On Coding the States of Sequential Machines with the Use of Partition Pairs

Zahle, Torben U.

Published in:
I E E E Transactions on Computers

Link to article, DOI:
10.1109/PGEC.1966.264309

Publication date:
1966

Document Version
Publisher's PDF, also known as Version of record

constraints to reduce their complexity; in general, it is worthwhile to resort to a tabular approach [4] to apply the simplifying rules and reduce the program as far as possible. Also, sometimes only the preprocessing phase is needed to obtain a solution. In the minimal closed partition program, the only reducing mechanism is dominance between constraints, which, in general, affords at most the elimination of some redundant constraint, but only in trivial cases permits to arrive at a complete solution.

Finally, we will mention that the Boolean algebra method, applicable in solving the minimal covering problem [4], is also applicable to solve the minimal partition problem. However, while in the minimal covering problem the individual Boolean expressions corresponding to the constraints are simply sums of literals, this is not true for the expressions corresponding to the covering constraints of the minimal partition problem. For example, the covering constraint,

$$y_1 + y_2 + y_3 \leq 1$$

becomes

$$\bar{y}_1\bar{y}_2\bar{y}_3 + y_1\bar{y}_2\bar{y}_3 + \bar{y}_1y_2\bar{y}_3 + \bar{y}_1\bar{y}_2y_3 = 1. \quad (8)$$

In (8), the y's are Boolean variables, and

$y_i = 1$ signifies that class C_i is selected
$y_i = 0$ signifies that class C_i is not selected.

"Multiplying out" the product of the individual expressions to a "sum of products" expression is therefore a more cumbersome process, which makes this method less attractive, especially for hand computation.

Acknowledgment

The author is grateful to J. Gimpel of Princeton University for a discussion on this problem.

References

[1] A. Grasselli, "Finite memory span covers of incompletely specified sequential networks," presented at the 1965 IFIP Cong., New York, N.Y.
[2] A. Grasselli, "Sur la realization des tableaux incomplets par des reseaux sequentiels à memoire finie," in *Algebre de Boole et machines logiques*, J. Kuntzmann and P. Naslin, Eds. Paris: Dunod, in press.
[3] M. C. Paull and S. H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions," *IRE Trans. on Electronic Computers*, vol. EC-8, pp. 356–367, September 1959.
[4] A. Grasselli and F. Luccio, "A method for minimizing the number of internal states in incompletely specified sequential networks," *IEEE Trans. on Electronic Computers*, vol. EC-14, pp. 350–359, June 1965.
[5] J. Hartmanis, "On the state assignment problem for sequential machines. I," *IRE Trans. on Electronic Computers*, vol. EC-10, pp. 157–165, June 1961.
[6] R. A. Gomory and W. J. Baumol, "Integer programming and pricing," *Econometrica*, vol. 28, pp. 521–550, 1960.

On Coding the States of Sequential Machines with the Use of Partition Pairs

TORBEN U. ZAHLE

***Abstract*—This article introduces a new technique of making state assignment for sequential machines. The technique is in line with the approach used by Hartmanis [1], Stearns and Hartmanis [3], and Curtis [4]. It parallels the work of Dolotta and McCluskey [7], although it was developed independently. The paper describes a procedure for making assignments based on partition pairs with successive choice of the partition.**

Manuscript received March 15, 1965; revised November 15, 1965.
The author is with the Laboratory for Pulse and Digital Techniques, Technical University of Denmark, Copenhagen, Denmark.

Introduction

It will be assumed that the reader is familiar with the works named in the abstract. Some usual assumptions are made, namely that realization is based on two-level AND-OR logic and delay elements, variables and their complements are available without additional cost, the input is coded in advance, and the output is to be handled separately. The fundamental idea of the technique is based on the following observations.

1) Any assignment whatsoever has to be built up of two-block partitions corresponding to each secondary variable.

2) A two-block partition corresponding to a y-variable fixes the placement of ones and zeros in the Y-diagram when we keep the same ordering of states.

Example

Machine A:

	0	1
a	d	c
b	f	c
c	e	b
d	b	e
e	a	d
f	c	d

	y_1	y_2	y_3
$a \rightarrow 0$	0	1	
$b \rightarrow 0$	1	1	
$c \rightarrow 1$	1	1	
$d \rightarrow 1$	1	0	
$e \rightarrow 0$	1	0	
$f \rightarrow 0$	0	0	

	0			1		
	1	1	0	1	1	1
	0	0	0	1	1	1
	0	1	0	0	1	1
	0	1	1	0	1	0
	0	0	1	1	1	0
	1	1	1	1	1	0
	Y_1	Y_2	Y_3			

or for y_1:

	0	1
$a \rightarrow 0$	1	1
$b \rightarrow 0$	0	1
$c \rightarrow 1$	0	0
$d \rightarrow 1$	0	0
$e \rightarrow 0$	0	1
$f \rightarrow 0$	1	1
y_1	Y_1	

which gives

$P = \overline{cd}\ \overline{abef}$

$C_0 = \overline{af}\ \overline{bcde}$

$C_1 = \overline{abef}\ \overline{cd}.$

The Y_1-diagram shown will be the same no matter how we choose the other variables. The partition made by ones and zeros in the 0-column is called C_0. In general, the C-partition has as index the input of the column. The notation P- and C-partition is in accordance with the definition of Curtis [4], p. 336. The logical product of the C-partitions corresponding to each column is the same as Curtis' C-partition.

3) The aim of the assignment procedure is to choose P-partitions so that the Y-functions can be realized with as few and as large subcubes as possible, or equivalently, so that the blocks of the C-partitions can easily be realized by the P-partitions.

Definition of Potentially Useful P-Partitions

λ: It will be potentially useful if two columns with neighboring input codes have identical C-partitions, because we then can realize the two columns at the same time. To find P-partitions with this property we construct a partition λ (which as index has the input variables which are common to the two columns) by taking the two neighbor columns and *identifying states of the same row.* Any $P \geq \lambda$ will then have the property that the two corresponding C-partitions are identical. This corresponds to Dolotta and McCluskey's identity in adjacent columns. If there are only two inputs, the λ-partition is identical with the partition Hartmanis introduced in [2].

I: We get another potentially useful partition if a whole column is entirely zeros or ones. These partitions we find by putting *all states of a column in the same group I*. Any P which does not split the group I will then have the desired property. This corresponds to Dolotta and McCluskey's all-zero or all-one entry.

C: We get a potentially useful partition if one of the C-partitions is identical with the P-partition. This can happen when the group of ones in the P-partition is identical with the group of ones in the C-partition. To find P-partitions where this is the case we construct a partition "C" by *identifying every state with the state it goes to* for the column in question. Any $P \geq$ "C" will then have the desired property. This corresponds to Dolotta and McCluskey's "identity with base entry."

$\bar{C}$: It would be equally desirable if the group of ones in the P-partition was identical with the group of zeros of the C-partition. To find all P's where this is the case we must construct a grouping "$\bar{C}$" by *placing each state and the state it goes to in opposing groups*. Any P which keeps the opposing groups, in the two different blocks of the P-partition, will have the desired property. This corresponds to Dolotta and McCluskey's "identity with complement of base entry."

S: Any C-partition of a particular column is $\geq$ the partition "S" made by *identifying states which go to "the same state in the particular column."* Therefore, any P-partition which should have a possibility of realizing a C-partition of the column must be $\geq S$. This corresponds to Dolotta and McCluskey's consideration of identity with already chosen column.

Example

Machine A:

	0	1
a	d	c
b	f	c
c	e	b
d	b	e
e	a	d
f	c	d

λ) λ: $\overline{dcfa}\ \overline{eb}$

$a \to 0$, $b \to 1$, which gives $c \to 0$, $d \to 0$, $P = \overline{be}\ \overline{acdf}$, $e \to 1$, $f \to 0$

		0	1
a	a	0	0
b	b	0	0
c	c	1	1
d	d	1	1
e	e	0	0
f	f	0	0

Y

$\underline{\underline{C_0 = C_1 = \overline{cd}\ \overline{abef}}}$

I) For column "1" we get $a \to 1$, $b \to 0$, $c \to 0$, $d \to 0$, $e \to 0$, $f \to 1$

I: $\overline{cbed}$ which gives $P = \overline{af}\ \overline{bcde}$

		0	1
a	a	0	0
b	b	1	0
c	c	0	0
d	d	0	0
e	e	1	0
f	f	0	0

Y

$\underline{\underline{C_1 = \bar{O}\ \bar{I}}}$

C) For column "1" we get $a \to 1$, $b \to 1$, $c \to 1$, $d \to 0$, $e \to 0$, $f \to 0$

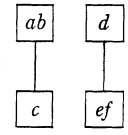
C: $\overline{acb}\ \overline{def}$ which gives $P = \overline{abc}\ \overline{def}$

		0	1
a	a	0	1
b	b	0	1
c	c	0	1
d	d	1	0
e	e	1	0
f	f	1	0

Y

$\underline{\underline{C_1 = P}}$

$\bar{C}$) For column "1" we get

$\bar{C}$: 

which gives $P = \overline{cd}\ \overline{abef}$ and $P = \overline{abd}\ \overline{cef}$ because the group ab can combine with $\bar{d}$ as well as $\overline{ef}$.

$a \to 0$, $b \to 0$, $P = \overline{cd}\ \overline{abef}$, $c \to 1$, $d \to 1$, $e \to 0$, $f \to 0$

		0	1
a	a	1	1
b	b	0	1
c	c	0	0
d	d	0	0
e	e	0	1
f	f	1	1

Y

$\underline{\underline{\bar{C}_1 = P}}$

S) For column "1" we get S_1: $\overline{ab}\ \bar{c}\ \bar{d}\ \overline{ef}$. We notice that all C_1's above are $\geq S_1$.

Assignment Procedure

For a given machine we now construct the λ-partitions, and for each column of the machine the group I, the grouping $\bar{C}$, and the partitions C and S. These are placed in the upper half of a diagram called *PCC*.

Machine A:

	0	1
a	d	c
b	f	c
c	e	b
d	b	e
e	a	d
f	c	d

PCC diagram

Machine A: P	C_0	C_1
λ: $\overline{cdfa}\ \overline{be}$	I: $\overline{abcdef}$ —	I: $\overline{bcde}$
	C: $adbfce$ —	C: $\overline{abc}\ \overline{def}$
	C: abc | dfe	$\bar{C}$: ab | c, d | ef
	S_0: $\bar{a}\ \bar{b}\ \bar{c}\ \bar{d}\ \bar{e}\ \bar{f}$	S_1: $\overline{ab}\ \bar{c}\ \bar{d}\ \overline{ef}$
1 0	1 0	1 0
be $acdf$	cd $abef$	cd $abef$
af $bcde$	$\overline{be}$ $\overline{acdf}$	$\bar{O}$ $\bar{I}$
abc def	def abc	abc def
abd cef	ade bcf	cef abd
cd $abef$	af $bcde$	$abef$ cd
ab $cdef$	de $abcf$	c $abdef$
ef $abcd$	bc $adef$	d $abcef$

All the potentially useful P-partitions are found, as described in the previous section, and listed in the lower half of the diagram. As an example the partition S_1: $\overline{ab}\ \overline{c}\ \overline{d}\ \overline{ef}$ have five partitions $\geq S_1$; that is, $P=\overline{ab}\ \overline{cdef}$, $P=\overline{cd}\ \overline{abef}$, $P=\overline{ef}\ \overline{abcd}$, $P=\overline{abc}\ \overline{def}$, and $P=\overline{abd}\ \overline{cef}$.

For each P-partition, the corresponding C-partitions are constructed as illustrated in the Introduction.

The ones and zeros in the middle of the diagram indicate that we always put the group which corresponds to the ones first.

When the problem is worked by hand it is not necessary to list all potentially useful P-partitions. We can limit ourselves to listing the most useful P-partitions, that is, partitions which satisfy several criteria at the same time. In our example we have not listed all the partitions to which S_0 leads because this would be the set of all possible P-partitions; thus, we might say that S_0 does not help us in finding particularly useful P-partitions.

In the diagram we now underline the C-partitions which can be easily realized, i.e., where one of the simplifications that are described in the previous section occur. If the simplication occurs because $C_0=C_1$, we underline both C's with a dotted line; this counts as one underlining.

The procedure now is to choose P-partitions one at a time so that at each step we secure the maximum number of easy C-partitions.

1) Choose the P-partition with the maximum number of underlinings. If two P's have the same number of underlinings preference is put on the λ and I simplifications because they can be realized with fewer diodes than the other simplifications.

2) Put a minus by all the P's that are not compatible with the chosen P.

3) Underline

a) all C's that are identical with the chosen P

b) all C's that are identical with the chosen C in the same column, provided this simplification does not spoil a λ-simplification, and

c) every P with one line for each of the now chosen C's that it realizes.

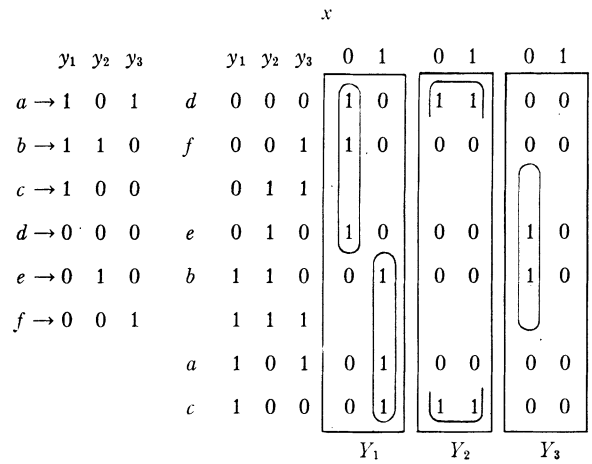
Note that each C-partition can only be underlined once.

The procedure continues until all P's are chosen. If among the potentially useful P-partitions we cannot find all the P-partitions we need, we choose the last P-partitions arbitrarily.

By carrying out this procedure we end up with the following diagram:

	P	C_0	C_1
	1 0	1 0	1 0
$y_2 \longrightarrow$	*be acdf*	*cd abef*	*cd abef*
$y_3 \longrightarrow$	*af bcde*	*be acdf*	*O I*
$y_1 \longrightarrow$	*abc def*	*def abc*	*abc def*
	abd cef	*ade bcf*	*cef abd*
	cd abef	*af bcde*	*abef cd*
—	*ab cdef*	*de abcf*	*c abdef*
—	*ef abcd*	*bc adef*	*d abcef*

Having thus found y_1, y_2, and y_3 we now have the application equations:



$Y_1 = \bar{x}\bar{y}_1 + xy_1$

$Y_2 = \bar{y}_2\bar{y}_3$ 10 diodes

$Y_3 = \bar{x}\ y_2.$

One may notice that by comparison of the three selected y-variables in the PCC-diagram with the Y_1, Y_2, and Y_3 tables, the application equations may be found directly from the PCC diagram. Whether it is advantageous to complement any of the variables can also be seen as easily from the PCC diagram as from the Y tables.

The method can also handle "don't cares." This is done by omitting the states which are "don't cares" in the C-partitions of a particular column. These "don't cares" are, in the course of the procedure, associated with either the block of ones or the block of zeros of each C-partition, depending on what is most convenient.

Results

The author has tried the method on all the machines of the articles [1]–[6]. In nearly all cases we found a result that required the same number of diodes or less. In the cases where we did not, it was because by choosing the P-partitions one at a time, we did not find a set of P-partitions which realized each other's C-partitions and, therefore, the simplification could only be seen if the set of P-partitions was taken as a whole. This is a difficulty which is intrinsically connected with choosing the P-partitions one at a time. However, when the method is worked by hand we do not have to stick strictly to this procedure. Curtis, who examines all possible sets of P-partitions in his (C, P)-method, overcomes this difficulty in principle. His aim, however, is not directly to minimize diode count, and so he finds a realization with 55 diodes (or 42 with optimal choice of complementing variables and of subcubes) while this much shorter method finds a realization with 38 diodes. To demonstrate our method for a bigger machine than that used above, this example is shown in the Appendix.

The procedure is offered with incomplete evaluation of its usefulness. There are many unanswered questions: a) Will the procedure be equally efficient on larger machines? b) Does the procedure have any value for problems for which no simple assignment exists? c) How does it compare with earlier procedures when there are many "don't cares"?

The method has not been programmed on a computer, but it is believed to be suited for programmation because it is rigorous and consists of a large number of simple steps.

Discussion

A drawback of the method is that it is only concerned with maximizing the number of very easily realizable C's, while it leaves to chance whether the remaining C's are realized with few or many subcubes. This again is connected with the selection of the P-partitions one at a time; it is of little value to see whether a C-partition can be realized by a set of a few chosen P's, because we don't know if

later we will get a P-partition which can realize the C-partition in an even more simple way.

The use of a criterion which evaluated the ability to get subcubes with common input variables was considered at a certain point. This showed extraordinary little efficiency. The explanation must be found in the fact that the inputs are coded in advance, and we therefore have too little freedom of choice to make larger subcubes with common input variables.

Conclusion

The advantages of the method are that it surveys the problem directly in the PCC diagram; it is relatively short; it goes to the center of the problem, which is to obtain a small number of large subcubes; it utilizes common subcubes; it is sufficiently straightforward to be programmed on a computer; if the problem is worked by hand, the Y-functions can be written down directly from the PCC diagram; the method can handle "don't cares"; and, it is particularly efficient if a very simple solution exists.

Appendix

PCC procedure on machine B in Curtis [4]

Machine B:

Curtis' Result:

	00	01	11	10
a	d	e	b	e
b	a	d	e	c
c	f	c	b	e
d	c	e	a	c
e	b	c	e	e
f	a	f	c	c

$$Y_1 = \bar{x}_1\bar{x}_1\bar{y}_1 + \bar{x}_1x_2y_1 + x_1x_2\bar{y}_1\bar{y}_3$$

$$Y_2 = \bar{x}_1\bar{x}_2\bar{y}_3 + \bar{x}_1x_2 + x_1\bar{x}_2 + x_1x_2y_3 \quad \underline{55 \text{ diodes}}$$

$$Y_3 = \bar{x}_1\bar{x}_2\bar{y}_1y_2 + \bar{x}_1x_2\bar{y}_1\bar{y}_2 + \bar{x}_1x_2y_1y_2 + \bar{x}_1\bar{x}_2\bar{y}_1 + x_1x_2\bar{y}_1 + x_1x_2\bar{y}_2$$

With optimal choice of complementing variables and of subcubes it gives $\underline{\underline{42 \text{ diodes}}}$

P	C_{00}	C_{01}	C_{11}	C_{10}
$\lambda\bar{x}_1$: $\overline{abcdef}$ —	I: $\overline{abcdf}$ —	I: $\overline{cdef}$	I: $\overline{abce}$	I: $\overline{ce}$
λx_1: $\overline{abce}$ $\overline{df}$	C: $\overline{adcfbe}$ —	C: $\overline{aecdb}$ $\overline{f}$ —	C: $\overline{abecdf}$ —	C: $\overline{acebdf}$ —
$\lambda\bar{x}_2$: $\overline{bdef}$ $\overline{ac}$	$\bar{C}$: ace	$\bar{C}$: c* —	$\bar{C}$: e —	$\bar{C}$: c —
λx_2: $\overline{abcdef}$ —	dfb	c	e	c
	S: $\bar{a}$ $\overline{bf}$ $\bar{c}$ $\bar{d}$ $\bar{e}$	S: $\overline{ad}$ $\bar{b}$ $\overline{ce}$ $\bar{f}$	S: $\overline{ac}$ $\overline{be}$ $\bar{d}$ $\bar{f}$	S: $\overline{ace}$ $\overline{bdf}$
1 0	1 0	1 0	1 0	1 0
$y_2 \rightarrow$ df $abce$	ac $dbef$	bf $acde$	0 1	0 1
— ac $bdef$	bdf ace	ce $abdf$	df $abce$	bdf ace
$y_1 \rightarrow$ ab $cdef$	bef acd	0 1	acd bef	0 1
— ace bdf	bdf ace	$acde$ bf	$bdef$ ac	1 0
— ce $abdf$	d $abcef$	$acde$ bf	bef acd	1 0
— ad $bcef$	abf cde	b $acdef$	d $abcef$	0 1
— af $bcde$	bcf ade	f $abcde$	d $abcef$	0 1
— bd $acef$	ae $bcdf$	b $acdef$	ac $bdef$	0 1
— bf $acde$	ce $abdf$	f $abcde$	ac $bdef$	0 1
— adf bce	$abcf$ de	bf $acde$	d $abcef$	0 1
— abf cde	$bcef$ ad	f $abcde$	acd bef	0 1
— abd cef	$abef$ cd	b $acdef$	acd bef	0 1
— ae $bcdf$	bf $acde$	ad $bcef$	bde acf	ace bdf
— cd $abef$	ad $bcef$	bce adf	f $abcde$	bdf ace
— de $abcf$	a $bcdef$	abd cef	be $acdf$	ace bdf
ade bef	abf cde	abd cef	bde acf	ace bdf
$y_3 \rightarrow$ acd bef	$abdf$ ce	bce adf	df $abce$	bdf ace
— be $acdf$	e $abcdf$	ad $bcef$	$abce$ df	ace bdf
acf bde	$bcdf$ ae	cef abd	df $abce$	ace bdf

* We stop at once when we see that c should be in both blocks.

	y_1	y_2	y_3	
$a \rightarrow$	1	1	0	$Y_1 = \overline{X}_1\overline{X}_2Y_3 + X_1X_3\overline{Y}_3$
$b \rightarrow$	1	1	1	$Y_2 = \underline{\overline{X}_1\overline{X}_2Y_3} + \overline{Y}_2\overline{Y}_3 + X_2\overline{Y}_1Y_2 + X_2Y_1\overline{Y}_3 + X_1$ 38 diodes
$c \rightarrow$	0	1	0	$Y_3 = \overline{X}_2\overline{Y}_1Y_2 + \overline{X}_1X_2\overline{Y}_2 + \underline{X_2Y_1\overline{Y}_3} + X_1X_2Y_2 + X_1Y_1\overline{Y}_3$
$d \rightarrow$	0	0	0	
$e \rightarrow$	0	1	1	We underline products which have been realized already.
$f \rightarrow$	0	0	1	

The procedure can arbitrarily start with $P = \overline{ab}\ \overline{cdef}$ or $P = \overline{df}\ \overline{abce}$. The second choice would have led to a solution with *47 diodes*, but in such a situation one would normally try both cases.

REFERENCES

[1] J. Hartmanis, "On the state assignment problem for sequential machines. I," *IRE Trans. on Electronic Computers*, vol. EC-10, pp. 157–165, June 1961.
[2] ——, "Maximal autonomous clocks of sequential machines," *IRE Trans. on Electronic Computers (Correspondence)*, vol. EC-11, pp. 83–86, February 1962.
[3] R. E. Stearns and J. Hartmanis, "On the state assignment problem for sequential machines. II," *IRE Trans. on Electronic Computers*, vol. EC-10, pp. 593–603, December 1961.
[4] H. Allen Curtis, "Multiple reduction of variable dependency of sequential machines," *J. Assoc. for Computing Machinery*, vol. 9, pp. 324–344, July 1962.
[5] D. B. Armstrong, "A programmed algorithm for assigning internal codes to sequential machines," *IRE Trans. on Electronic Computers*, vol. EC-11, pp. 466–472, August 1962.
[6] ——, "On the efficient assignment of internal codes to sequential machines," *IRE Trans. on Electronic Computers*, vol. EC-11, pp. 611–622, October 1962.
[7] T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential circuits," *IEEE Trans. on Electronic Computers*, vol. EC-13, pp. 549–562, October 1964.

A Design for (d, k) Graphs

HENRY D. FRIEDMAN, SENIOR MEMBER, IEEE

A (d, k) graph is defined as a linear graph of diameter k whose nodes are each of degree d or less. The degree of a node is the number of branches incident at that node. The diameter of a graph is the max-min distance (measured in branches) between all pairs of nodes. Elspas [1] has been interested in finding (d, k) graphs for which the number of nodes $n(d, k)$ is a maximum. Akers [2] gave an elegant method for constructing $(d, d-1)$ graphs of degree d and diameter $d-1$, which results in a graph with

$$\binom{2d-1}{d}$$

nodes. Akers' graphs, though not demonstrably maximal, are equal or superior to any graphs reported in Elspas.

In this note we construct (d, k) graphs that establish a general lower bound on $n(d, k)$ when d is sufficiently large. We first construct a simple hierarchic tree with $[k/2+1]$ levels of nodes (using $[\ ]$ to indicate "greatest integer in"), in which all nodes have degree d except for the lowest level, where all nodes have degree 1. (When $d=5$ and $k=4$, we would have $[6/2]=3$ hierarchic levels. (See Fig. 1.) To construct the desired (d, k) graph we take d identical hierarchic trees and cause them to share, in common, their lowest-level nodes (see Fig. 2). Thus, the full (d, k) graph consists of d trees joined at these nodes without changing the order of the branches.

It is obvious that each node of the resulting graph has degree d. To show that the graph has diameter k, we note that, within any tree, the shortest distance between any two lowest-level nodes is at most k branches. Therefore, any two given nodes of the full graph lie on a loop of $2k$ or fewer branches; hence, they cannot be farther apart than k branches.

Manuscript received September 3, 1965; revised December 20, 1965.
The author is with the Data Processing Department, Smithsonian Astrophysical Observatory, Cambridge, Mass. He was formerly with the Applied Research Laboratory, Sylvania, Waltham, Mass.

The number of nodes at the different levels of a hierarchic tree are $1, d, d(d-1), d(d-1)^2, \cdots, d(d-1)^u$, where $u = [k/2-1]$. By counting each level d times, except for the lowest level, which is counted only once, we easily find that the number of nodes in the full (d, k) graph is (for $d>2$)

$$N = \frac{2d}{d-2}\left((d-1)^{[k/2]} - 1\right).$$

This expression is not particularly useful when d is small (compare with Table I in Elspas [1]). However, when d is sufficiently large, N can be used as a general lower bound on $n(d, k)$. It is interesting, also, to compare the expression for N with the Moore graph upper-bound expression [1]

$$n_M(d, k) = \frac{d(d-1)^k - 2}{d-2}.$$

When $k=d-1$, comparing N with Akers' number

$$\binom{2d-1}{d}$$

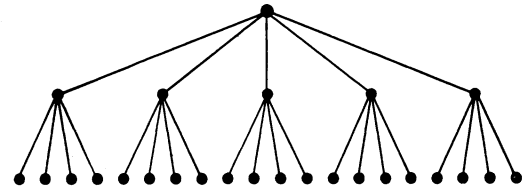


Fig. 1. Hierarchic tree for $d=5$, $k=4$.

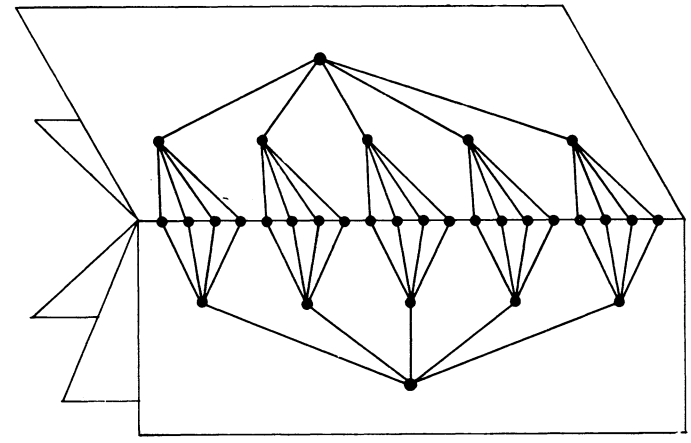


Fig. 2. Construction of (5, 4) graph. Each plane contains a tree.