since the first stage and the last stage of W are identical, the lemma follows. □

Let σ_N (or simply σ if no ambiguity arises) be the following sub-shuffle:

$\sigma(a_{k-1}a_{k-2}a_{k-3}\ldots a_0) = a_{k-1}a_{k-3}\ldots a_0a_{k-2}$ where $N = 2^k$. Clearly, $\sigma(i) \le N/2 - 1$ if $i \le N/2 - 1$ and $\sigma(i) \ge N/2 - 1$ if $i \ge N/2 - 1$. Therefore, σ can be written as (ϕ,θ) where $\phi(0a_{k-2}a_{k-3}\ldots a_0) = 0a_{k-3}\ldots a_0a_{k-2}$ and $\theta(1a_{k-2}a_{k-3}\ldots a_0) = 1a_{k-3}\ldots a_0a_{k-2}$. Note that ϕ and θ are the same permutation except that the former maps $\{0, 1, \ldots, N/2 - 1\}$ to itself and the latter maps $\{N/2, N/2+1, \ldots, N-1\}$ to itself.

Lemma 7.17: If $T = (g^{-1}S_Ng)$ is a legitimate stage of size $N = 2^k$ and $g = (g_1,g_2)$ is in R_N, and if col 1, col 2, ... , col $k-1$ of T^k are permuted by $g^{-1}S^{k-1}\sigma^{-(k-1)}g$, $g^{-1}S^{k-2}\sigma^{-(k-2)}g$, ... , $g^{-1}S\sigma^{-1}g$, respectively, then T^k transforms into the form $Q = X(Q_1,Q_2)$ such that $Q_1 = T_1^{k-1}$ and $Q_2 = T_2^{k-1}$, of size $N/2$, where $T_1 = (g_1^{-1}\phi g_1)$ and $T_2 = (g_2^{-1}\theta g_2)$.

Proof: $T = \{g^{-1}S_Ng, g^{-1}S_NgE\}$. By Lemma 7.3, $(\text{col } i, \text{col } i+1)_Q$ has the following two permutations: $(g^{-1}S^{k-i}\sigma^{-(k-i)}g)g^{-1}S_Ng(g^{-1}S^{k-i-1}\sigma^{-(k-i-1)}g)$ and $(g^{-1}S^{k-i}\sigma^{-(k-i)}g)g^{-1}S_NgE(g^{-1}S^{k-i-1}\sigma^{-(k-i-1)}g)$. By algebraic manipulation, it can be shown that the first permutation is $g^{-1}\sigma g$ and the second is $g^{-1}\sigma gE$ for all $i = 1, 2, \ldots, k-1$. As $g = (g_1,g_2)$ and $\sigma = (\phi,\theta)$, we get $g^{-1}\sigma g = (g_1^{-1}\phi g_1, g_1^{-1}\theta g_1)$ and $g^{-1}\sigma gE = (g_2^{-1}\phi g_2E, g_2^{-1}\theta g_2E)$ and the lemma follows. □

Let β_N be the following permutation of $\{0,1,\ldots,N-1\}$: $\beta_N(i) = N-1-i$.

Lemma 7.18: If g is in H_N and $g^{-1}S_Ng = S_N$, then $g = e$ or $g = \beta_N$.

Proof: If g is in $H_{N/2} \times H'_{N/2} = R_N$, then $g = e$ after Lemma 7.13.

If g is in $H''_{N/2} \times H'''_{N/2}$, then it can be easily shown that $\beta_N g$ is in $H_{N/2} \times H'_{N/2}$ and $\beta_N^{-1} = \beta_N$ and $\beta_N^{-1} S_N \beta_N = S_N$. It follows that $(\beta_N g)^{-1}S_N\beta_N g = S_N$ and $\beta_N g$ is in R_N leading to $\beta_N g = e$. Hence, $g = \beta_N^{-1} = \beta_N$. □

Proposition 7.2: If W is a bare-ended D-controllable network of size $2N = 2^{k+1}$ such that $G(W)$ is an MRB $= T^k$, then $T = T_s = (S_N)$.

Proof: Let $T = (g^{-1}S_N g)$ for some g in R_N. Permute col *1*, col 2, ... , col *k-1* of W in the same way as in Lemma 7.17. W then transforms into a canonical form $Y(W_1,W_2)$ such that W_1 and W_2 are bare-ended D-controllable networks of size N each and $G(W_1) = T_1^{k-1}$ and $G(W_2) = T_2^{k-1}$ where T_1 and T_2 are as in Lemma 7.17. The proof proceeds from here by induction on k.

Basis: $k = 1$, there is only one one-stage banyan, namely that of Fig. 7.3-b which is T_s.

Induction: Assume that the lemma is true for all values up to *k-1* and it will be shown true for the value k. The inductive hypothesis applies on W_1 and W_2. Hence, $T_1 = T_s = (S_{N/2})$. On the other hand, $T_1 = (g_1^{-1}\phi g_1)$. Therefore, $T_1 = (S_{N/2}) = (g_1^{-1}\phi g_1)$ and since $\phi = S_{N/2}$, we have $g_1^{-1}S_{N/2}g_1 = S_{N/2}$. By Lemma 7.18, $g_1 = e$ or $\beta_{N/2}$. If $g_1 = \beta_{N/2}$, then $(N/2\ -1)g^{-1}S_N g = (N/2\ -1)g_1^{-1}S_N g_1 = (N/2\ -1)\beta_{N/2}^{-1}S_N\beta_{N/2} = N/2\ -1$ which is odd. This contradicts Lemma 7.16. Hence, $g_1 = e$. The same proof applies to show that $g_2 = e$. Consequently, $g = e$ and thus $T = T_s$. □

Let F be the set of permutations of $\{0, 1, \ldots, N-1\}$ that map $\{2i, 2i+1\}$ to itself for all $i \le N/2\ -1$. Thus, if f is in F, then for every i, either $f(2i) = 2i$ and $f(2i+1) = 2i+1$, or $f(2i) = 2i+1$ and $f(2i+1) = 2i$. In the latter case , f is said to *swap* $2i$ and $2i+1$.

If W is modular LogN-stage network with a left interconnection h (then all the connections between stages are h too) and if h is replaced by hf for some f in F, then the resulting network has the same control and passes the same permutations as W, because the effect of hf on h is to swap the links coming to switch i if f swaps $2i$ and $2i+1$.

Note that if two links are swapped at the right side of a switch, the control of the network changes.

Lemma 7.19: If W and W' are two modular D-controllable networks with left interconnections h and h', respectively, and if $G(W) = G(W')$, then $h' = hf$ for some f in F. In particular, $W \equiv W'$.

Proof: That is so because if a right upper (resp. lower) link of a switch i goes to a switch j in W, this link must also go from switch i to switch j in W' because $G(W) = G(W')$. It has to be an upper (resp., lower) right link of switch i of W' for otherwise the control of W' would change and W' would no longer be D-controllable. If this link "changes entrance" to switch j in W', that accounts for a swapping between $2j$ and $2j+1$. Hence the need for an f in F to make "these corrections". This link swapping does not alter the set of permutations admissible by W, that is, W and W' are strongly equivalent. □

Theorem 7.5: If W is a modular D-controllable network, then $W \equiv \Omega$.

Proof: Let h be the left interconnection of W, and W' be W without its left interconnection. Since W' is a bare-ended modular D-controllable network, $G(W') = T_s^k$ by Proposition 7.2. Hence, $G(W) = G(\Omega) = T_s^k$ and consequently, $W \equiv \Omega$ after the previous lemma. □

Lemma 7.19 and the proof of the previous theorem say more than the theorem itself. They show that every modular D-controllable network can be derived from Ω by replacing the shuffle-interconnections S of Ω by the interconnection Sf for some f in F.

Theorem 7.6: If W is a modular FD-controllable network, then W is conjugately equivalent to Ω.

Proof: Let h be the left interconnection of W and g be the control function of W (i.e., the control tag that establishes the path between source i and destination is

$g(j)$). Since the rightmost interconnection of W is the identity, it is easy to show that $g(2i+1) = g(2i)+1$. Let $f(i) = g(2i)/2$ for $i = 0, 1, .. , N/2$. Permute the columns of W by f and the input and output terminals by g. Let the resulting network be W'. It should be clear that W' is modular and right bare-ended. Moreover, W' is D-controllable. Using Theorem 7.5, we conclude that $W' \equiv \Omega$. Since W' can pass the same permutations as W by renaming the input terminals and output terminals of W' by g^{-1} (due to the definition of W'), and as $W' \equiv \Omega$, Ω should pass the same permutations as W by renaming the input and output terminals of Ω by g^{-1}. Hence the conjugate equivalence between W and Ω. □

It can be concluded from the previous theorem that FD-controllable modular networks with arbitrary left and right interconnections are weakly equivalent to Ω.

7.4 Conclusion

It has been shown that all D-controllable modular logN-stage networks are strongly equivalent to omega networks; that all FD-controllable modular networks are conjugately equivalent to omega; and that all FD-controllable modular networks with arbitrary left and right interconnections are weakly equivalent to omega. This leads to the conclusion that modular D- and FD-controllable networks do not bring us any more communication power than omega, and the search for FD-controllable modular networks that meet given communication requirements reduces to finding a relabeling of the terminals of omega..

Chapter 8

A New Approach to Fast Control of 3-Stage Benes Networks

MIN's have thus far been the focus of study. The attention will be shifted next to Benes networks due to their functional universality that MIN's lack. A new control scheme that turns three-stage Benes networks D-controllable is the subject matter of this chapter.

8.1 Introduction

The major deficiency of MIN's is their incapability of realizing all permutations. The need is then for "universal networks" that can realize all possible permutations, which are preferably control efficient.

Benes networks realize all possible permutations but their control is inefficient [BEN65]. They cannot be controlled by control tags because there are multiple paths between sources and destinations. Controlling Benes networks involves either computing the switch configurations of permutations in advance and storing them, or computing them at run time. The first way is costly in space for large systems, and the second is costly in time [WAK68].

Two different approaches have been introduced to bypass this control complexity. The first, due to Nassimi and Sahni [NAS81], consists of using destination addresses in a specified manner, and allows for efficient realization of a subset of permutations. The second, due to Lenfant [LEN78], identifies frequently used permutations, and develops a specialized control algorithm that realizes these permutations efficiently.

The first approach produces optimal control but enables only a small fraction of the possible permutations. The second approach offers more potential for larger families of permutations but still suffers speed inefficiency and implementation overhead. What is needed is an approach that combines the positive properties of both approaches and avoids their limitations.

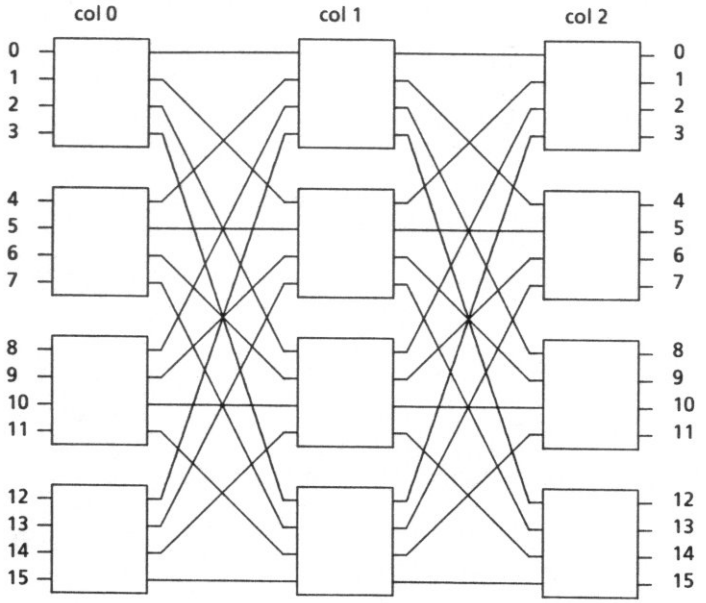
This chapter introduces a new approach to controlling 3-stage Benes networks (Fig. 8.1). It consists of *fixing the first stage of the network so that the remaining network can be self-routed* and can pass the required permutations. This approach requires that, for each class of problems, the family of permutations be identified and checked to see if the first column can be fixed so that the remaining network passes the family without conflict (such families are called *compatible*).

The compatible families of permutations are characterized, a technique to show compatibility is developed, and the families of permutations of FFT and bitonic sorting and other interesting problems are shown to be compatible.

The rest of this chapter is organized as follows. The next section gives some preliminaries and definitions. Compatibility is characterized in Section 8.3. Section 8.4 shows the compatibility of some interesting classes of permutations, including Lenfant's five families of "Frequently Used Permutations" and the families of permutations required by FFT and bitonic sorting. Finally, some concluding statements and a possible implementation of the new scheme are presented in Section 8.5.

8.2 Preliminaries and Definitions

BC(r,2) has been defined in Chapter 2. It has $N = r^2$ input terminals, N output terminals , and is composed of $r \times r$ permutation switches. This chapter will be concerned primarily with this network. Fig. 8.1 shows BC(4,2).



BC(4,2) = 16x16 3-stage Benes

Fig. 8.1

BC(r,2) has 3 columns, numbered 0, 1, 2 from left to right. (col 0, col 1) = $U_{N,r,0}$ (the perfect unshuffle) and (col 1, col 2) = $S_{N,r,0}$ (the perfect shuffle).

The r-ary representation of any number $x \le N\text{-}1$ has 2 r-ary digits. If $x = pq$ (in r-ary), $U_{N,r,0}(pq) = qp$, and $S_{N,r,0}(pq) = qp$. Therefore, $U_{N,r,0} = S_{N,r,0}$.

Denote by $f = U_{N,r,0} = S_{N,r,0}$ throughout. Thus, (col 0, col 1) = (col 1, col 2) = f. Note that $f = f^{-1}$ and $f(pr+q) = qr+p$ for all p,q where $0 \le p,q \le r\text{-}1$.

The main observation that underlies the new control approach is that if col 0 of BC(r,2) is set to some configuration, then the resulting network is in MIN(r,2) and is therefore self-routed via r-ary control tags. Note that if col 0 is set to the identity configuration, the resulting network is $\Omega(r,2)$ (see Section 2.2.1). Therefore, if col 0 of BC(r,2) is set to an arbitrary configuration h, the resulting network is $h\Omega(r,2)$ and hence D-controllable.

The main concern in this chapter is:

Given a family of permutations, can the first column of BC(r,2) be set to some configuration in such a way that the resulting network can pass the whole family without conflict?

If the answer is positive, the first column is set to the appropriate configuration and the network is self-routed thereafter for the duration of the whole family. Some new definitions and notations will be useful in dealing with these families of permutations.

Let R_N be the set $\{0, 1, \ldots, N\text{-}1\}$ and S_N the set of permutations of R_N. Any number x in R_N is uniquely written as $pr+q$ where $0 \le p, q \le r\text{-}1$.

Let H be the set of permutations (of R_N) that are realizable by any column of switches of BC(r,2).

Observe that H is a subgroup of the symmetric group S_N, and h is in H if $h(pr+q) = pr + q'$ for some q' function of p and q. Let then $q' = t(p,q)$.

If h is in H, a column of BC(r,2) is said to be set to h (or h-configured) if the p-th switch of that stage is set in such a way that the q-th input port of that switch is connected to its $h(pr+q)$-th output port.

Definition 8.1: A permutation is said to be *h-realizable* for some h in H if BC(r,2) can realize it with column 0 set to h. It is also said to be t-realizable, where $t(p,q) = h(pr+q) \bmod r$.

Definition 8.2: Let $\phi_1, \phi_2, \ldots, \phi_m$ be m permutations in S_N. $\phi_1, \phi_2, \ldots, \phi_m$ are compatible if there is some h in H such that $\phi_1, \phi_2, \ldots, \phi_m$ are all h-realizable. Then $\phi_1, \phi_2, \ldots, \phi_m$ are called *h-compatible.*

As shown in Chapter 2, the number of permutations realizable by $h\Omega(r,2)$ is $(r!)^{2r}$, which is a small fraction of the overall number $N!$ of possible permutations. However, the freedom of changing h to satisfy various needs adds versatility and more potentiality to this control scheme. It is in this freedom of selecting h where the power of the new approach lies.

8.3 Characterization of Compatible Permutations

Each class of problems, such as sorting, FFT, matrix computations, etc. ..., gives rise to a family of permutations needed to carry out the communication required if that class is run on a Benes-connected machine. Of most interest is to know whether a given family of permutations is h-compatible for some h in H and how to find h. The following theorem characterizes compatibility, and thus will help show the compatibility of several interesting families of permutations, including those required by FFT and bitonic sorting. The proof of the theorem, being tedious and requiring a number of lemmas, is put in Appendix III.

Theorem 8.1: Let $\phi_1, \phi_2, \ldots, \phi_m$ be m permutations in S_N. $\phi_1, \phi_2, \ldots, \phi_m$ are compatible iff there is $t : R_r \times R_r \rightarrow R_r$ such that
(i) $t(p,.)$ is a permutation of R_r for every p in R_r.

(ii) If for some i, $\alpha_i(p,q) = \alpha_i(p',q')$ and $p \neq p'$, then $t(p,q) \neq t(p',q')$, where $\alpha_i(p,q) = \lfloor \phi_i(pr + q)/r \rfloor$.

It can be seen from the proof of this theorem in Appendix III that $t(p,q) = h(pr+q) \bmod r$, where $\phi_1, \phi_2, \ldots, \phi_m$ are h-compatible. It follows then that a permutation ϕ is h-realizable iff (i) $t(p,.)$ is a permutation of R_r for every p in R_r, and (ii) if $\alpha(p,q) = \alpha(p',q')$ and $p \neq p'$, then $t(p,q) \neq t(p',q')$, where $\alpha(p,q) = \lfloor \phi(pr + q)/r \rfloor$ and $t(p,q) = h(pr+q) \bmod r$.

If $\alpha(p,q) = \alpha(p',q')$ implies $p = p'$, then condition (ii) is trivially satisfied, and thus ϕ is h-realizable for any h in H. In particular, the permutations in H are h-realizable for any h in H.

Theorem 8.1 can be easily mapped into a graph-theoretic problem, namely, node coloring problem. Let $\phi_1, \phi_2, \ldots, \phi_m$ be m permutations in S_N. Let $G = (V,E)$ be the following graph: $V = R_r \times R_r$ and $((p,q),(p',q')) \in E$ if

(a) $p = p'$ or

(b) $\alpha_i(p,q) = \alpha_i(p',q')$ for some i.

The theorem can now be given as follows: $\phi_1, \phi_2, \ldots, \phi_m$ are compatible iff G can be r-colored in such a way that no two adjacent nodes have the same color.
To prove this, let $t(p,q)$ be the color of (p,q). (a) is equivalent to (i) and (b) is equivalent to (ii).

The general r-coloring problem is NP-complete, but it remains open whether these graphs have any peculiarities that open the door to a fast (i.e., polynomial) algorithm to r-color them. Such an algorithm would be useful.

The characterization theorem will help to prove in the following sections that the two families of permutations required by FFT and bitonic sorting are both compatible, opening the way to efficient computation of these two interesting

problems on BC(r,2). FFT is computed efficiently on Stone's shuffle-exchange network, but the results are left at the end in bit-reversed order, and cannot be restored to natural order in one pass on that network. The new scheme takes care of that efficiently as will be shown in the next section.

Before getting to the next sections, it is worthwhile to note that not every two permutations are compatible. This is shown by a counter example using the theorem and its graph formulation.

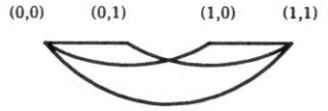


Fig. 8.2

Take $r = 2$. Thus, $N = 4$, and $R_r = \{0,1\}$. $\phi_1 = (0\ 1\ 2)(3)$ and $\phi_2 = (0)(1\ 2\ 3)$. It can be shown that

$\alpha_1(0,0) = \alpha_1(1,0) = 0$ $\qquad$ $\alpha_2(1,1) = \alpha_2(0,0) = 0$

$\alpha_1(0,1) = \alpha_1(1,1) = 1$ $\qquad$ $\alpha_2(0,1) = \alpha_2(1,0) = 1$

The corresponding graph G is depicted in Fig. 8.2. G has a 3-clique, and it cannot therefore be 2-colored. Thus, ϕ_1 and ϕ_2 are <u>not</u> compatible.

8.4 Compatible Families of Permutations

Various families of permutations will be shown compatible in this section. Some of these families include the permutations required by FFT and bitonic sorting, making these two problems fit the new control scheme of 3-stage Benes networks.

In the remainder of this chapter, r is assumed to be a power of 2 ($r = 2^k$). Thus, $N = r^2 = 2^{2\times k}$.

Any number x in R_N can be written then in r-ary and in binary. If $x = pq$ (in r-ary), then $x = p_{k-1}p_{k-2}\cdots p_0 q_{k-1}q_{k-2}\cdots q_0$ (in binary), where $p = p_{k-1}p_{k-2}\cdots p_0$ and $q = q_{k-1}q_{k-2}\cdots q_0$.

The bit positions 0, 1, ... , k-1 of x are said to form the *q-wing*, and the bit positions k, $k+1$, ... , $2k$-1 the *p-wing*. For $i = 0, 1, \ldots, k-1$, bits q_i and p_i are called *siblings*. p_i is the p-sibling of q_i, and q_i the q-sibling of p_i.

A bit permutation f_n is called a *pseudo bit translation* if the following is satisfied: f_n moves a bit from the q-wing to the p-wing iff f_n moves the p-sibling of that bit to the q-wing.

The pseudo bit translations will be shown to be compatible.

Examples of pseudo bit translations: The shuffle S is a pseudo bit translation because $S(p_{k-1}p_{k-2}\cdots p_0 q_{k-1}q_{k-2}\cdots q_0) = p_{k-2}\cdots p_0 q_{k-1}q_{k-2}\cdots q_0 p_{k-1}$, thus, S moves q_{k-1} to the p-wing and p_{k-1} to the q-wing, and it leaves the other bits in their own wings. Similarly, S^l for $l < k$ is a pseudo bit translation because it moves $q_{k-1}, q_{k-2}, \ldots, q_{k-l}$ to the p-wing, and $p_{k-1}, p_{k-2}, \ldots, p_{k-l}$ to the q-wing. For $l \geq k$, S^l can be similarly shown to be a pseudo bit translation. The unshuffle U is a pseudo bit translation because $U = S^{2k-1}$. Similarly, $U^l = S^{2k-l}$ is a pseudo bit translation. Finally, the bit reversal permutation ρ is a pseudo bit translation because $\rho(p_{k-1}\cdots p_0 q_{k-1}\cdots q_0) = q_0\cdots q_{k-1}p_0\cdots p_{k-1}$ and therefore, ρ moves all the p's to the q-wing and all the q's to the p-wing.

Theorem 8.2: All pseudo bit translations are compatible.

Proof: Let $h(pr+q) = pr + p\oplus q$ and $t(p,q) = p\oplus q$, where $p\oplus q$ is the bit-by-bit xor. It will be shown that every pseudo bit translation is h-realizable.

It is clear that $t(p,.)$ is a permutation of R_r for all p.

Let ϕ be a pseudo bit translation. Let E be the set of bit positions of the q-wing that are moved to the p-wing by ϕ. Then E is the set of p-wing bit positions that move to the q-wing. Let $\alpha(p,q) = \lfloor \phi(pr+q)/r \rfloor$ = the leftmost k bits of $\phi(pr+q)$.

$\alpha(p,q) = \alpha(p',q')$ implies that p and p' agree in all bit positions that are not in E, and q and q' agree in all bit positions in E. Thus, $q \oplus q'$ has "0" in all bit positions in E.

If moreover $p \neq p'$, then there exists i_0 in E such that $p_{i_0} \neq p'_{i_0}$. Thus, $p \oplus p'$ has "1" in bit position i_0.

$t(p,q) \oplus t(p',q') = p \oplus p' \oplus q \oplus q'$. As $p \oplus p'$ has "1" in bit position i_0 and $q \oplus q'$ has "0" in bit position i_0 (because i_0 is in E), it follows that $p \oplus p' \oplus q \oplus q'$ has "1" in bit position i_0, and therefore, $p \oplus p' \oplus q \oplus q'$ is not 0. Consequently, $t(p,q) \neq t(p',q')$ if $\alpha(p,q) = \alpha(p',q')$ and $p \neq p'$.

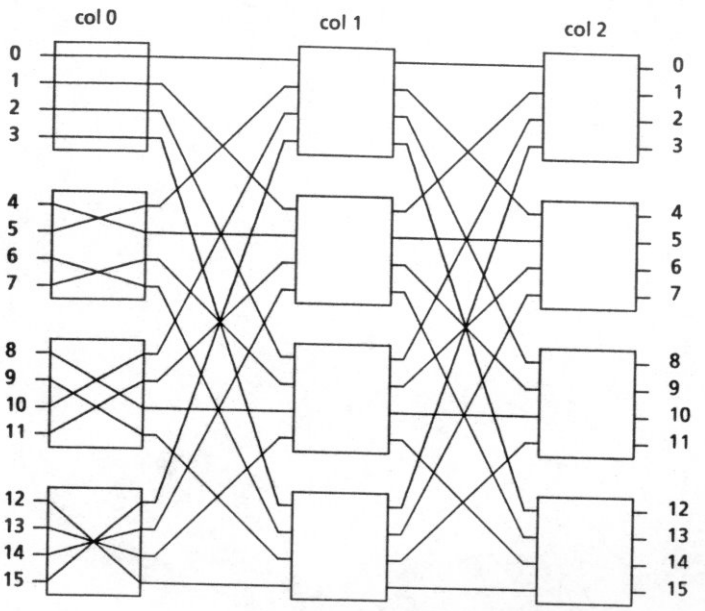
Hence, ϕ is h-realizable. As ϕ is arbitrary, all pseudo bit translations are then h-realizable and thus compatible. □

8.4.1 The Compatibility of Fast Fourier Transform

To compute FFT in the way described in [STO71], two permutations are needed: Shuffle (S), and exchange (E). However, at the end of the computation, the components of the resulting vector are in bit reversed order. To restore the order, the bit reversal permutation (ρ) is needed. Thus, the overall family of permutations needed by FFT is S, E and ρ.

As was shown earlier, S and ρ are two pseudo bit translations, and thus t-realizable where $t(p,q) = p \oplus q$. Moreover, E is t-realizable because E is in H.

Hence, ρ, S and E are compatible and FFT can be computed on BC(r,2) efficiently: Set column 0 to t at the outset of the computation of FFT, then any permutation needed (i.e., E, ρ, S) is self-routed. Fig. 8.3 shows BC(4,2) with column 0 set to h.



Configuration of col 0 for FFT
Fig. 8.3

8.4.2 Compatibility of the L-Family

In this subsection a new family L of permutations is introduced, which, as will be shown later, includes the permutations required by bitonic sorting.

Let L = {ϕ | if $\alpha_\phi(p,q) = \alpha_\phi(p',q')$ and $p \neq p'$, then p and p' agree in all but one bit position, and q and q' agree in at least one bit position}

L clearly includes every bit permutation that moves exactly one p-wing bit to the q-wing, and exactly one q-wing bit to the p-wing.

Moreover, L includes permutations that are not bit permutations. The next chapter will show a communication pattern that requires three such permutations.

It will be shown next that the permutations of L are compatible. To do so, the switches of column 0 are divided in two subsets E_k and F_k, which will be defined inductively, such that the binary numbers (i.e., the switch labels) in each subset disagree in at least two bits.

Let $E_1 = \{0\}$, $F_1 = \{1\}$ and recursively $E_i = 0E_{i-1} \cup 1F_{i-1}$ and $F_i = 0F_{i-1} \cup 1E_{i-1}$, where $0E_{i-1}$ is the set of elements of E_{i-1} after appending the bit "0" to their left, and $1E_{i-1}$ is the set of elements of E_{i-1} after appending the bit "1" to their left. $0F_{i-1}$ and $1F_{i-1}$ are defined similarly.

E_2 and F_2 are derived to clarify the definition. $0E_1 = \{00\}$, $0F_1 = \{01\}$, $1E_1 = \{10\}$, $1F_1 = \{11\}$, and consequently, $E_2 = \{00, 11\}$ and $F_2 = \{01,10\}$.

It can be easily shown by induction on $i = 1, 2, \ldots, k$ that $E_i \cup F_i = \{0,1, \ldots, 2^i\text{-}1\}$, $E_i \cap F_i = \varnothing$ and for $i > 1$, the numbers in each of the two sets E_i and F_i disagree in at least two bit positions.

In particular, $E_k \cup F_k = \{0,1, \ldots, r\text{-}1\}$ and $E_k \cap F_k = \varnothing$.

Theorem 8.3: All the permutations of L are compatible.

Proof: Let $a = 11\ldots1$ (k bits), $t(p,q) = q$ if p is in E_k, and $q \oplus a$ if p is in F_k. t is well-defined because $E_k \cap F_k = \varnothing$, and it is totally defined because $E_k \cup F_k = \{0,1, \ldots, r\text{-}1\}$. Moreover, it can be easily shown that $t(p,.)$ is a permutation of R_r for all p.

It will be shown that every permutation in L is t-realizable. Let ϕ be in L.

We have to show that if $\alpha_\phi(p,q) = \alpha_\phi(p',q')$ and $p \neq p'$, then $t(p,q) \neq t(p',q')$.

If $\alpha_\phi(p,q) = \alpha_\phi(p',q')$ and $p \neq p'$, then p and p' agree in all but one bit position, and q and q' agree in at least one bit position. Consequently, p and p' cannot both be in the same set E_k or F_k. Assume without loss of generality that p is E_k and p' is in F_k.

Thus, $t(p,q) = q$ and $t(p',q') = q' \oplus a$. It follows that $t(p,q) \oplus t(p',q') = q \oplus q' \oplus a$. As q and q' agree in at least one bit position, $q \oplus q'$ has "0" in at least one bit position and therefore, $q \oplus q' \oplus a$ has "1" in at least one position because a has all "1"'s. Hence, $t(p,q) \neq t(p',q')$.

ϕ is then t-realizable. As ϕ is arbitrary, every permutation of L must be t-realizable, and the theorem follows. □

8.4.3 The Compatibility of Bitonic Sorting

Although parallel sorting algorithms of O(log N) time have been found [AKS83], they are not practical due to the extremely large multiplicative constant of log N. Bitonic sorting, though of time complexity $O(\log^2 N)$, is a practical parallel algorithm [STO71]. This justifies the selection of bitonic sorting among all other parallel sorting algorithms to run on 3-stage Benes and to study the compatibility of its required permutations.

As shown in [STO71], N numbers can be sorted using a sorting network of $\log_2 N(\log_2 N+1)/2$ stages of comparison switches and based on Batcher's bitonic sorter [BAT68]. Simulating this sorting network on 3-stage Benes involves realizing the interconnections (i.e., permutations) between the columns of the sorting network, as well as the columns of switches themseleves.

In this subsection, bitonic sorting and the sorting network based on the bitonic sorter are reviewed briefly, the simulation of the sorting network on 3-stage Benes is specified in terms of the permutations required, and these permutations are shown compatible.

A sequence of real numbers $a_0, a_1, \ldots, a_{N-1}$ is bitonic if

1) there is an i such that $a_0, a_1, \ldots, a_i$ is monotonically increasing, and $a_i, \ldots, a_{N-1}$ is monotonically decreasing; or if

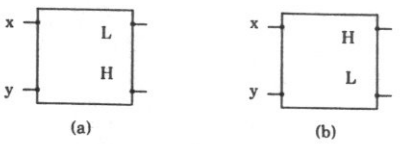
2) the sequence can be shifted cyclically so that 1) is satisfied.

An $N \times N$ bitonic sorter is a recursive network where for $N = 2$, it is the comparison switch shown in Fig. 8.4a, and for larger N, it is as depicted in Fig. 8.5. It is shown in [BAT68] and [STO71] that if the input is a bitonic sequence, then this network sorts the input.

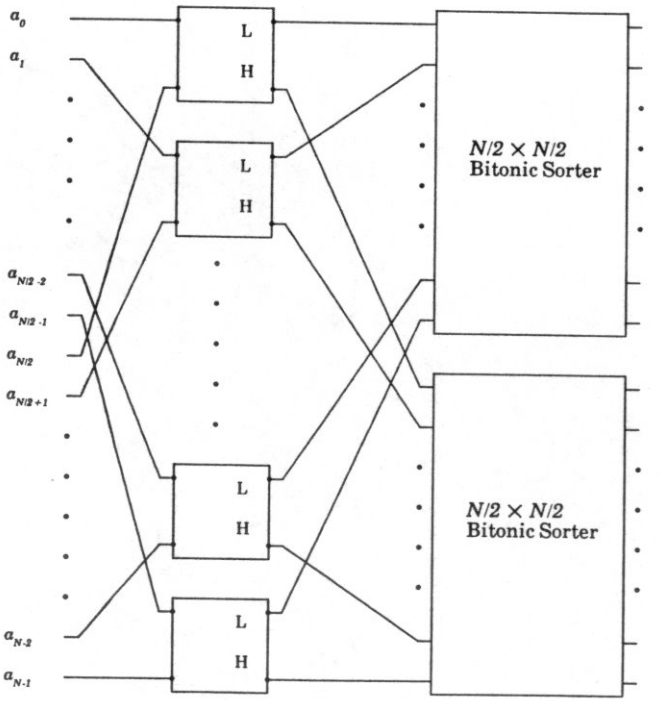
A full sorting network that sorts any sequence of $N = 2^m$ numbers can be built in m steps, where each step is a column of bitonic sorters of some size. The first step is a column of $N/2$ 2×2 bitonic sorters, the second is a column of $N/4$ 4×4 bitonic sorters, and in general, the i-th step is a column of $N/2^i$ $2^i \times 2^i$ bitonic sorters. Fig. 8.6 shows the general structure of this sorting network.

The network works as follows. The first step sorts pairs of numbers, some in increasing order and some in decreasing order so that each consecutive pair of these couples is a bitonic sequence of 4 numbers. The second step sorts these bitonic sequences, some in increasing order and some in decreasing order so that each sequence of consecutive 8 numbers is a bitonic sequence. And so on to the last step which receives a bitonic sequence of length N. Since the last step is an $N \times N$ bitonic sorter, it can sort the incoming bitonic sequence, and thus the input sequence is sorted. Fig. 8.7 shows an 8×8 sorting network, where the shaded switches place the larger item on the top output, and the blank switches place the smaller item on top.

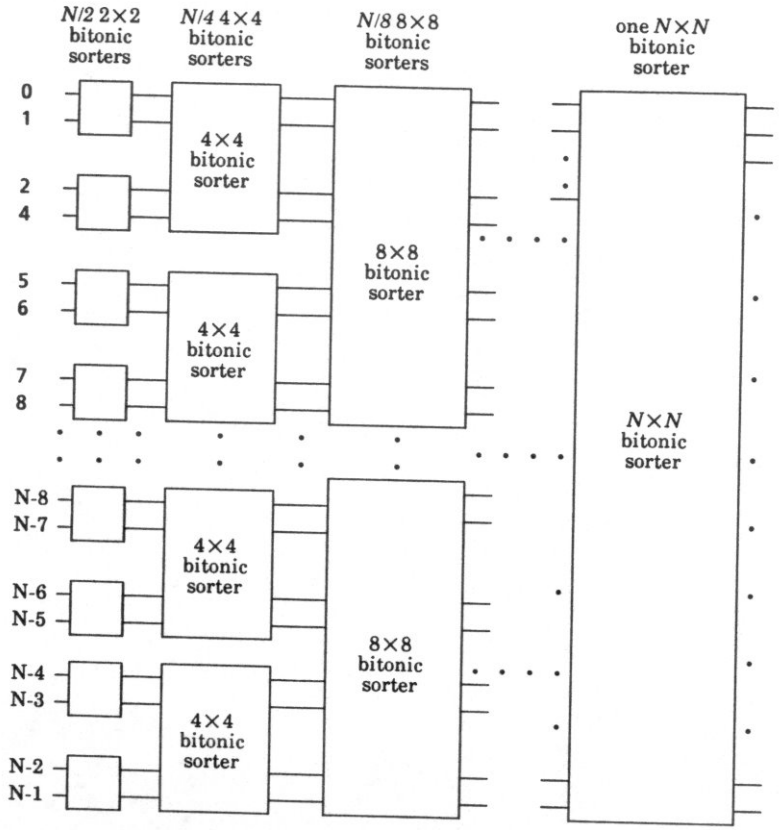
The operation of the sorting network can be viewed as a sequence of data permutations. First, the items are permuted by the first column of comparison switches, second they are permuted by the interconnection between the first column and the second column, then they are permuted by the second column of switches, and so on to the last column.



Bitonic Switch, also a
2×2 Bitonic Sorter
Fig. 8.4

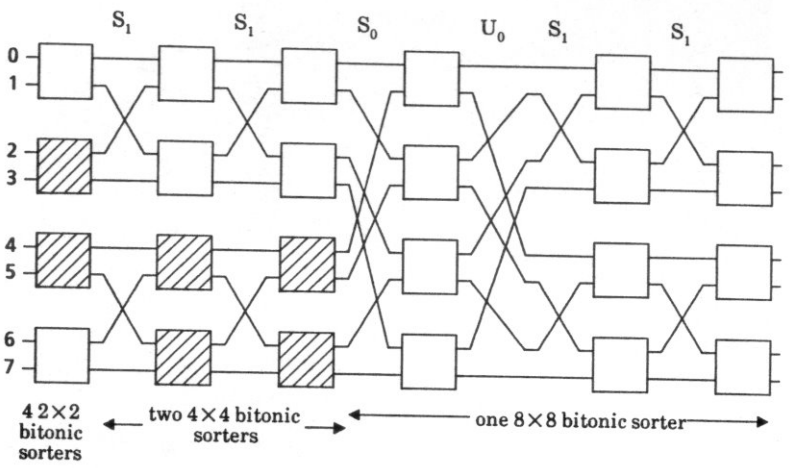


The Structure of Bitonic Sorter
Fig. 8.5



General structure of $N \times N$ sorting networks based on bitonic sorters
Fig. 8.6

Therefore, the simulation of the sorting network on Benes is done by executing the above sequence of permutations on Benes in order. At the beginning,



An 8×8 Sorting Network
Fig. 8.7

item a_i is in pe_i (i.e., the i-th input terminal of Benes, which is also the i-th output terminal). After executing a permutation ϕ, a_i is moved to $pe_{\phi(i)}$.

When simulating the permutation done by a column (column j, say) of comparison switches of the sorting network, more than permuting is required. Assume the items coming to this column are $b_0, b_1, \ldots, b_{N-1}$ from top to bottom. Then, by our simulation, these items are in $pe_0, pe_1, \ldots, pe_{N-1}$, respectively, when column j is due to be simulated. At switch i of column j, the incoming items are b_{2i} and b_{2i+1}. Hence, in simulation, these items are in pe_{2i} and pe_{2i+1}, respectively.

From the position of the column and the switch of the column, it can be known (*a priori*) whether the switch should be in state (a) or (b) of Fig. 8.4. To be able to do the comparison in simulation, each of pe_{2i} and pe_{2i+1} has to have both b_{2i} and b_{2i+1},

for all $i = 0, 1, \ldots, N\text{-}1$. This can be accomplished by exchanging the N items in Benes (i.e., executing the exchange permutation on Benes), then, if the switch is in state (a), pe_{2i} keeps $\min(x,y)$ and discards $\max(x,y)$, while pe_{2i+1} does the opposite. If in state (b), pe_{2i} keeps $\max(x,y)$ and discards $\min(x,y)$, while pe_{2i+1} does the opposite.

Now the simulation of bitonic sorting on Benes is clear, and the permutations required are the exchange and the interconnections between the columns of the sorting network. These interconnections will be found next. They will be shown to be subshuffles $S_{N,2,i}$ (denote it S_i for short) and subunshuffles $U_{N,2,i}$ (denote it U_i for short) and compositions thereof.

A close inspection of Fig. 8.5 shows that the leftmost interconnection of the bitonic sorter is the perfect shuffle S_0, and the interconnection between the first column and the other 2 $N/2 \times N/2$ bitonic sorters is the unshuffle U_0. The leftmost interconnection of these 2 $N/2 \times N/2$ bitonic sorters must be S_1. It follows then that the interconnection between the first two columns is U_0S_1. Inductively, the remaining interconnections can be shown to be $U_1S_2, U_2S_3, \ldots, U_{m-1}S_m$.

The general structure of the sorting network of Fig. 8.6 shows that the rightmost block of columns is an $N \times N$ bitonic sorter, the second rightmost block of columns consists in 2 $N/2 \times N/2$ bitonic sorters, and in general, the i-th rightmost block of columns consists in 2^{i-1} $N/2^{i-1} \times N/2^{i-1}$ bitonic sorters. It follows then that the interconnections between the columns of the sorting network are :

$S_0, U_0S_1, U_1S_2, U_2S_3, \ldots, U_{m-1}S_m$ (for the rightmost block)

$S_1, U_1S_2, U_2S_3, \ldots, U_{m-1}S_m$ (for the second rightmost block)

$S_2, U_2S_3, \ldots, U_{m-1}S_m$ (for the third rightmost block)

.....

$S_{m-1}, U_{m-1}S_m$ (for the second leftmost block, that is, the interconnection between the leftmost column and the second column, and the

interconnection between the second column and the third column of the sorting network).

The number of these interconnections is $2+3+4+\ldots+m = m(m+1)/2 - 1$.

These permutations will be shown to be in class L, and hence compatible.

Note that all the permutations off the diagonal occur in the first row. Therefore, the permutations to be considered for compatibility are $S_0, S_1, \ldots, S_{m-1}, U_0S_1, U_1S_2, U_2S_3, \ldots, U_{m-1}S_m$. Note that U_iS_{i+1} swaps the 0-th bit and the $(m-i)$-th bit because $(x_{m-1}\ldots x_1x_0)U_iS_{i+1} = (x_{m-1}\ldots x_{m-i}x_0x_{m-i-1}\ldots x_1)S_{i+1} = x_{m-1}\ldots x_{m-i+1}x_0x_{m-i-1}\ldots x_1x_{m-i}$.

We assumed that $N = 2^{2k}$, thus $m = 2k$.

For $i \geq k$, each of the S_i and $U_{i+1}S_{i+2}$ permute the bits of the q-wing but leaves each p-wing bit in place. Therefore, these S_i's and $U_{i+1}S_{i+2}$'s are in H and hence h-realizable for any h. Recall that H is the set of permutations realizable by any of the three columns of BC(r,2).

Consequently, it suffices to show that $S_0, S_1, \ldots, S_{k-1}, U_0S_1, U_1S_2, U_2S_3, \ldots, U_kS_{k+1}$ are compatible.

For all $i<k$, $S_i(p_{k-1}p_{k-2}\cdots p_0q_{k-1}q_{k-2}\cdots q_0) = p_{k-1}\cdots p_{k-i}p_{k-i-2}\cdots p_0q_{k-1}q_{k-2}\cdots q_0p_{k-i}$. Thus, S_i moves exactly one p-wing bit (which is p_{k-i}) to the q-wing, and exactly one q-wing bit to the p-wing, namely, q_{k-1}. Let $\alpha_i(p,q)$ be the leftmost k bits of $S_i(p_{k-1}p_{k-2}\cdots p_0q_{k-1}q_{k-2}\cdots q_0)$. That is, $\alpha_i(p,q) = p_{k-1}\cdots p_{k-i}p_{k-i-2}\cdots p_0q_{k-1}$.

Hence, if $\alpha_i(p,q) = \alpha_i(p',q')$ and $p \neq p'$, then p and p' agree in all bit positions except for position k-i, and q and q' agree at least in bit position k-1. Therefore, S_i is in L.

U_iS_{i+1} also moves exactly one p-wing bit (which is p_{k-i}) to the q-wing, and exactly one q-wing bit to the p-wing, namely, q_0. Thus, U_iS_{i+1} can be similarly shown to be in L for all $i<k$.

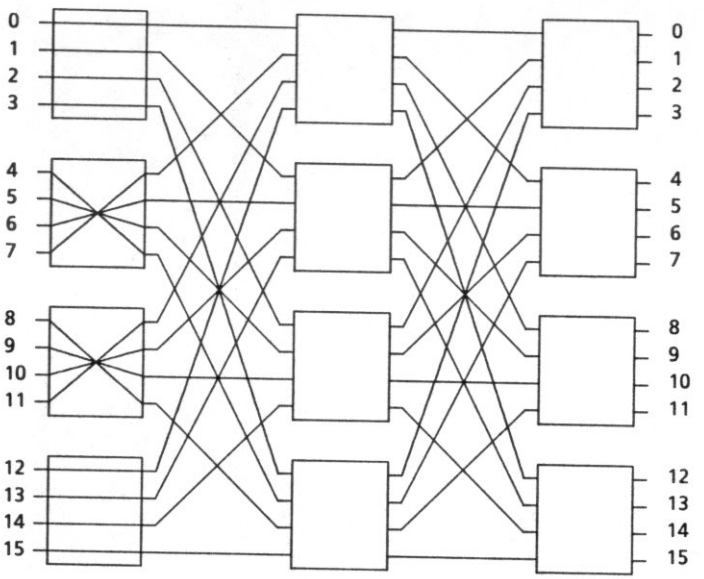
It follows then that $S_0, S_1, \ldots, S_{k-1}, U_0S_1, U_1S_2, U_2S_3, \ldots, U_kS_{k+1}$ are all in L and hence compatible.

As the permutations needed to simulate the sorting network are the exchange (E) and the interconnections between the columns, and as the exchange is in H and hence is h-realizable for any h, it follows that all these permutations are compatible. Thus bitonic sorting fits the new control scheme.

The number of permutations required by our simulation is the sum of the number of the interconnections considered above (i.e., $m(m+1)/2$ -1) and the number of the columns of the sorting network (i.e., $m(m+1)/2$). Hence, the overall number is $m(m+1)$ - 1 = m^2+m-1. This offers a factor of 3/2 speed up over Stone's bitonic sorting on the shuffle exchange network which requires $3/2m^2+3/2m$-1 permutations (or passes) [STO71].

8.4.4 Compatibility of "Frequently Used Permutations"

Lenfant [LEN78] introduced five families of permutations that have emerged from the literature on parallel algorithms. The first is the family of cyclic shifts ($\lambda_{j,l,2k}$: $x \rightarrow j^*x + l \bmod 2^{2k}$, for odd j). The second consists of cyclic shifts within segments ($\delta_{j,l}$: $(a_1,a_2) \rightarrow (a_1, \lambda_{1,l,2k-j}(a_2))$, where a_1 has j bits, and a_2 has $2k$-j bits). The third is the bit reversals within segments (ρ_j : $x_{2k-1}...x_jx_{j-1}...x_0 \rightarrow x_{2k-1}...x_jx_0x_1...x_{j-1}$). The fourth ($\beta_{j,l,m,s}$: $(a_4, a_3, a_2, a_1) \rightarrow (a_4, a_3, a_2, a_1) \oplus s$, where a_4, a_3, a_2 and a_1 have j bits, l bits, m bits, and $2k$-j-l-m bits, respectively, and s is a $2k$-bit number), and the fifth ($\gamma_{j,l,m,h}$: $(a_5, a_4, a_3, a_2, a_1) \rightarrow (a_5, a_2, a_3, a_4, a_1)$, where a_5, a_4, a_3, a_2 and a_1 have j bits, l bits, m bits, h bits, and $2k$-j-l-m-h bits, respectively) are variations on the shuffles and unshuffles and their purpose is to cover the permutations required by bitonic sorting. The extra permutation of the last two families are added for the "stability of the analysis". Thus, permutations required by bitonic sorting are the "essence" of the last two families.



Configuration of col 0 for Bitonic Sorting
Fig. 8.8

Each of the first three families will be shown compatible. The essence of the last two families was proved in the preceding subsection to consist of compatible permutations.

Cyclic shifts are realizable by $\Omega(2,2k)$ (see [LAW75]) which is covered by $\Omega(r=2^k,2)$. Therefore the first family is h-compatible where h is the identity permutation.

The cyclic shifts within segments can be similarly shown to be realizable by $\Omega(2,2k)$, following the same line of reasoning as in [LAW75]. The second family is then compatible too.

Bit reversals within segments can be easily shown to be pseudo bit translations. Since all pseudo bit translations are compatible, the family of bit reversals within segments must be compatible. This brings us to the end of the study of compatibility in this chapter. We will turn next to the use of bitonic sorting compatibility and the implementation of the new control scheme.

8.5 Conclusions

A new approach to controlling 3-stage Benes networks has been developed. It is more versatile than other previous approaches. Compatibility was characterized and a technique to show the compatibility of families of permutations (generated by classes of problems) was derived from the characterization theorem. Various families of permutations were shown compatible. In Particular, FFT and bitonic sorting were proven to produce compatible families, providing efficient communication for these two problems. There are other classes of problems that yield to the new scheme, too. For example, matrix computations (matrix multiplication, linear transformation, LU decomposition), semi-group computation(addition and multiplication of N numbers, maximum and minimum of N values), recurrence equations of order ≥ 1, etc. ... Actually, the communication patterns of these problems are binary trees; consequently, these problems yield families of permutations that are admissible by omega networks and therefore by the new scheme, for omega networks are a special case. It is of interest to inspect other classes of problems and see whether they produce compatible families of permutations. The unsolved case is how to proceed when the permutations cannot be functionally defined (as in FFT and Bitonic Sorting) but are irregular as in sparse linear systems.

It remains to say a few words about the implementation of the scheme. The implementation can be integrated into the instruction set of the system. For each known h-compatible family, the configuration h is stored in memory. A new instruction is added to the instruction set. This instruction takes an operand that points to the location of h. The execution of this instruction consists of setting column 0 to h. If the system has already an instruction LOAD-PERM that takes a permutation (or a pointer to it) as an operand and sets the network to it, then a new bit c can be added to this instruction. If c is set to 0, the same execution takes place; and if c is set to 1, then the configuration h is loaded into col 0 and the network is left to operate in the self-routed mode using r-ary control tags. This is one of the possible implementations. It shows the utility of the scheme when the problem can be represented with compatible permutations.

As sorting the destination tags brings the sources to their destinations, any sorting algorithm can be used to rout any permutation through $BC(r,2)$ in as many passes as is needed to sort the destination tags. Therefore, the bitonic sorting algorithm can be used to rout arbitrary permutations, yielding an $O(\log_2^2 N)$ control algorithm of $BC(r,2)$, which is as efficient as Stone's S-E network for sorting, while keeping the generality and versatility of $BC(r,2)$ for compatible families.

Chapter 9

Parallel Random Walk Computation and its Application to the Monte Carlo Solution of PDE's

In the preceding chapter, problems with "standard" communication patterns were studied and shown to fit the new control scheme of 3-stage Benes. In a way, the accommodation of these patterns was staraightforward . To illustrate the adaptation of the novel, hence "non-standard", parallel algorithm to the three-stage Benes network, this chapter introduces a new technique for the parallelization of random walks. The underlying assumption is an unlimited supply of processors and hence the applicability in this specific case is limited. However, the functional composition scheme employed would be useful in other parallel computation contexts also.

9.1 Introduction

The Monte Carlo Method has been studied and used to solve elliptic and parabolic partial differential equations [HAL70], [HAM64], [SAD74]. It holds several advantages over other methods, such as solving problems with irregular boundaries and/or discontinuities; giving solutions at single points independently from the solutions at other points; and allowing great parallelism .

The great amount of inherent parallelism is drawn from the fact that the solutions at different points are independent, paving the way to independent processes that can run in parallel. Moreover, the solution at each point consists of evaluating a "primary estimator" along a large number of random walks, then averaging these values. The random walks are independent, so here too the estimations along the random walks can be computed in parallel.

A much less obvious amount of parallelism can be introduced into the evaluation of the primary estimator along a random walk. It is less obvious because the random walk is constructed sequentially making the computation proportional to the length of the random walk.

In this chapter we parallelize the construction of random walks and along with it the evaluation of the primary estimator (this is called intra-walk parallelism), reducing the time of this part of the solution from O(n) to O(log$n$) where n is the length of the random walk.

In section 9.2, the Monte Carlo Method for PDE's is presented briefly, and all its possible areas of parallelism are pointed out. Section 9.3 introduces the intra-walk parallelism and presents the parallel algorithm for the random walk construction. In section 9.4 the complexity of the algorithm is analyzed. Section 9.5 proposes suitable architectures for random walk computation, particularly a 3-tree pipeline. Section 9.6 maps this pipeline onto 3-stage Benes and shows that mapping to fit the new control scheme of 3-stage Benes.

9.2 The Monte Carlo Method and its Inherent Parallelism

Let $AU_{xx} + 2BU_{xy} + CU_{yy} + DU_x + EU_y + F = 0$ (1)

be a PDE and D a region with boundary C. A, B, C, D, E and F are given functions of x, y and possibly the time variable t, and U is the function to be computed.

The Monte Carlo Method is used to solve the following two problems:

A. The elliptic PDE problem:

U, A, B, C, D, E and F are time-independent and $B^2 - AC < 0$ on D.

Solve equation (1) subject to the boundary condition:

$$U(x,y) = \phi(x,y) \quad \text{if } (x,y) \in C \quad (2)$$

B. The parabolic PDE problem:

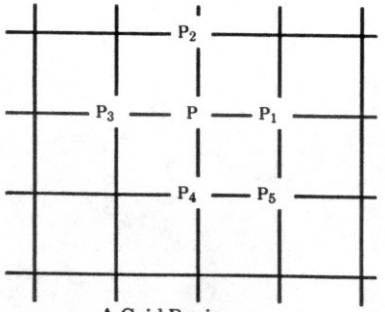
U, A, B, C, D, E and F are time-dependent and $B^2 - AC = 0$ on D.

Solve equation (1) subject to:

Boundary condition: $U(x,y,t) = \phi(x,y,t)$ if $(x,y) \in C$ (3)

Initial condition: $U(x,y,0) = g(x,y)$ if $(x,y) \in D$ (4)

The region D is divided into a regular grid of size h. Each point P of the grid (except the boundary points) has five neighbors P_1, P_2, P_3, P_4, P_5 as depicted in Fig. 9.1 We denote by d_i the direction along which we move from P to P_i where $i = 1, 2, 3, 4, 5$. A random number generator generates random directions (i.e., n, s, e, w, and sw). A random walk starting at P is constructed by moving away from P following directions generated by the random number generator until an absorbing point is hit. In the elliptic case, the absorbing points are the boundary points, while in the parabolic case, they are either boundary points or points reached at time point 0.



A Grid Region
Fig.9.1

Let $r(P) = 2(A - B + C) + h(E + D)$ where A, B, C, D, E and F are evaluated at (x,y), the coordinates of P.

Let W_i be a random walk starting at P and ending at a boundary point Q_i, and

$$Z(W_i) = \phi(Q_i) + h^2 \sum_{P_j \in W_i} \frac{F(P_j)}{r(P_j)} \qquad (5)$$

The Monte Carlo solution of the elliptic equation (1) at point P, subject to (2), consists of generating a number of random walks $W_1, W_2, \ldots, W_N$, all starting at P and ending at $Q_1, Q_2, \ldots, Q_N$, respectively. Then, $Z(W_i)$ is evaluated (Note that $Z(W_i)$ can be evaluated while generating W_i). Afterwards, $U(P)$ is approximated by

$$\theta = \frac{1}{N} \sum_{i=1}^{N} Z(W_i) \qquad (6)$$

.

$Z(W_i)$ is called the primary estimator of $U(P)$, and θ the secondary estimator. For the proof that this method yields a good approximation of U, see [CUR49], [SAD].

For the parabolic case, where U and the coefficients of (1) are time dependent, the time scale is discretized into equal units of length k (i.e., $t_n = nk$, $n \geq 0$), and $U(P)$, A, B, C, D, E, F and $r(P)$ at time t_n are denoted $U_n(P)$, $A_n, B_n, C_n, D_n, E_n, F_n$ and $r_n(P)$, respectively.

Random walks W's are constructed as before except that W is started at P at time $t_n = nk$, and at each step (following a new direction), the time is decreased one unit. W is finished if either a boundary point is reached or time runs out (after n steps), whichever comes first.

In this case,

$$Z(W_i) = V_s(Q_i) + h^2 \sum_{j=0}^{n-s} \frac{F_{n-j}(P_j)}{r_{n-j}(P_j)} \qquad (7)$$

$V_s(Q_i) = \phi_s(Q_i)$ if $Q_i \in C$, $s \geq 0$

$g(Q_i)$ if $s = 0$ (i.e., Q_i is reached at time 0, and may be a non-boundary point)

The Monte Carlo solution of the parabolic equation (1) at point P, at time t_n, subject to (3) and (4) consists of generating W_1, W_2, ..., W_N, evaluating the $Z(W_i)$'s and averaging them, as in the previous case.

Now it can be clearly seen that the random walks W_1, W_2, ..., W_N are independent, $Z(W_1)$, $Z(W_2)$, ..., $Z(W_N)$ can be computed independently (and thus in parallel). This inter-walk parallelism has been studied in [BHA77], [BHA79], [SAD74]. It is also clear to see that U can be computed at different points independently (and thus in parallel).

The third candidate for parallelism is the generation of a random walk W_i, and the computation of $Z(W_i)$ along with it. We call this part random walk computation.

We shall describe the sequential random walk computation next. The computation of $Z(W_i)$ in the elliptic case is carried out as follows:

```
x = 0;
temp-end = P;      {holds the temporary end-point of the walk}
is-boundary = false;

while(is-boundary = false) do
  begin
    is-boundary = check-boundary(temp-end);
    if (is-boundary = false)
      x = x + F(temp-end) / r(temp-end);
    d = RNG();      {a random direction}
```



```
    temp-end = new-node(temp-end,d);   { updates the temporary end-point
                                         using the previous end and the direction d}
  end
Z = φ(temp-end) + h*h*x;
```

Algorithm 9.1

The parabolic case is quite similar and will not be treated separately.

Some slight parallelism is obvious: the random number generator, the updating of end-point, and the checking of boundary-crossing are independent and can run in parallel. This parallelism along with the inter-walk parallelism has been studied in [SAD74]. Different implementation schemes on different machine architectures such as SIMD, MIMD, etc. ... have been studied in [SAD74], [BHA77], [BHA79] and [BHA81].

The bulk of the computation time remains in the sequential random walk generation (the **while**-loop runs as many times as the random walk length). At first glance, this sequentiality seems inherent. However, it can be parallelized, cutting down the random walk computation time from $O(n)$ to $O(\log n)$ where n is the random walk length. This is the subject of the next sections.

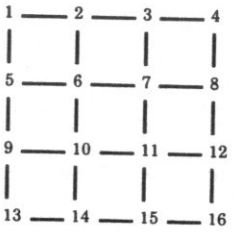
9.3 New Intra-Walk Parallelism

The first idea is to have a number of independent random number generators rather than only one. They can be thought of as a multiple random number generator that generates sequences of random numbers. Assume there are n RNG's and $n = 2^k$.

The problem can now be cast as follows: Given a grid, a point P of the grid, and a sequence of random directions $f_1, f_2, ..., f_n$ (taken from the five directions $n, s, e, w,$

sw), construct in parallel the random walk that starts at P and moves away following direction $f_1, f_2, ..., f_n$, consecutively.

Suppose that the grid points are numbered from 1 to S, row-wise. We can think of $f_1, f_2, ..., f_n$ as functions acting on integers. As an example, take the grid in Fig. 9.2.



A Labeled Grid
Fig. 9.2

$w(i) = i + 1; e(i) = i - 1; n(i) = i - 4; s(i) = i + 4; sw(i) = i + 5.$

Let $P_1 = f_1(P)$, $P_2 = f_2(P_1)$, ..., $P_n = f_n(P_{n-1})$. P, P_1, P_2, ..., P_n is the random walk sought. the problem then is to find $P_1, P_2, ..., P_n$ in parallel, and compute $Z(W)$ as the search for $P_1, P_2, ..., P_n$ takes place.

The algorithm consists of three phases. Phase I involves the following computation:

$$Step\ 1: f_1^{(1)} = f_2 f_1, f_2^{(1)} = f_4 f_3, ..., f_{n/2}^{(1)} = f_n f_{n-1}$$

$$Step\ 2: f_1^{(2)} = f_2^{(1)} f_1^{(1)}, f_2^{(2)} = f_4^{(1)} f_3^{(1)}, ..., f_{n/4}^{(2)} = f_{n/2}^{(1)} f_{n/2-1}^{(1)}$$

$$Step\ i: f_1^{(i)} = f_2^{(i-1)} f_1^{(i-1)}, f_2^{(i)} = f_4^{(i-1)} f_3^{(i-1)}, ..., f_{n/2^i}^{(i)} = f_{n/2^{i-1}}^{(i-1)} f_{n/2^{i-1}-1}^{(i-1)}$$

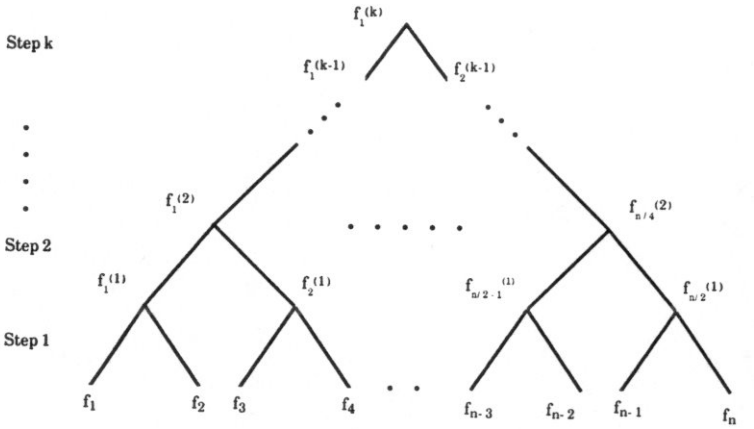
$$Step\ k: f_1^{(k)} = f_2^{(k-1)} f_1^{(k-1)}$$

This is depicted in Fig. 9.3, in a bottom-up fashion, assuming that $n = 2^k$ for simplicity.

Note that $f_1^{(k)} = f_n f_{n-1} \cdots f_2 f_1$, $f_1^{(k-1)} = f_{n/2} f_{n/2-1} \cdots f_1$, $f_2^{(k-1)} = f_n f_{n-1} \cdots f_{n/2+1}$, $f_1^{(i)} = f_{n/2^i} \cdots f_2 f_1$,

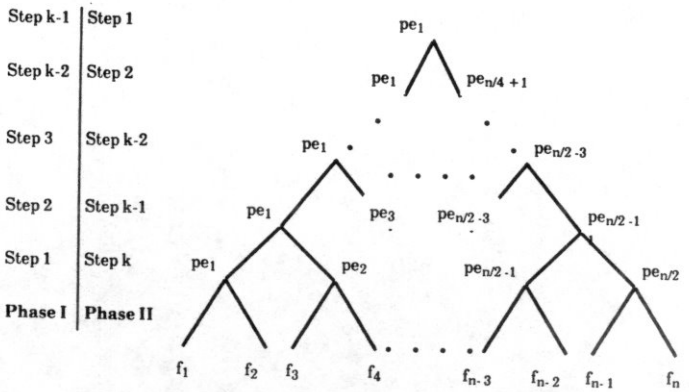
and in general $f_j^{(i)} = f_{2^i j-1} \cdots f_{2^i j-2^i+2} f_{2^i j-2^i+1}$, *is* a composition of $n/2^i$ functions.

Each step can be processed in parallel by a number of processing elements (pe's).



Phase I
Fig. 9.3

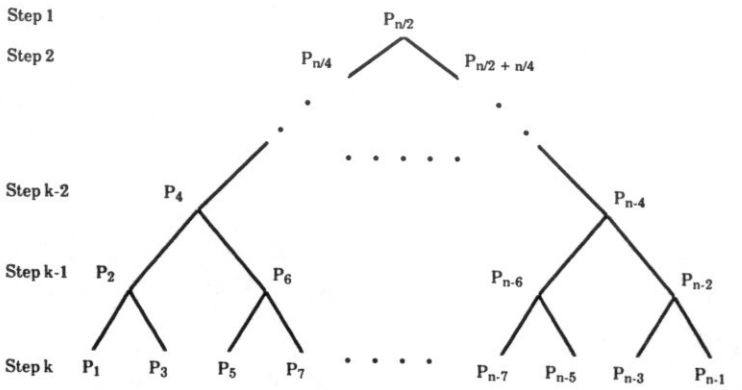
In phase II, $P_1, P_2, ..., P_{n-1}$ are found in a top-down fashion as depicted in Fig. 9.5. Here the indices of the points determined at the nodes coincide exactly with the numbers of the nodes of that tree if it is a binary search tree with keys 1, 2, ..., n-1. At the end, P_n is determined by pe_1. Note that the assignment of the pe's is the same as in Fig. 9.4. Along with the determination of $P_1, P_2, ..., P_n$ goes the boundary-crossing checking and the necessary measures to be taken in this case. This will be explained in more detail later.



Assignment of pe's for Phase I and II

Fig. 9.4

In phase III, the terms of the summation part of (5) (or (7)) that correspond to the points found and proved to be within boundary in phase II are summed in a bottom-up fashion, with the same assignment of pe's as in Fig. 9.4.



Phase II

Fig. 9.5

If the boundary is not reached yet, a new sequence of random directions is generated and the three phases are repeated with this new sequence and the last point reached (in the previous iteration) as input. This cycling continues until an absorbing point Q is reached. During the cycling, the partial sums computed in phase III are accumulated. In the final step, Z is computed by multiplying the accumulated sum by h^2 and then adding it to $\phi(Q)$.

Several questions arise at this point:

(i) Whether the composition of those functions can be carried out easily and independently from arguments. The answer is positive, as will be seen later, making phase I implementable.

(ii) How to compose those functions (in order to find the points in phase II).

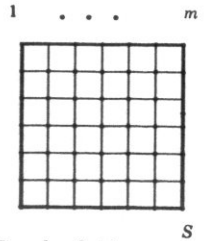
(iii) Whether the resulting functions reveal if there is any boundary crossing and where. The answer is negative.

(iv) What measures to take to detect boundary crossing (in order to make phase II implementable).

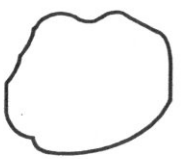
These questions are handled next.

If the region is a full grid as in Fig. 9.6, we number its nodes row-wise from 1 to S. However, if it is irregular as in Fig. 9.7, we make the assumption that the region can be gridded so that all the rows have the same number of nodes and so do all the columns (Fig. 9.8). Nodes are numbered row-wise in this case too. This numbering scheme is used to make the five direction-functions simple. Suppose the row size is m. Then, $w(i) = i + 1, e(i) = i - 1, n(i) = i - m, s(i) = i + m,$ *and* $sw(i) = i + m + 1$.

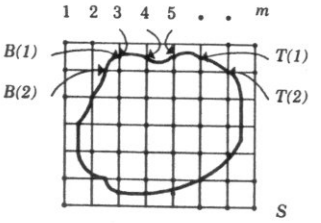
Note that these direction-functions n, s, e, w, sw are "translations" of the form t_a where $t_a(x) = x+a$. Since $t_a t_b = t_{a+b}$, each translation t_a can be identified by its parameter a (i.e., its computer representation is the number a), and the composition is done by adding the corresponding parameters without referring to the argument x. Therefore, the f_i's and their compositions are easily computable translations. This answers question (i). Hence, the functions of phase I are computed by mere additions of their representations, and are themselves translations, answering question (ii). It is important to note the distinction between *computation* of a function and



Regular Grid
Fig. 9.6



Irregular Region
Fig. 9.7



Irregular Gridded Region
Fig. 9.8

evaluation of a function at some integer; the first yields a function while the second yields an integer in this context.

As to question (iii), note that since $f_j^{(i)}$ is a translation (of parameter a_{ij}, say), a_{ij} does not provide any information as to whether the boundary is crossed (except in the case where $a_{ij} > S$).

Now we come to the last and most important question, the boundary-crossing detection. This will be done in phase II of the algorithm as follows: As the finding of the P_i's proceeds in the top-down fashion, each time a P_i is computed at some pe_j, P_i is checked if it is on boundary, off boundary or within boundary. If P_i is within boundary, the computation proceeds normally to find the other points; if it is off boundary, P_i and all the the P_s for $s>i$ are discarded from the walk (i.e., all the points computed (or that will be computed) at the right side of P_i in the tree in Fig. 9.5); if P_i is on boundary, all P_s for $s>i$ are discarded. Discarding those points can be carried out on an SIMD machine by having the pe that computes P_i report to the control unit that P_i is off boundary; then the control sends instructions to all the pe's

responsible for P_s, $s>i$, to discard those P_s's. In phase III, those discarded points do not contribute to the sum.

The implementation of the checking is discussed next. In case the region is a full grid as in Fig. 9.6, it is simple:

Let $I = P_i \bmod m$ (recall that P_i is an integer)
If $P_i > S$ or $P_i < 1$, it is off boundary
If $I = 0$ or 1, it is on boundary
Otherwise, it is within boundary

In case the region is irregular as in Fig.9.7, some data structure has to be kept in the memory of each processing element. After embedding the region in a grid as in Fig. 9.8, for each row i, we keep $B(i)$ and $T(i)$, beginning and ending nodes of row i on the region, respectively.

Let $I = \lceil P_i/m \rceil$ (the row number of P_i)
If $P_i > S$ or $P_i < 1$ or $P_i < B(I)$ or $P_i > T(I)$, it is off boundary
If $P_i = B(I)$ or $P_i = T(I)$, it is on boundary
If $B(I) < P_i < T(I)$, it is within boundary

The full algorithm is shown below.

procedure Random-Walk-Computation (P)
begin
x=0; {holds the accumulated sum}

$P_0 = P$; {P_0 holds the temporary end of the random walk which is initially the point P only}
Z = 0; {the primary estimator}
boundary = **false**; { a boolean variable to tell if the boundary has been crossed}
while(boundary = false) **do**
begin
for i=1 **to** n **step** 1 **in parallel do**
begin
f_i = RNG$_i$(); {parallel generation of n random directions. f_i = 1,-1,m,-m or $m+1$}
end
{phase I next}
for i = 1 **to** k **step** 1 **do** {i is the step number of Fig. 7.3}
for j = 1 **to** n/2^i **step** 1 **in parallel do**
$f_j^{(i)} = f_{2*j}^{(i-1)} + f_{2*j-1}^{(i-1)}$; { as seen earlier, function composition is equivalent to the addition of their representations }
{phase II next}
for i = k **to** 1 **step** = -1 **do** {i is the step number in Fig. 7.5}
begin
$P_n = P_0$; { this is for index handling only}
for j = 1 **to** n/2^i **step** 1 **in parallel do**
begin
index = (2*j-1)*2^{i-1}; {the index of the j-th point (from left) in the i-th step of Fig. 7.5}
father_index = (⌈(j-1)/2⌉ +2)*2^{i-1}; {the index of the

```
                                                    father of P_index }
            P_index = f_j^(i) + P_father-index ;    {f(P) = f + P if f is a
                                                    translation}
            boundary = check__boundary(P_index);
        end

    end
  if(boundary = false) then
    begin
      P_n = f_n + P_n-1;
      boundary = check__boundary(P_n);
      if(boundary = false) then   {updates the temporary end-point of
                                   the walk if the boundary is not yet
        P_0 = P_n;                 reached}

    end
{phase III next}
    for j = 1 to n step 1 in parallel do
      if (P_j is discarded) then
        x_j^(0) = 0;
      else  x_j^(0) = F(P_j)/r(P_j);
    for i = 1 to k step 1 do
      for j = 1 to n/2^i step 1 in parallel do
        x_j^(i) = x_2*j^(i-1) + x_2*j-1^(i-1) ;
{end of phase III}
    x = x + x_1^(k);        {update the accumulated sum}
end {of the while loop)
```



```
    Z = h*h*x + φ(P_0);     {final value of the random walk}
end
```

algorithm 9.2

The check__boundary procedure is presented next for the general (i.e., irregular) region :

```
procedure check__boundary(P_i)
begin
  I = ⌈P_i /m⌉ ;
  if(P_i > S or P_i < 1 or P_i < B(I) or P_i > T(I)) then
    begin
      discard all P_s for s ≥ i;    {done by the control unit}
      boundary = true;
    end
  if(P_i = B(I) or P_i = T(I)) then
    begin
      discard all P_s for s > i;    {done by the control unit}
      boundary = true;
      P_0 = P_i;      {update endpoint of the walk}
    end
  return(boundary);
end
```

Algorithm 9.2 is for elliptic equations. For parabolic equations, it needs only minor, straightforward modifications.

9.4 The Complexity of the Algorithm

This section discusses the computation time, the number of pe's needed for full parallelism, and the communication time.

Let n be the average random-walk length (not to be confused with the direction-function n), p the number of pe's used for the random walk computation, and q the number of random number generators.

The sequential time of Random-Walk-Computation is $O(n)$ because the while-loop of algorithm 9.1 loops n times. For the parallel time (of algorithm 9.2), suppose first that $p \geq n/2$. The expected number of times algorithm 9.2 will loop is $\lceil q/n \rceil$. Each step of phase I takes one time unit because it involves one addition (actually many additions all done in parallel because the available pe's are more than the addition operations). Thus, phase I takes $k = \log n$ time. Each of the remaining two phases have similar computational patterns (i.e., k steps, each executes in parallel and takes a constant time). Therefore, the three phases take $O(\log n)$ time, and the whole algorithm takes $O(\lceil q/n \rceil \log n)$ time ($= O(\log n)$ if $q = n$).

If $p \leq n/2$, each of the three phases takes $O(\lceil (n/2)/p \rceil \log n)$ time; consequently, the whole algorithm takes $O(\lceil q/n \rceil \lceil (n/2)/p \rceil \log n)$ time.

To complete the solution at one point, the $Z(W_i)$'s (N of them) have to be computed and averaged. Their computation can be done in parallel taking Np pe's and $O(\lceil q/n \rceil \lceil (n/2)/p \rceil \log n)$ time. Averaging them takes $\log N + 1$ time units on N pe's.

As a result, $U(P)$ takes $O(\log n) + \log N + 1$ time on $Nn/2$ pe's if Random-Walk-Computation is fully parallelized, and $O(n) + \log N + 1$ time if Random-Walk-Computation is run sequentially.

The above figures are for the computation time. For the communication time, Fig. 9.3, Fig. 9.4 and Fig. 9.5 show that the three phases have the same communication patterns: A binary tree pattern, bottom-up or top-down. the

communication implementation is architecture-dependent. The next section proposes suitable architectures on which the communication time is $O(\log n)$.

From the analysis above, it is concluded that if the grid is large and fine-grained, then there are very many grid points and n is expected to be very large, and parallel Random-Walk-Computation offers great speed-up (in the order of $n/\log n$). On the other hand, if n is relatively small, the speed-up is small and may not warrant the cost of extra processing elements.

It should be noted that if the number of available pe's is less than or equal the number of random walks, then running Random-Walk-Computation in parallel would increase the overall computation time of θ. *Consequently, Parallel* Random-Walk-Computation *should be used only when the number of pe's exceeds the number of the random walks.*

9.5 Architectures for Random Walk

In this section, three different architectures are proposed for Random-Walk-Computation. The first is an omega-connected machine, the second a tree machine, and the third is a pipeline of 3 trees.

The communication requirements can be cast in terms of permutations. The permutations required are those which carry out the data movement from one level to the next in the tree of Fig. 9.4. In the bottom-up phase, the partial permutation needed between step i and step $i+1$ can be easily shown to be $\alpha_i(x) = x\text{-}2^i$, for $i = 1, 2, \ldots, k\text{-}2$. Similarly, in the top-down phase, the partial permutation needed between step i and step $i+1$ is $\beta_i(x) = x + n/2^{i+1}$, for $i = 1, 2, \ldots, k\text{-}2$. These partial permutations are embedded in cyclic shift permutations defined next (see [LAW75]). A cyclic shift of parameter i is $\pi_i(x) = x + i \bmod n$, where i is an integer. α_i is clearly embedded in π_j where $j = n\text{-}2^i$. β_i is embedded in π_j where $j = n/2^{i+1}$.

The cyclic shifts are realizable by Ω [LAW75]. Therefore, the O(3log$n$) permutations required by parallel Random-Walk-Computation are realizable by Ω, and thus, an omega-connected machine is quite suitable for this problem.

A second architecture is a tree machine of $n/2$ leaves. It is a nearly ideal architecture for Random-Walk-Computation due to the tree-structure of the algorithm. The number of communication steps for each phase is obviously k-2. Thus, the communication time of Random-Walk-Computation on this architecture is O(3log$n$). For more detail on the communication requirements of tree-structured algorithms, see [ARD82].

As the algorithm needs more than one phase, one tree is easily pipelined. To have a pipelinable machine for a multi-phase tree-structured algorithm, an array of as many trees as the number of phases is a good choice. In the Random-Walk-Computation case, the pipeline has 3 trees, as elaborated next.

Each of the trees has $n/2$ leaves and is of k-1 height. The i-th node (from the left) of the j-th level of the first (resp., second) tree is linked to the i-th node of the j-th level of the second (resp., third) tree (Fig. 9.9). Phase I, II and III are run on the first, second and third tree, respectively. After each computation step (at level i, say), all the pe's at level i of the first tree send to the pe's at level i of the second tree the data needed for the second phase, then they send to the pe's of the next level of the same tree the data needed in the next step of the same phase. Similarly, after each computation step (at level i), all the pe's at level i of the second tree send to the pe's at level i of the third tree the data needed for the third phase, then they send to the pe's of the next level of the same tree the data needed in the next step of the same phase. Clearly, this emulates the three phases on one tree, and thus solves the problem correctly. As the data movement is unidirectional along each link (Fig. 9.9), the architecture can be used to pipeline successive random walk computations (not the

phases of the same walk), where the input comes through the leaves of the first tree and the output goes out from the root of the third tree.

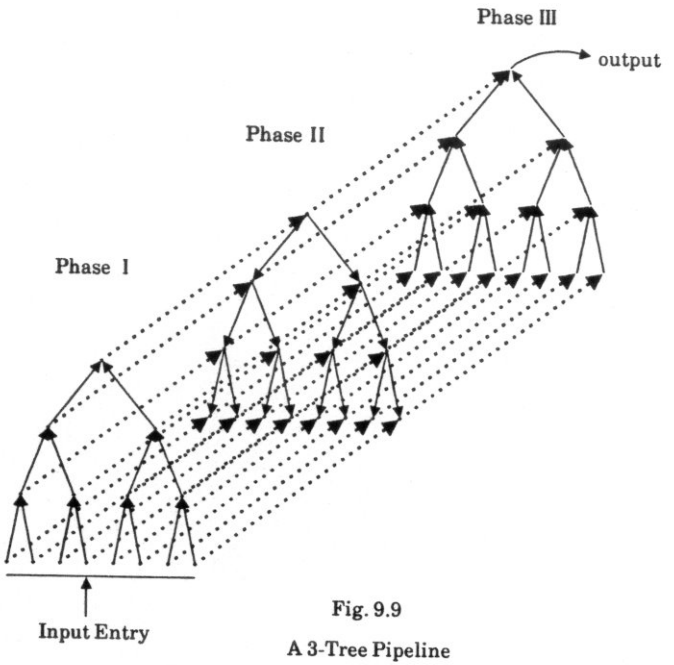


Fig. 9.9
A 3-Tree Pipeline

9.6 Mapping of the 3-Tree Pipeline onto 3-Stage Benes

This pipeline architecture can be mapped onto 3-stage Benes networks by breaking the communication pattern into permutations. This is done in such a way

that if pe i sends to pe j in the fixed interconnection network, one of the permutations must map i to j.

A lower bound on the number of permutations to simulate a fixed interconnection network is the maximum indegree and outdegree of the nodes of the network.

In the case of the 3-tree pipeline, this lower bound is three.

The main question is whether the pe's of the 3-tree pipeline can be labeled in such a way that it can be simulated by *three compatible* permutations.

The answer is positive and will be presented next.

Labeling each tree: Assume h is the height of the tree. The number of nodes is then 2^{h+1}-1. Label the root by 0...01 ($h+1$ bits). The nodes of the tree are labeled inductively as follows. The left child of pe r is labeled $2r$, and the right child $2r+1$. Note then that the nodes of level i (level 0 is the root) are labeled 2^i, 2^i+1, 2^i+2, ... , 2^{i+1}-1.

Simulation of a tree: If the tree is directed from leaves to root (fan-in), then the perfect unshuffle U and the permutation EU (where E is the exchange permutation) simulate the tree because U maps each left child to its father, and EU maps each right child to its father.

If the tree is directed from root to leaves (fan-out), the perfect shuffle S and the permutation SE simulate the tree because S maps each node to its left child and SE maps each node to its right child.

Therefore, a fan-in tree is simulated by U and EU, and a fan-out tree is simulated by S and SE.

Labeling the 3-tree pipeline: First, label the nodes of each tree as indicated above. Then add the two bits "00" to the left of the labels of the leftmost tree, the two bits "01" to the left of the labels of the middle tree, and the two bits "10" to the left of the labels of the rightmost tree. Assume the length of the labels is now $2k$.

Simulation of the pipeline: The leftmost and rightmost trees are fan-ins and hence each is simulated by U' and $U'E$, where U' is the sub-unshuffle that moves the rightmost bit to the third bit from the left (due to the addition of the two bits to the left of the labels). The middle tree is a fan-out and is then simulated by S' and $S'E$, where S' is the sub-shuffle that moves the third bit (from the left) to the rightmost position. Consequently, the three trees can be simulated at the same time by two permutations f_1 and f_2, where $f_1(x) = U'(x)$ or $S'(x)$ or $U'(x)$ depending on whether the two leftmost bits of x are 00, 01 or 10, respectively, and $f_2(x) = (x)EU'$ or $(x)S'E$ or $(x)EU'$ corresponding to the same three cases of the leftmost two bits of x.

The "horizontal" movement of data (i.e., from tree to tree) is simulated by the permutation f_3 where $f_3(00ab...c) = 01ab...c$ and $f_3(01ab...c) = 10ab...c$.

In summary, the pipeline is then simulated by 3 permutations f_1, f_2 and f_3.

Compatibility of f_1, f_2 and f_3: Let $\alpha_i(p,q)$ be the leftmost k bits of $f_i(pn+q)$ *where* $n = 2^k$. From the definition of f_3 above it follows that if $\alpha_3(p,q) = \alpha_3(p',q')$, then $p = p'$. Therefore, f_3 is t-realizable for any t (Theorem 8.1 and what follows).

As $f_1(x) = U'(x)$ or $S'(x)$ and as U' and S' each moves only one p-wing bit to the q-wing and one q-wing bit to the p-wing, f_1 moves then only one p-wing bit to the q-wing and one q-wing bit to the p-wing. It follows that f_1 is in L (Subsection 8.4.2).

As for f_2, note that $EU' = U'g_{\underline{b}}$, where $\underline{b} = 0010...0$ and $g_{\underline{b}}(x) = x \oplus \underline{b}$.

Thus, $f_2(x) = (x)U'g_{\underline{b}}$ or $(x)S'E$. It can now be easily shown that if $\alpha_2(p,q) = \alpha_2(p',q')$ and $p \neq p'$, then p and p' agree in all but one bit position, and q and q' agree in at least one bit position. Hence, f_2 is in L too.

Since the permutations of L are compatible and f_3 is t-realizable for any t, f_1, f_2 and f_3 are compatible. The compatibility function t is the one defined in Theorem 8.3.

This shows that the 3-tree pipeline can be simulated on 3-stage Benes by a minimum number of compatible permutations, making the parallel random walk computation fit the new control scheme of 3-stage Benes.

Chapter 10

Conclusions

10.1 Summary

Various aspects of a class of multistage interconnection networks, called MIN's, have been studied in this dissertation. They are complete, single path, rectangular, unidirectional, and composed of $r \times r$ permutation switches, The focus has been on the class as a whole rather than on specific networks. The interdependence of functionality and topology of MIN's, the functional comparison of networks with different switch sizes, the efficiency of control schemes, the structure of easily controlled networks, and the interconnection modularity were among the focal points of analysis.

It has been shown that two MIN's are functionally equivalent if and only if they are topologically equivalent. This establishes the equivalence between functionality and topology. Based on that, an optimal algorithm of complexity $O(N\log N)$ to decide if two networks realize the same permutations was developed. Here N is the number of terminals on each side of the networks.

The functional covering relation between networks of different switch sizes is a natural generalization of functional equivalence. It was shown that one network functionally covers another network if and only if the switch size of the first is a power of that of the second, and moreover, the first network unfolds to the second by replacing the switches of the first by networks whose building blocks are switches of the same size as that of the second network. An optimal algorithm of complexity $O(N\log_s N)$ was developed to decide functional covering, where s is the smaller switch size. Furthermore, existing networks, namely, Ω, Ω^{-1}, baseline, butterfly and generalized cube networks were generalized to accommodate various switch sizes,

and shown to be covered by networks of their own class but of switch size equal to a power of their own. In other terms, each of the above cited networks is guaranteed that its permutations would still be realizable if its switch size is raised to a power while keeping N constant.

Beyond the functional-topological relationship, there is the question of network control. MIN's are controllable via control tags due to the single path property. The speed of control depends on the efficiency of control tag computation. Storing the control tags is prohibitive for large systems. Networks with efficiently computable control tags are desirable. Two new control schemes, namely D-control and FD-control, were introduced. In D-control, the control tags are the destination addresses. Therefore, this scheme is direct and efficient. In FD-control, the control tags are a permutation of destination addresses. The structure of D-controllable and FD-controllable networks was studied and shown to be recursive, and optimal algorithms to decide D- &FD-controllability were devised.

The two forms of network equivalence and network control were combined to study some interesting aspects of subclasses of networks, namely, doubly controllable networks, digit-permutation networks and modular networks.

Doubly controllable networks, those that are FD-controllable from left to right and from right to left were studied for their theoretical interest. Most previously studied networks turned out to be doubly controllable. The study of this subclass showed that all doubly controllable networks are functionally equivalent; in particular, all the previously cited existing networks are functionally equivalent, and moreover, no new doubly controllable network can bring more functional power than the networks already studied.

Digit-permutation networks, where the interconnection between columns are digit manipulations were shown to be doubly FD-controllable and consequently

functionally equivalent to the baseline and omega. Efficient routing control was developed for these networks.

Modular networks, where the interconnections between stages are identical, were also a focus because of potential manufacturing efficiency and because one stage can simulate the whole network in a logarithmic number of passes. D-controllable modular networks were shown to be strongly equivalent to Ω, and FD-controllable modular networks weakly equivalent. Consequently, modularity brings no more functional power than Ω.

One major implication of the functional equivalence of digit-permutation networks and doubly FD-controllable networks is that the quest for such networks which can realize a given set of permutations reduces to finding an appropriate relabeling of the baseline or omega network.

As MIN's do not pass all permutations, there is a need for universal networks. Benes networks, and in particular the three-stage implementation, realize all permutations but are slow to control. A new approach to fast control of three-stage Benes networks was introduced and studied. It consists of setting the first column to some configuration, thus making the resulting network self-routed via control tags. The issue of interest was that for a given family of permutations, whether the first column can be set to a configuration so that the resulting network can realize the family without blocking. Such families are called compatible.

Compatibility was characterized and shown to reduce to a graph coloring problem. Several families of permutations were shown to be compatible, including those required by FFT and bitonic sorting. As a result an efficient solution of these problems using on 3-stage Benes networks is possible.

Finally, a parallel algorithm for random walk computation was developed. This illustrated some parallel techniques and an application of the controllable networks. The algorithm is tree-structured and employs a functional composition

technique which can be utilized to parallelize other problems. As the algorithm has three phases (fan-in, fan-out, and finally fan-in), a pipeline of three trees of processors is the ideal architecture for this problem. This pipeline is optimally mapped onto three-stage Benes by three compatible permutations, making random walk computation run efficiently on three-stage Benes.

10.2 Future Research

As in any research endeavor, there are a number of open questions and natural extensions for future work. The following is a list of some of these questions and extensions.

(1) Knowing the topology of a MIN, one can deduce its functionality. The converse is an open problem in the design of special-purpose systems. Given a family of permutations, is there a MIN that realizes this family? An easy-to-control MIN? A procedure to design such a MIN for an arbitrary family of permutations? And, in case a given network like omega can realize a given set of permutations if the terminals are relabeled appropriately, how can such a relabeling be found?

(2) An optimal strong equivalence algorithm was given in Chapter 3 based on the equivalence between topology and functionality. It is desirable to have efficient algorithms to decide weak equivalence.

(3) The structure of D-controllable and FD-controllable networks is now well understood. It is of interest to study the structure of G-controllable networks (i.e., where $CT(i,j) = f(i,j)$) for some particular control functions.

(4) Applicable characterization of permutations realizable by D-controllable networks, and of those realizable by FD-controllable networks would be valuable.

(5) The structure of D- & FD-controllable modular networks was shown to be equivalent to Ω. The extension to this problem is the structure of *any* modular network. Is it still equivalent to Ω?

(6) The general graph coloring problem is NP-complete. Deciding the compatibilty of m permutations $\phi_1, \phi_2, \ldots, \phi_m$ of $\{0, 1, \ldots, N\text{-}1\}$, where $N = r^2$, is a special graph r-coloring problem. The graph has r r-cliques labeled $0, 1, \ldots, r\text{-}1$, and the nodes within each clique are labeled $0, 1, \ldots, r\text{-}1$. There is an edge between node q of clique p and node q' of clique p' if $\lfloor \phi_i(pr + q)/r \rfloor = \lfloor \phi_i(p'r + q')/r \rfloor$ for some i.
Is this coloring problem still NP-complete? If not, find an efficient compatibility algorithm. Such an algorithm would be of great use in a compiler for a 3-stage-Benes system.

(7) A network is universal if it realizes every posiible permutation. WW'^{-1} is a universal network where W and W' are D-controllable networks. Is WW^{-1} a universal network for any MIN W? What class of networks W has the property that WW^{-1} is universal? What class of networks W has the property that WW is universal?

(8) In MIN's the stages are concatenated and hence used sequentially and possibly in a pipeline fashion. We propose that these stages be put in parallel between the input and output terminals. Although the degree of terminals is thus raised to log$N$, a number of advantages are gained. The system can use one stage, two stages, or any number up to log$N$ stages, either sequentially or in parallel. The communication delay can then be less than O(log$N$). Also, as the ends of each stage are now processors (or memories), it is possible to pipeline as well as to queue at the processors to resolve conflicts. Additional functional capabilities may be gained too

since the stages can now be used in any order. Standard fixed communication patterns may be simulated on fewer parallel stages than log$N$. Network partitioning may also prove to be more efficient on these parallel-stage networks. Switch fault detection should be easier, and switch fault tolerance is clearly higher.

The study and implementation of these features of parallel-stage networks have a practical value and may realize new potentials of multistage interconnection networks.

Appendix I

Proofs for propositions 3.1, 3.2 and 3.3

This appendix proves Propostions 3.1, 3.2, and 3.3 of Chapter 3. These proofs involve other lemmas as intermediary steps.

Lemma 3.1: If W and W' are two networks in IMIN($2,k,l$) and $W \equiv W'$, then there is a path between input terminal a and output terminal b of W (denoted $a \rightarrow b$) if and only if $a \rightarrow b$ is a path in W'.

Proof: Let $a \rightarrow b$ be a path in W. This path goes through a number of switches. Configure these switches to establish this path and complete the switch setting (arbitrarily) to a permutation π of W which maps a to b. Since $W \equiv W'$, π can pass through W' without conflict, mapping thus a to b. therefore $a \rightarrow b$ is in W.
If $a \rightarrow b$ is in W', the same proof applies to conclude that it is in W. □

Lemma 3.2: Let W and W' be two networks in IMIN($2,k,l$). $W \equiv W'$ if and only if any two paths that do not conflict in one network do not conflict in the other.

Proof: Follows the same line of reasoning as the proof of the previous lemma. □

Notation: If W is a network in IMIN($2,k,l$) and a an input terminal of W, denote by T_a the tree in W rooted at a (or more precisely, rooted at the switch linked to a) and by O_a the leaves of T_a (i.e., the output terminals reachable from a). If u is a switch of W, let T_u and O_u denote the same things, and let $O_u^c = O_a - O_u$.

Lemma 3.3: Let W be a network in IMIN($2,k,l$), a and b two input terminals of W, u a closest switch reachable from a and b (the distance is the number of columns

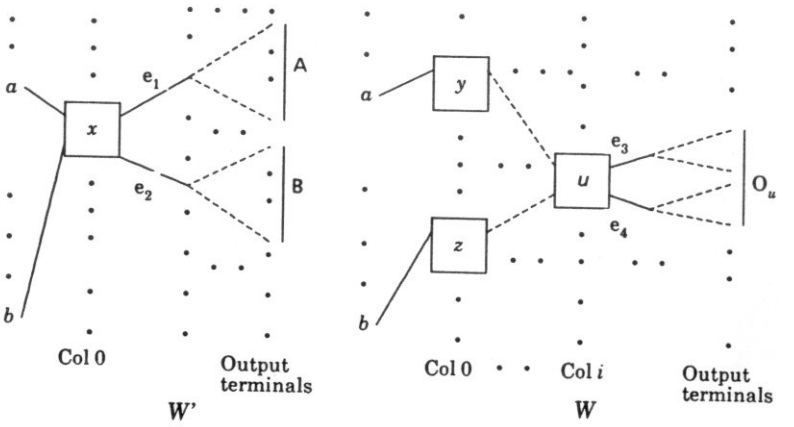


Fig. 3.2

between a (or b)and u), and $O_u^c = O_a - O_u$. If q is an output terminal in O_u^c, then the path $a \rightarrow q$ does not use any switch of either T_u or the path between b and u (Fig. 3.2-b).

Proof: The claim that $a \rightarrow q$ does not use any switch of T_u should be obvious because T_a is a binary tree, T_u is a subtree of T_a and q is not a leaf of T_u. If on the other hand $a \rightarrow q$ used any switch v of the path between b and u (excluding u because it is in T_u), then v would be reachable from a and b and closer to them than u, violating the definition of u. □

Lemma 3.4: Let W, a, b, u, and O_u^c be as in *lemma* 3.3 and let p be in O_u and q in O_u^c. Then $a \rightarrow q$ and $b \rightarrow p$ do not conflict in W.

Proof: It should follow immediately from *lemma* 3.3 since $b \rightarrow p$ uses the path between b and u and then switches in T_u. □

Lemma 3.5: Let W' be a network in IMIN(2,k,l), a and b two input terminals of W' linked to one switch x (fig. 3.2-a), A (resp. B) the set of output terminals reachable from x through link e_1 (resp. e_2) that leaves x. If p and q are two output terminals in A (or in B), then $a \rightarrow q$ and $b \rightarrow p$ conflict in W'.

Proof: They conflict over link e_1 if p and q are in A, and over e_2 if they are in B. □

Proposition 3.1: Let W and W' be two one-column networks in IMIN(2,k,l) such that $W \equiv W'$. Then W' can be transformed to W by a sequence of (sws) and (pwc) operations.

Proof: For each switch x of the column of W, let a and b (resp. c and d) be the two inputs (resp. output) linked to x, denoting the whole by (a,b,x,c,d). It is straightforward to show that there exists a switch y such that (a,b,y,c,d) is in W'.
Four or fewer (sws) operation can be applied on (a,b,y,c,d) to make its connectivity identical to that of (a,b,x,c,d). Then, y is relabeled by x. After processing all the switches in this manner (i.e., applying (sws) and (pwc) operation on W'), it is obvious to see that W' transforms into W. □

Proposition 3.2: Let W and W' be two networks in IMIN(2,k,l), where $l \leq k$, such that $W \equiv W'$ and the first stage (- , col 0) of W is identical to that of W'. We can apply a sequence of (sws)'s on the right side of the switches of the leftmost column of W' so that if the first stage (i.e., the leftmost interconnection and column of switches) is deleted from each network, the resulting networks are strongly equivalent.

Proof: If input terminals a and b are both linked to a switch x in W (and thus in W'), denote this by (a,b,x). From x stem two subtrees whose leaves form two sets A

and B in W and C and D in W'. It can be easily shown that A = C and B = D (or A = D and B = C). Assume the first is true and that A (resp. B) stems from the upper (resp. lower) output-port of x in W. If C stems from the lower output-port of x in W', apply (sws) on the right side of x in W', otherwise do nothing.
If the leftmost stage is deleted from both W and W', it is straightforward to show that the resulting networks are strongly equivalent. □

Proposition 3.3: Let W and W' be two networks in IMIN(2,k,l) such that $W \equiv W'$. If a and b are two input terminals that are linked to one switch x in W' (fig. 3.2-a), then a and b must both be linked to one switch in W too.

Proof: Assume the opposite is true (fig.3.2-b), that is, a is linked to some switch y, and b to a different switch z. Let A (resp. B) be the set of output terminals of W' reachable from x through link e_1 (resp. e_2) that leaves x.
Let c be in A. $a \rightarrow c$ and $b \rightarrow c$ conflict in W'. Therefore, they conflict in W. Hence, there is at least one switch that is reachable from a and b in W. Let u be a closest such switch (in the sense defined in Lemma 3.3). The notation O_a, O_u^c, O_u, T_a and T_u is kept for network W.
It will be proved that $O_u \cap A \neq \emptyset$, $O_u \cap B \neq \emptyset$.
Let m (resp. n) be in O_u(i.e., an output terminal in W) and reachable from u through link e_3 (resp. link e_4), as in fig. 6-b. Clearly, $a \rightarrow m$ and $b \rightarrow n$ do not conflict in W. Thus, they do not conflict in W' either. By Lemma 3.5, m must be in A and n in B, or vice versa. Consequently, $O_u \cap A \neq \emptyset$, $O_u \cap B \neq \emptyset$.
Note that $O_a = A \cup B$ because the outputs reachable from a are the same in both networks. Note also that $O_a = O_u \cup O_u^c$. Hence, $O_u^c \subseteq A \cup B$ and thus $O_u^c \cap A \neq \emptyset$ or $O_u^c \cap B \neq \emptyset$. Assume without loss of generality that $O_u^c \cap A \neq \emptyset$.
Let p be in $O_u \cap A$ and q in $O_u^c \cap A$.
By Lemma 3.4, $a \rightarrow q$ and $b \rightarrow p$ do not conflict in W because p is in O_u and q in O_u^c.

By Lemma 3.5, $a \rightarrow q$ and $b \rightarrow p$ conflict in N' because p and q are both in A.

This *contradicts* the fact that $W \equiv W'$. Therefore, a and b must both be linked to one switch in W. □

Appendix II

This appendix proves Propositions 4.1, 4.2 and 4.3 of Chapter 4. The other lemmas are intermediary steps.

Lemma 4.1: If W is in IMIN(r,k,l), E a subset of switches of some column i of W, and F the set of switches in col $i+1$ that are linked to switches in E, then $|E| \leq |F|$. Moreover, if E' is the set of switches in col i that are linked to switches in F, then E = E' iff $|F| = |E|$.

Proof: The number L of links going out of the switches of E is $r|E|$. All these links come into F. Let t_j be the number of links coming from E to the j-th switch of F.

$L = \Sigma t_j$ for $j = 1, \ldots, |F|$. Thus, $L \leq r|F|$ because $t_j \leq r$. Therefore, $r|E| \leq r|F|$ and $|E| \leq |F|$.

Similarly, $|F| \leq |E'|$. So we have $|E| \leq |F| \leq |E'|$. If E = E', then $|E| = |E'|$ and hence $|E| = |F|$. If $|E| = |F|$, then $r|E| = r|F|$. Thus, the number of links leaving E (i.e., $r|E|$) is the same number of links coming to F (i.e., $r|F|$). Therefore, every link coming to F is from E, and hence E = E'. □

Lemma 4.2: Let W be in IMIN(r,k,l), E_i a subset of switches of some column i of W, and E_{i+1}, ... E_j defined inductively as follows. E_t is the set switches in col t that are reachable from switches in E_{t-1}, for $t = i+1, i+2, \ldots, j$. Then $|E_i| \leq |E_{i+1}| \leq \ldots \leq |E_j|$. Moreover, if $|E_i| = |E_j|$, then $|E_i| = |E_{i+1}| = \ldots = |E_j|$ and the switches of E_i, E_{i+1}, ... E_j and their interconnections form an nxn subnetwork of W, where $n = r|E_i|$.

Proof: By applying the previous lemma repeatedly, we get $|E_i| \leq |E_{i+1}| \leq \ldots \leq |E_j|$. Therefore, if $|E_i| = |E_j|$, then $|E_i| = |E_{i+1}| = \ldots = |E_j|$. This shows that all the links coming to E_{i+1} are from the switches of E_i, those coming to E_{i+2} are from E_{i+1} and so

on. As a result, the switches of E_i, E_{i+1}, ... E_j and their interconnections form an nxn subnetwork of W, where $n = r|E_i|$. □

Proposition 4.1: If W is in IMIN(r,k,1) and W' in IMIN(s,k',t) and W functionally covers W', then r is a power of s and W topologically covers W'.

Proof: The number of outputs reachable from an input in W is $r^l = r$, and in W' it is s^t. As W functionally covers W', this two numbers must be equal due to the second condition of covering as indicated in Definition 2.12. Hence, $r = s^t$.

Let $a_1, a_2, \ldots, a_r$ (resp., $b_1, b_2, \ldots, b_r$) be the input (resp., output) terminals linked to a switch x of the one-column network W. Let E_0 be the switches of col 0 of W' to which $a_1, a_2, \ldots, a_r$ are linked. Define $E_1, E_2, \ldots, E_{t-1}$ as in Lemma 4.2. Every a_i can reach all $b_1, b_2, \ldots, b_r$ in W'.

If c is an input of W' such that c is not equal to any of $a_1, a_2, \ldots, a_r$, and c is linked to a switch y in E_0, then c can reach the outputs $b_1, b_2, \ldots, b_r$ in W' but not in W, contradicting the fact that W covers W'. Hence, the only inputs linked to E_0 are $a_1, a_2, \ldots, a_r$ and therefore, $|E_0| = r/s$. It can be shown that $|E_{t-1}| = r/s$ in a similar fashion.

By Lemma 2, the inputs $a_1, a_2, \ldots, a_r$, the outputs $b_1, b_2, \ldots, b_r$, the switches of $E_1, E_2, \ldots, E_{t-1}$ and their interconnections form an rxr subnetwork of W'. Clearly, this subnetwork is an unfolding of switch x of W.

As switch x is arbitrary, we conclude that every switch of W unfolds to a subnetwork of W', and on the whole, W unfolds to W'. □

Note that if W is in IMIN(r,k,t) and W' in IMIN(s,k',t') such that W functionally covers W', then $r^t = s^{t'}$ because r^t (resp., $s^{t'}$) is the number of output terminals reachable from an input terminal of W (resp., W'). These two numbers of terminals must be equal because W functionally covers W'. If $N = r^k = s^{k'}$, then $r = s^{k'/k}$ and $r = s^{t'/t}$ and thus $k'/k = t'/t$.

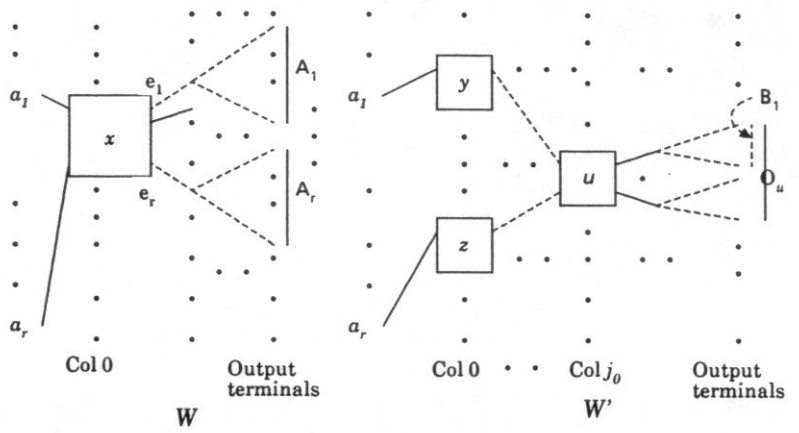


Fig. 4.1

Lemma 4.3: Let W be in IMIN(r,k,t), W' in IMIN(s,k',t') functionally covered by W, a_1, a_2, ... , a_r the input terminals to a switch x in W, u a leftmost switch that is reachable from all $a_1, a_2, \ldots, a_r$ in W', j_0 the column of u, A_i the set of output terminals of W that are reachable from output port i of switch x ($i = 1, 2, \ldots, r$), O_u the output terminals of W' reachable from u, and B_i the output terminals of W' reachable from output port i of switch u ($i = 1, 2, \ldots, s$), see Fig. 4.1. Let also $I_j = \{i \mid A_i \cap B_j = \varnothing\}$ and $I = \cup I_j$ for $j = 1, 2, \ldots, s$. Then $I_j \neq \varnothing$ for all j, $I_j \cap I_{j'} = \varnothing$ for every $j \neq j'$, and $|I| \geq s$.

Proof: Let $B = (\cup B_j)_{j=1,\ldots,s}$ and $A = (\cup A_j)_{i=1,\ldots,r}$. $B \subseteq A$ because each element d of B is an output terminal reachable from a_1 in W' and hence in W as W covers W',

implying that d is in A too. Therefore, for every j, $B_j \cap A = B_j \neq \varnothing$ and thus there is an i such that $B_j \cap A_i \neq \varnothing$. i is then in I_j and consequently $I_j \neq \varnothing$.

if $I_j \cap I_{j'} \neq \varnothing$ for some $j \neq j'$, let i be in $I_j \cap I_{j'}$. Then $A_i \cap B_j \neq \varnothing$ and $A_i \cap B_{j'} \neq \varnothing$.

There obviously exists some a_l such that a_1 and a_l reach switch u through two non-conflicting paths in W'. Let h be in $A_i \cap B_j$ and h' be in $A_i \cap B_{j'}$. Clearly the paths $a_1 \rightarrow h$ and $a_l \rightarrow h'$ do not conflict in W' because $j \neq j'$, but conflict in W because h and h' are both in A_i, contradicting the hypothesis that W covers W'.

$|I| \geq s$ follows immediately □

Lemma 4.4: Let W, W and j_0 be as in Lemma 4.3. Then $j_0 < k'/k$ where $N = r^k = s^{k'}$.

Proof: Assume that $j_0 \geq k'/k$. Let $E = (\cup A_j)_{i \in I}$ and $O_u^c = E - O_u$. Clearly $O_u = (\cup B_j)_{j=1,\ldots,s}$, and thus $|O_u| = \Sigma|B_j| = s|B_j|$. Note that $|B_j| = s^{t'-j_0-1}$.

It can be easily shown that $s^{t'-j_0-1} \leq r^{t-1-t/t'} < r^{t-1}$ using the fact that $r^t = s^{t'}$ and $j_0 \geq k'/k = t'/t$. As $|A_i| = r^{t-1}$, it follows that $|B_j| < |A_i|$ and therefore $|O_u| < s|A_i|$.

$|E| = \Sigma|A_i| = |I||A_i| \geq s|A_i|$ because $|I| \geq s$ after the previous lemma.

Thus, $|O_u| < |E|$ and therefore $E \not\subseteq O_u$ implying that $O_u^c \neq \varnothing$.

As $O_u^c \subseteq E$, there exists i in I such that $O_u^c \cap A_i \neq \varnothing$.

$O_u \cap A_i \neq \varnothing$ because i is in I and $O_u = (\cup B_j)_{j=1,\ldots,s}$.

Let p be in $O_u^c \cap A_i$ and q in $O_u \cap A_i$.

Since u is a leftmost switch reachable from $a_1, \ldots, a_r$ in W', there must exist a and b in $\{a_1, \ldots, a_r\}$ a and b reach switch u through two non-conflicting paths in W'.

It can be shown that the paths $a \rightarrow p$ and $b \rightarrow q$ do not conflict in W' because p is in O_u^c and q in O_u.

However, $a \rightarrow p$ and $b \rightarrow q$ conflict in W because both p and q are in A_i, contradicting the assumption that W covers W'. Therefore, $j_0 < k'/k$. □

Lemma 4.5: $j_0 = \lceil k'/k - 1 \rceil$.

Proof: The number of input terminals reachable from u in W' is s^{j0+1} on the one hand and $\geq |\{a_1, \ldots, a_r\}| = r$ on the other hand. Thus, $s^{j0+1} \geq r = s^{k'/k}$ and therefore, $j_0 \geq k'/k - 1$. We then have $k'/k - 1 \leq j_0 < k'/k$ and consequently, $j_0 = \lceil k'/k - 1 \rceil$. □

It will be shown next that the switches of col 0, ... , col j_0 of W' that are reachable from any of the inputs $a_1, \ldots, a_r$ form an independent subnetwork of W'.

Let F be the set of switches in col j_0 of W' that are reachable from *every* input of the set $\{a_1, \ldots, a_r\}$. Clearly, u is in F.

For every $i = 1, 2, \ldots, r$, let F_i be the set of switches of col j_0 that are reachable from a_i in W'.

Lemma 4.6: If W functionally covers W', then $F_i = F$ for every $i = 1, 2, \ldots, r$.

Proof: Fix an i. Let h be a switch in F_i. h is reachable from a_i. Let d be an output terminal reachable from a_i through h in W'.
Since W functionally covers W' and a_i can reach d in W', a_i can reach d in W too. As $a_1, \ldots, a_r$ are all linked to the same switch x in W, d is then reachable from all $a_1, \ldots, a_r$ in W. Hence, d is reachable from all $a_1, \ldots, a_r$ in W' too because W functionally covers W'. There must then exist a switch that is reachable from $a_1, \ldots, a_r$ in W' on their way to d. Let v be a leftmost such switch. Let j_1 be the label of the column of v in W'. Lemmas 4.3, 4.4 and 4.5 apply to j_1 and v as they apply to j_0 and u. Thus, $j_1 = \lceil k'/k - 1 \rceil$. Hence, $j_1 = j_0$ and v is in col j_0. As the path $a_i \rightarrow d$ goes through switch h and switch v in W', and as h and v are in col j_0, $h = v$. Consequently, h is reachable from $a_1, \ldots, a_r$ in W' implying that h is in F. Thus, $F_i \subseteq F$. The inclusion in the other direction is trivial. □

The following proposition is now obvious.

Proposition 4.2: Let W be in IMIN(r,k,t), W' in IMIN(s,k',t') functionally covered by W, $a_1, a_2, \ldots, a_r$ the input terminals linked to a switch x in W, j_0 the leftmost column in which there is a switch that is reachable from all $a_1, a_2, \ldots, a_r$ in W', see Fig. 4.1. The switches that can be traced from $a_1, \ldots, a_r$ rightward to col j_0 in W' form an independent subnetwork S in IMIN(s,j_0+1).

Proof: Follows immediately from Lemmas 4.6 and 4.2. □

Let $m = s^{j0+1}$, $c_1, c_2, \ldots, c_m$ be the output terminals of S (as defined in Proposition 4.2), and $C_1, C_2, \ldots, C_m$ the output terminals of W' that are reachable from $c_1, c_2, \ldots, c_m$, respectively. Let $A_1, A_2, \ldots, A_r$ be the sets of output terminals of W that are reachable from output ports 1, 2, ... , r of switch x (of Proposition 4.2), respectively, $A = (\cup A_j)_{j = 1, \ldots, r}$, and $C = (\cup C_j)_{j = 1, \ldots, m}$. Note that if p is in C_i and q is in C_j for $i \neq j$, then the paths $c_i \rightarrow p$ and $c_j \rightarrow q$ do not conflict in W'.
A is the set of terminals reachable from a_1 in W, and C is the set of terminals reachable from a_1 in W' because $F = F_1$. As W functionally covers W', $A = C$.

Proposition 4.3: For every $j = 1, 2, \ldots, m$, there is an i in $\{1,2, \ldots, r\}$ such that $C_j = A_i$. Moreover, $r = m = s^{j0+1}$.

Proof: Fix j. Since $C_j \subseteq C = A$, there is an i in $\{1,2, \ldots, r\}$ such that $C_j \cap A_i \neq \emptyset$.
Let p be in $C_j \cap A_i$. We claim that $A_i \subseteq C_j$. If not, let q be in A_i but not in C_j. As q is in $A = C$, there is an l in $\{1, 2, \ldots, m\}$ such that q is in $C_l \neq C_j$.
The path $a_1 \rightarrow p$ in W' goes through output terminal c_j of S because p is in C_j.
As $c_j \neq c_l$, there must exist an input a_t such that the paths $a_1 \rightarrow c_j$ and $a_t \rightarrow c_l$ do not conflict in S. Therefore, the paths $a_1 \rightarrow p = a_1 \rightarrow c_j \rightarrow p$ and $a_t \rightarrow q = a_t \rightarrow c_l \rightarrow q$ do not conflict in W'. However, both p and q are in A_i, making paths $a_1 \rightarrow p$ and $a_t \rightarrow q$ conflict in W, and contradicting the assumption that W functionally covers W'.
Hence, $A_i \subseteq C_j$.

It is easy to show that $s^{j_0+1} \geq r$ because $j_0 + 1 \geq k'/k$, and $|C_j| = s^{t'-j_0-1} = s^{t'}/s^{j_0+1}$ $= r^t/s^{j_0+1} \leq r^t/r = r^{t-1}$. Moreover, $|A_i| = r^{t-1}$. Thus, $|C_j| \leq |A_i|$ and $A_i \subseteq C_j$, resulting in $A_i = C_j$. In particular, $|A_i| = |C_j|$ and $r = s^{j_0+1}$. □

Appendix III

Proofs of Theorems 8.1 and 8.3

Lemma 9.1: If ϕ is a permutation, there exist h_0, h_1, h_2 in H such that $\phi = h_0 f h_1 f h_2$.

Proof: B(r,2) passes all permutations in S_N [3], and any configuration of any of the three columns of B(r,2) is in H. Therefore, $S_N = HfHfH$. □

Lemma 9.2: If ϕ is in S_N , there exist two mappings α and β from $R_r \times R_r$ to R_r such that $\phi(pr+q) = \alpha(p,q)r + \beta(p,q)$.

Proof : Define $\alpha(p,q) = \lfloor \phi(pr+q)/r \rfloor$ and $\beta(p,q) = \phi(pr+q) - \alpha(p,q)r$. It is easy to see that α and β are two mappings from $R_r x R_r$ to R_r. □

Lemma 9.3: h is in H iff there exists $t : R_r \times R_r \rightarrow R_r$ such that $h(pr+q) = pr + t(p,q)$ and $t(p,.)$ is a permutation of R_r for every p,q in R_r.

Proof: Only if:

Define $t(p,q) = h(pr+q) - pr$. It is obvious to see that t is a mapping from $R_r x R_r$ to R_r, and $t(p,.)$ is a permutation of R_r for every p.

If:

It is enough to show that h is one-to-one.

If $h(pr+q) = h(p'r+q')$, then $pr + t(p,q) = p'r + t(p',q')$, which implies that $p = p'$ and $t(p,q) = t(p',q')$. Thus, $t(p,.)(q) = t(p,.)(q')$ and therefore $q = q'$ because $t(p,.)$ is a permutation. Hence $pr+q = p'r+q'$. □

Note that if a column is set to h, then its p-th switch is set to $t(p,.)$.

Lemma 9.4: If ϕ is in S_N and $\phi = h_0 f h_1 f h_2$ for some h_0, h_1, h_2 in H, and if t_0 is such that $h_0(pr+q) = pr + t_0(p,q)$ and $\phi(pr+q) = \alpha(p,q)r + \beta(p,q)$, then $h_1(t_0(p,q)r + p) = t_0(p,q)r + \alpha(p,q)$ and $h_2(\alpha(p,q)r + t_0(p,q)) = \alpha(p,q)r + \beta(p,q)$.

Proof: $(pr+q)h_0 f h_1 f h_2 = (pr + t_0(p,q))f h_1 f h_2 = (t_0(p,q)r + p)h_1 f h_2 = t_0(p,q) + p')f h_2 = (p'r + t_0p,q))h_2 = p'r + q'$ for some p' and q' in R_r.

Since $\phi = h_0 f h_1 f h_2$, $p'r + q' = \alpha(p,q)r + \beta(p,q)$ and therefore $\alpha(p,q) = p'$ and $\beta(p,q) = q'$. Hence: $h_1(t_0(p,q)r + p) = t_0(p,q)r + p' = t_0(p,q)r + \alpha(p,q)$ and $h_2(\alpha(p,q)r + t_0(p,q)) = h_2(p'r + t_0(p,q)) = p'n + q' = \alpha(p,q)r + \beta(p,q)$.

Theorem 9.1: Let $\phi_1, \phi_2, ..., \phi_m$ be m permutations in S_N. $\phi_1, \phi_2, ..., \phi_m$ are compatible iff there is $t : R_r \times R_r \rightarrow R_r$ such that

(i) $t(p,.)$ is a permutation of R_r for every p in R_r.

(ii) If for some i, $\alpha_i(p,q) = \alpha_i(p',q')$ and $p \neq p'$, then $t(p,q) \neq t(p',q')$, where $\alpha_i(p,q) = \lfloor \phi_i(pr+q)/r \rfloor$.

Proof: <u>The if part.</u> We have $t : R_r \times R_r \rightarrow R_r$ satisfying (i) and (ii).

Let h_0 be the following permutation of H: $h_0(pr+q) = pr + t(p,q)$.

After Lemma 9.3, $h_0 \in H$. Define $h_1^{(i)}$ and $h_2^{(i)}$ for $i = 1, 2, ..., m$ as follows: $h_1^{(i)}(sr+p) = sr + \alpha_i(p,q)$ where $q = (t(p,.))^{-1}(s)$ and $h_2^{(i)}(\alpha_i(p,q)r + t(p,q)) = \alpha_i(p,q)r + \beta_i(p,q) = \phi_i(pn+q)$. It will be shown that $h_1^{(i)}$ and $h_2^{(i)}$ are well-defined permutations and $\phi_i = h_0 f h_1^{(i)} f h_2^{(i)}$ for $i = 1, 2, ..., m$.

Since $t(p,.)$ is a permutation of R_r, then $(t(p,.))^{-1}$, the inverse mapping, exists and is a permutation too. Thus, $h_1^{(i)}$ is well defined as a mapping from R_r to R_r.

$h_1^{(i)}$ is easily shown to be one-to-one and consequently it is a permutation of R_N because R_N is finite.

$h_2^{(i)}$ is a permutation:

We have to prove that $h_2^{(i)}$ is well-defined and bijective. By well-defined we mean that if $\alpha_i(p,q)r+t(p,q) = \alpha_i(p',q')r+t(p',q')$ then $h_2^{(i)}(\alpha_i(p,q)r+t(p,q)) = h_2^{(i)}(\alpha_i(p',q')r+t(p',q'))$.

$\alpha_i(p,q)r+t(p,q) = \alpha_i(p',q')r+t(p',q') \Rightarrow \alpha_i(p,q) = \alpha_i(p',q')$ and $t(p,q) = t(p',q') \Rightarrow p = p'$ by hypothesis (ii).

$p = p'$ and $t(p,q) = t(p',q') \Rightarrow t(p,.)(q) = t(p,.)(q') \Rightarrow q = q'$ because $t(p,.)$ is one-to-one.

Thus, we have $p = p'$ and $q = q'$ and therefore $h_2^{(i)}(\alpha_i(p,q)r+t(p,q)) = h_2^{(i)}(\alpha_i(p',q')r+t(p',q'))$

Hence, $h_2^{(i)}$ is well-defined, and the same argument proves that $h_2^{(i)}$ is one-to-one where it is defined. So if we prove that $h_2^{(i)}(R_N) = R_N$, then $h_2^{(i)}$ should be totally defined because $|h_2^{(i)}(R_N)| = |\{x \in R_N | h_2^{(i)}(x) \text{ is defined}\}|$.

For every x in R_N, there exists $pn+q$ in R_N such that $\phi_i(pr+q) = x$ because ϕ_i is a permutation.

$\phi_i(pr+q) = \alpha_i(p,q)r+\beta_i(p,q) \Rightarrow x = \alpha_i(p,q)r+\beta_i(p,q)$. But $h_2^{(i)}((\alpha_i(p,q)r+t(p,q)) = \alpha_i(p,q)r+\beta_i(p,q) = x$. Therefore, $h_2^{(i)}(R_N) = R_N$, and $h_2^{(i)}$ is well-defined, totally defined, and one-to-one $\Rightarrow h_2^{(i)} \in H$.

Now, it is a simple matter to prove that $\phi_i = h_0 f h_1^{(i)} f h_2^{(i)}$ for $i = 1, 2, \ldots, m$. Thus, ϕ_1, $\phi_2, \ldots, \phi_m$ are h_0-compatible.

The only if part. We have $\phi_1, \phi_2, \ldots, \phi_m$ compatible $\Rightarrow$ there exists h in H such that ϕ_1, $\phi_2, \ldots, \phi_m$ are h-compatible $\Rightarrow$ there exist $h_1^{(i)}$ and $h_2^{(i)}$ such that $\phi_i = h f h_1^{(i)} f h_2^{(i)}$ for $i = 1, 2, .., m$.

$h \in H \Rightarrow$ there exists $t : R_r \times R_r \to R_r$ such that $h(pr+q) = pr+t(p,q)$ and $t(p,.)$ is a permutation of R_r, for all p, q in R_r, after Lemma 9.3.Thus, (i) is satisfied.

To prove (ii), We reason by contradiction.

If, for some i, $\alpha_i(p,q) = \alpha_i(p',q')$ and $p \neq p'$, and $t(p,q) = t(p',q')$, then $h_2^{(i)}(\alpha_i(p,q)r+t(p,q)) = h_2^{(i)}(\alpha_i(p',q')r+t(p',q'))$. After Lemma 9.4, $h_2^{(i)}(\alpha_i(p,q)r+t(p,q)) = \alpha_i(p,q)r+\beta_i(p,q) = \phi_i(pr+q)$ and

$h_2^{(i)}(\alpha_i(p',q')r+t(p',q')) = \alpha_i(p',q')r+\beta_i(p',q') = \phi_i(p'r+q')$. Therefore, $\phi_i(pr+q) = \phi_i(p'r+q')$.

$\Rightarrow pr+q = p'r+q' \Rightarrow p = p'$ and $q = q' \Rightarrow p = p'$. Contradiction.

Thus,(ii) must be true. □

REFERENCES

[ABI80] M. A. Abidi, "Parallel processing: The interconnection problem," Ph.D. dissertation, Wayne State University, Detroit, Michigan, 1980.

[AHO74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, "The design and analysis of computer algorithms," Addison-Wesley, Reading Mass. 1974.

[AKS83] M. Ajtai, J. Kanlós and E. Szémerédi, "Sorting in clog n parallel steps," Combinatorica 3, pp. 1-19, 1983.

[ARD82] B. W. Arden and R. Ginosar, "MP/C a multiprocessor/computer architecture," IEEE Trans. Comput., C-31, 5, pp. 455-473, May 1982.

[ATT84] J. B. Attili, "On the construction of conjugation maps for functionally equivalent interconnection networks," Masters thesis, Rensselaer Polytechnic Institute, Troy, NY, Sept. 1984.

[BAT68] K. E. Batcher, "Sorting networks and their applications," Proc. Spring Joint Computer Conf., AFIPS Conf. (Montvale, N.J. :AFIPS Press, 1968), vol. 32, pp. 307-314.

[BAT74] K. E. Batcher, "STARAN parallel processor system hardware," AFIPS Conf. Proc. 1974 National Comput. Conf., May 1974, pp. 405-410.

[BEN65] V. E. Benes, "Mathematical theory of connecting networks and telephone traffic," Academic Press, New York, 1965.

[BHA81] V. C. Bhavsar, "Some parallel algorithms for Monte Carlo solutions of partial differential equations," *Advances in Computer Methods for Partial Differential Equations*, vol. 4, R. Vichnevestky and R. S. stepleman (Ed.), New Brunswick: IMACS, pp. 135–141, 1981.

[BHA77] V. C. Bhavsar and V. V. Kanetkar, "A multiple microprocessor system (MMPS) for the MonteCarlo solution of partial differential equations," *Advances in Computer Methods for Partial Differential Equations*, vol. 2, R. Vichnevestky (Ed.), New Brunswick: IMACS, pp. 205–213, 1977.

[BHA79] V. C. Bhavsar and A. J. Padgaonkar, "Effectiveness of some parallel computer architectures for Monte Carlo solution of partial differential equations," *Advances in Computer Methods for Partial Differential Equations*, vol. 3, R. Vichnevestky and R. S. stepleman (Ed.), New Brunswick: IMACS, pp. 259–264, 1979.

[CUR49] J. H. Curtiss, "Sampling methods applied to differential and difference equations," *Proc. Seminar Scientific Computation*, I.B.M., 1949.

[DEG81] R. D. Degroot, "Mapping computation structures onto SW-banyan networks," Doctoral dissertation, Univ. of Texas, Austin, Texas, Dec. 1981.

[FEN81] T. Feng, "A survey of interconnection networks," Comput., vol. 14, No. 12, pp. 12–27, Dec. 1981.

[GOK73] L. R. Goke and G. J. Lipovski, "Banyan networks for patitioning multiprocessor systems," Proc. 1st Annual Symp. on Comp. Arch., Dec. 1973, pp. 21-28.

[HAL70] J. H. Halton, "A retrospective and prospective survey of the Monte Carlo method," *SIAM Rev.*, vol. 12, Jan. 1970.

[HAM64] J. M. Hammersly and D. C. Handscomb, *Monte Carlo Methods*. London, England: Methuen, 1964.

[LAN76] T. Lang and H. S. Stone, "A shuffle-exchange network with simplified control," IEEE Trans. Comput., vol. C-25, pp. 55–65, Jan. 1976.

[LAW75] D. K. Lawrie, "Access and alignment of data in an array processor," IEEE Trans. Comput., vol. C-24, pp. 1145-1155, Dec. 1975.

[LEN78] J. Lenfant, "Parallel permutations of data: A Benes network control algorithm for frequently used permutations," IEEE Trans. Comput., vol. C-27, No. 7, pp. 637-647, Jul 1978.

[LIP77] G. J. Lipovski and A. Tripathi, "A reconfigurable varistructure array processor," Proc. of the 1977 IPCC, Aug. 1977, pp. 165-174.

[LIP79] G. J. Lipovski, "The architecture of the banyan switch for TRAC," Department of Electrical Engineering, TRAC Report TR-7, Univ. of Texas, Austin, Texas, JAn. 1979.

[NAS80] D. Nassimi and S. Sahni, "Parallel algorithms to set-up the Benes permutation network," Univ. of Minnesota, minneapolis, Tech. Rep. 79-19, June 1979; also in proc. Workshop on Interconnection Networks for

Parallel and Distributed Processing, Purdue University, Lafayette, IN, Apr. 1980.

[NAS81] D. Nassimi and S. Sahni, "A self-routing Benes network and parallel permutation algorithms," IEEE Trans. Comput., vol. C-30, No. 5, pp. 332-340, May 1981.

[ORU85a] A. Y. Oruc and M. Y. Oruc, "On testing permutation network isomorphisms," Proc. of the 1985 Intl. Conf. on Par. Proc., pp. 361-368.

[ORU85b] _____ "Equivalence relations among interconnection networks," Journal of Par. and Dist. Comput., 2, pp. 30-49, 1985.

[ORU85c] A. Y. Oruc, M. Y. Oruc and N. Balabanian, "Reconfiguration algorithms for interconnection networks," IEEE Trans. Comput., Vol. C-34, pp. 773-776, Aug. 1985.

[PEA76] M. C. Pease, "The indirect binary n-cube multiprocessor array," IEEE Trans. Comput., Vol. C-26, pp. 458-473, May 1976.

[PRM81] U. V. Premkumar, "A theoretical basis for the analysis and partitioning of regular SW banyans," Doctoral dissertation, Univ. of Texas, Austin, Texas, 1981.

[SAD74] E. Sadeh and M. A. Franklin, " Monte Carlo solutions of partial differential equations by special purpose digital computer," *IEEE Trans. Comput.*, C–23, pp. 389–397, Apr. 1974.

[SAD] E. Sadeh, "A Monte Carlo computer for solution of partial differential equations," M.S. thesis,Washington Univ., St. Louis, Mo.

[SCH74] M. A. Schlumberger, "De Bruijn cmmunication networks," Ph.D. dissertation, Dep. Comput. Sci., Stanford Univ., Stanford, CA, June 1974.

[SCH80] J. T. Schwartz, "Utracomputers," ACM Trans. on Programming Languages and Systems, Vol. 2, No. 4, pp. 484-521, Oct. 1980

[SIE79] H. J. Siegel, "A model of SIMD machines and a comparison of various interconnection networks," IEEE Trans. Comput., C-28, 12, pp. 907-917, Dec. 1979.

[SIE81] H.J. Siegel and R.J. McMillan, "Using the augmented DAta manipulator network in PASM," Computer, Vol. 14, Feb. 1981, pp. 25-33.

[SIE78] H. J. Siegel and D. Smith, "Study of multistage SIMD interconnection networks," Proc. Fifth Annual Symp. Comp. Arch., Apr. 1978, pp. 223-229.

[STO71] H. S. Stone, "Parallel processing with the perfect shuffle," IEEE Trans. Comput., vol. C-20, pp. 153-161, Feb. 1971.

[THU74] K. J. Thurber, "Interconnection networks – A survey and assesment," AFIPS Conf. Proc., vol. 43, 1974, pp. 909–919.

[WAK68] A. Waksman, "A permutation network," JACM, vol. 15, No. 1, pp. 159-163, Jan 1968.

[WU80a] C. W. Wu and T. Feng, "On a class of multistage interconnection networks," IEEE Trans. Comput., Vol. C-29, pp. 694-702, Aug 1980.

[WU80b] _____"The reverse-exchange interconnection network," IEEE Trans. Comput., Vol. C-29, pp. 801-811, Sept. 1980.