

TRANSFORMATION OF HIERARCHICAL STRUCTURE IN
WARNIER-ORR DIAGRAMS: EXAMPLES AND RULES

by

SHU-MEI LIN

BACHELOR OF LAW, NATIONAL TAIWAN UNIVERSITY, 1969

A MASTER'S REPORT

Submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE


Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1982

Approved by :


Major Professor

Spec.
Coll.
LD
2668
R4
1982
L56
C.2

A11202 246865

1

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my advisor, Dr. David A. Gustafson, for his help, guidance and patience to make this report possible to complete. Thanks are also given to my patient family.

TABLE OF CONTENTS

1. Introduction	4
1.1 Introduction	4
1.2 The Warnier-Orr Diagram forms	6
2. Transformations	11
2.1 The important of transformations	11
2.2 Example	12
2.3 Semantics of brace, "{"	16
3. Specific transformations	20
3.1 motivation	20
3.2 Example	20
3.2.1 Transform three-level diagram into one-level diagram	20
3.2.2 Decompose one-level diagram into more levels diagram	27
3.3 Transformation rules	31
3.3.1 Rules for combining the two right most levels	31
3.3.2 Rules for decomposing one level diagram into diagram with several levels	34
4. Informal proof of the specific transformations	37
5. Conclusion	46
Selected bibliography	48
Appendix A	49

LIST OF FIGURES

1. Three-level example diagram of Warnier-Orr Diagram .	22
2. Transformed diagram of figure 1	24
3. Transformed diagram of figure 2	26
4. Transformed diagram of figure 3	28
5. Transformed diagram of figure 4	30
6. Simplified diagram of figure 1	37
7. Simplified diagram of figure 2	38
8. Simplified diagram of figure 3	40
9. Simplified diagram of figure 4	42
10. Simplified diagram of figure 5	44

CHAPTER 1

INTRODUCTION

1.1 Introduction

Most of the software errors are made during the design phase [1]. If the program design is bad, it will take more time to find the errors, to debug and to test the program completely. Much of the cost overrun is the result of bad design. A well-designed program will have fewer errors during its implementation. Since it is well-designed, it will be easier to find and repair the errors. Getting a program to work is not sufficient, but getting it designed right is the most important thing. A well-designed program is more likely to operate correctly in all circumstances and to be understandable and modifiable [2].

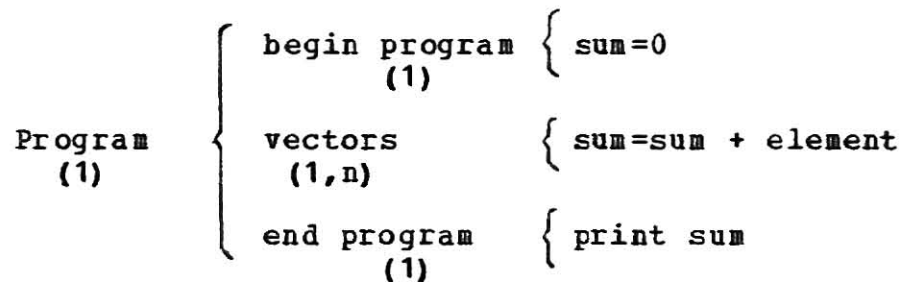
The major motivation for looking at design methodologies is the desire to reduce the cost of producing and maintaining software. Hence a successful design methodology must be able to produce a consistent, useful, low cost, reliable result; to allow for easy maintenance; and to be simple enough to let anyone to use it [3].

The ability to transform the design diagram to produce

alternative designs allows the designers to compare the alternative designs. Among the alternative designs, the designers could choose the one which is the lowest cost, the most reliable or the easiest to maintain.

One of the newer design tools is Warnier-Orr design diagrams [4]. This methodology was developed by Jean-Dominique Warnier and his group of researchers at Honeywell-Bull in Paris, France and Kenneth Orr at Langston, Kitch & Associates in Kansas. The Warnier-Orr diagrams began as a design method but was quickly noticed for its value as a documentation tool as well. This design tool aids in the understanding, design, verification, modification and optimization of programs and systems. The various forms of this diagrams are useful in many stages of a software development project, from requirements definition to the operation and use of the delivered project [4].

The Warnier-Orr diagram inherently incorporates hierarchy and tree structure into the design notation. An simple example is demonstrated below. The example computes the total sum of n vectors. It is necessary to initialize the value of sum as zero, then do the summation and finally print the sum. The unique features of this design tool are displayed below:



The design diagram is composed of three parts; "begin program" followed by "vectors" followed by "end program". In "begin program" section the sum is initiated. In "vectors" section the calculation is performed from 1 to n times, and in "end program" section the sum is printed.

In Warnier-Orr Diagram every brace has a hierarchical name placed to the left, (e.g., "program" and "vectors" are hierarchical names). A sequence of processes as other braces are placed to the right. Every brace can have "Begin" and "End" placed at the top and bottom respectively. Sometimes these are omitted from the diagrams for the sake of clarity. Repetition is in parentheses under the hierarchical name with letters, numbers or conditional statements using the "while" or "until" phrases [3].

1.2 The Warnier-Orr Diagram Forms

There are six constructs that are used to represent various data and process structures:

HIERARCHY: Hierarchy is the most fundamental and most valuable of the diagram structures. The brace here denotes decomposition and is read as "consists of" or "is composed of". A simple example is as following:

$$AA \left\{ BB \left\{ CC \left\{ \right. \right. \right.$$

It read as AA is composed of BB, BB is composed of CC.

SEQUENCE: To represent sequence on a Warnier-Orr diagram, it is only necessary to list elements serially within a level of hierarchy, and this implies a serial order of execution. The example is:

$$AA \left\{ \begin{array}{l} B1 \\ B2 \\ B3 \\ B4 \end{array} \right.$$

It read as AA is composed of B1 followed by B2 followed by B3 followed by B4.

REPETITION: The primary ways that repetition can be expressed are shown below:

$$AA \left\{ \begin{array}{c} BB \\ (1,b) \end{array} \right\} \qquad AA \left\{ \begin{array}{c} BB \\ (10) \end{array} \right\}$$

Each of these expressions indicates AA is broken down into many BBs. The numbers in parentheses under BB indicate that BB is a repeating subgroup within AA. These numbers also indicate the fewest and greatest number of times, or the exact number if known, that the subset BB can occur on the calculation, for example, (1,b) means the process of the subset of BB will occur from 1 to b times. "(10)" shows the process of the subset of BB will occur exactly 10 times.

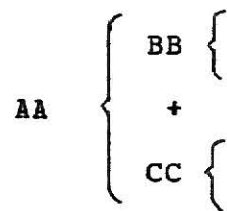
ALTERNATION: The alternation structure (or selection structure) is presented as following:

$$AA \left\{ \begin{array}{c} BB \\ (0,1) \\ \oplus \\ CC \\ (0,1) \end{array} \right\}$$

It means AA is composed of either BB or CC but not both. The (1,0) number of times under BB and CC indicates that AA may

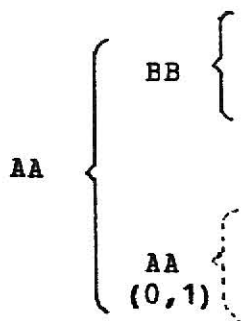
or may not be a particular one of these two (BB or CC); the exclusive OR symbol, \oplus , indicates that AA must be either BB or CC.

CONCURRENCY: Concurrency is used to express the situation that occurs when two or more things are in operation simultaneously. The plus sign, "+", is the concurrency operator in Warnier-Orr diagram. It indicates that the situations are not mutually exclusive.



The structure shown above means AA is composed of both BB and CC, and AA can have both cycles operating simultaneously.

RECURSION: The structure of recursion, which shows hierarchical replication (horizontal repetition on a Warnier-Orr Diagram), also appears with some frequency in systems work. A broken brace is used to show recursion on a Warnier-Orr Diagram.



The structure shows above means AA is composed of both BB and optionally itself. The broken brace indicates that each sub-AA is composed of BB and sub-sub-AAs and sub-sub-AA is composed of BB and sub-sub-sub-AAs, and so on.

Although the transformations presented in this report may be applicable to all of the structures mentioned above, the hierarchical structure will be the only one considered in this report. The hierarchical structure offers the most possibilities for alternate designs. It also is the most fundamental of the control structures.

CHAPTER 2

TRANSFORMATIONS

2.1 The Importance of Transformations

There have been many efforts in the area of transformations [5,6]. Transforming programs will allow the comparisons of different but equivalent programs. The transforms of a design diagram will allow the comparisons of the alternative designs.

The Warnier-Orr Diagrams possess the mathematical properties of associativity and commutativity [5]. The associative property is that two adjacent braces can always be associated as one brace and two hierarchic names can be combined into one name. A notation can be used to represent this concept:

$$A \left\{ B \left\{ C \left\{ \right. = (A,B) \left\{ C \left\{ \right. = A \left\{ (B,C) \left\{ \right. = (A,B,C) \left\{ \right.$$

The commutative property is the members of the hierarchy are commuted among themselves, the following notation defines this property:

$$A \left\{ B \left\{ C \left\{ \right. \right. = B \left\{ A \left\{ C \left\{ \right. \right. = B \left\{ C \left\{ A \left\{ \right. \right. = C \left\{ A \left\{ B \left\{ \right. \right.$$

Using the associative and commutative properties, the design structures can be transformed and the alternative designs can be compared. For example, in building systems, these properties can be used in combining multiple and incompatible hierarchic structures.

2.2 Example

To illustrate the commutative notion, Warnier-Orr Diagrams of two programs which consist of four levels of nested Do-loop structures are shown below [5].

$$\begin{array}{l} \text{Routine I} \left\{ \begin{array}{l} K \\ (k=1, 7) \end{array} \right\} \left\{ \begin{array}{l} V \\ (v=1, 7) \end{array} \right\} \left\{ \begin{array}{l} S \\ (s=1, 30) \end{array} \right\} \left\{ \begin{array}{l} T \\ (t=1, 30) \end{array} \right\} \\ \text{Routine II} \left\{ \begin{array}{l} S \\ (s=1, 29) \end{array} \right\} \left\{ \begin{array}{l} V \\ (v=1, 7) \end{array} \right\} \left\{ \begin{array}{l} K \\ (k=v, 7) \end{array} \right\} \left\{ \begin{array}{l} T \\ (t=3, 29) \end{array} \right\} \end{array}$$

Routine I has 44,100 loops to be performed, Routine II has 21,924 loops to be performed; the total will be 66,024 loops. It is desired to combine these two loop structures

into one routine while maintaining a sequential continuity of flow from beginning to end.

At first glance, these hierarchies are incompatible; the hierarchical names are identical but not in the same order, the indices of the repetition of identical hierarchical names have different ranges, and the indices of the repetition of K in Routine II are dependent on V.

By using the commutative property of Warnier-Orr diagrams, the hierarchies can be rearranged to a suitable and corresponding pattern, which are shown below:

$$\begin{array}{l}
 \text{Routine I} \quad \left\{ \begin{array}{c} V \\ (v=1, 7) \end{array} \right\} \left\{ \begin{array}{c} S \\ (s=1, 30) \end{array} \right\} \left\{ \begin{array}{c} T \\ (t=1, 30) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=1, 7) \end{array} \right\} \\
 \text{Routine II} \quad \left\{ \begin{array}{c} V \\ (v=1, 7) \end{array} \right\} \left\{ \begin{array}{c} S \\ (s=1, 29) \end{array} \right\} \left\{ \begin{array}{c} T \\ (t=3, 29) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=v, 7) \end{array} \right\}
 \end{array}$$

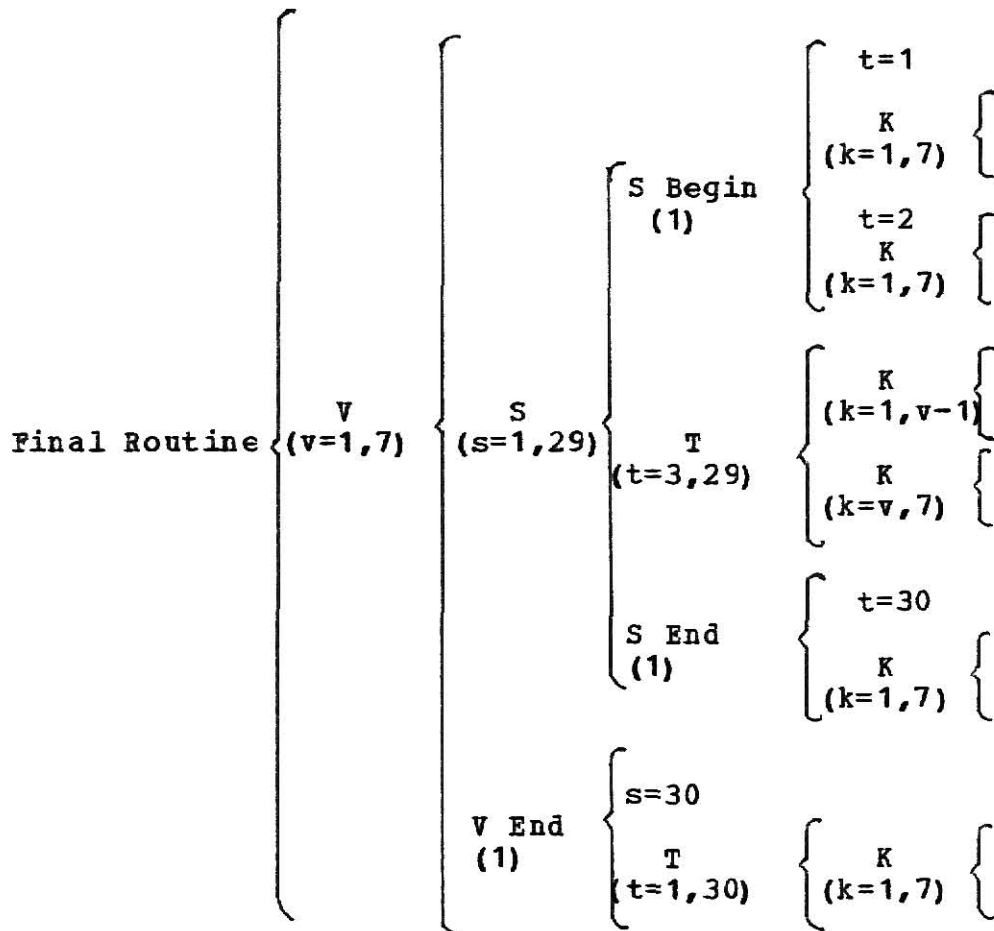
These two routines now can be combined into one routine by realizing that like hierarchical names can be combined. The new diagram is shown below:

$$\text{New Routine} \left\{ \begin{array}{c} V \\ (v=1,7) \end{array} \right\} \left\{ \begin{array}{c} S \\ (s=1,30) \end{array} \right\} \left\{ \begin{array}{c} T \\ (t=1,30) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=1,7) \end{array} \right\} \\ \left\{ \begin{array}{c} S \\ (s=1,29) \end{array} \right\} \left\{ \begin{array}{c} T \\ (t=3,29) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=v,7) \end{array} \right\} \end{array}$$

It remains to combine the last three levels of braces to produce a modified and optimized loop structure. By associating the BEGIN braces or END braces, the following diagram demonstrates the new combined hierarchy.

$$\text{New Routine} \left\{ \begin{array}{c} V \\ (v=1,7) \end{array} \right\} \left\{ \begin{array}{c} S \\ (s=1,29) \end{array} \right\} \left\{ \begin{array}{c} T \\ (t=1,30) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=1,7) \end{array} \right\} \\ \left\{ \begin{array}{c} T \\ (t=3,29) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=v,7) \end{array} \right\} \\ V \text{ End} \\ (1) \left\{ \begin{array}{c} s=30 \\ T \\ (t=1,30) \end{array} \right\} \left\{ \begin{array}{c} K \\ (k=1,7) \end{array} \right\} \end{array}$$

Continuing this process throughout the remaining levels of hierarchies and combining two separate loop structures produces a modified loop structured as illustrated below:



The final routine has more codes and more calls to process, but the total number of loops in the final routine is 44,100. There are 21,924 loops that would not be performed .

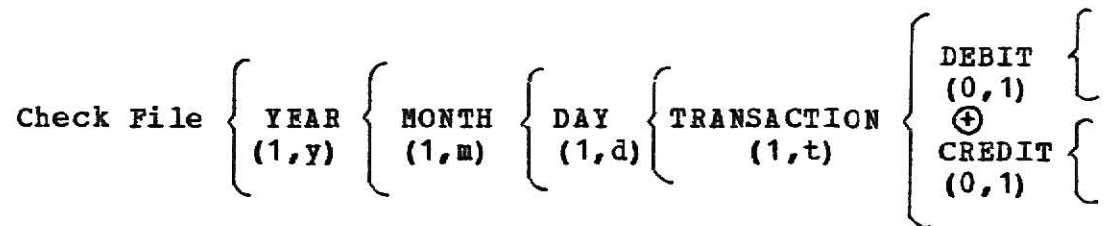
This demonstrates the usefulness of the commutative property of the Warnier-Orr Diagrams in combining hierarchies. It applies to a large class of loop structures,

logic structures, and data structures.

2.3 Semantics of the brace, "{"

Braces [4] in Warnier-Orr Diagrams are called "universals", a term borrowed from mathematical set theory. One level of the structure corresponds to one brace and the levels are counted left to right.

Each brace may represent a segment of processing, a subroutine, a procedure, or a do-loop.



The diagram shown above is for the checkbook balance report. In this diagram, it shows the conceptual relationships of one part of the structure to another. The report is organized by year; within year by months; within each month by days; and within each day by transactions, which are either debits (checks) or credits (deposits).

Year, month, day and transactions all appear in the report at least once and possible many times. The YEAR, MONTH, DAY, TRANSACTIONS, DEBIT and CREDIT can be treated as subroutines, procedures, or inner loops, depending on what language is used or how the programmers code the program. For instance, people using BASIC may treat them as subroutines. Programmers using another language may make MONTH an inner loop within YEAR, DAY an inner loop of MONTH, TRANSACTION an inner loop of DAY and may use IF-THEN-ELSE to take care of DEBIT and CREDIT within TRANSACTION.

Although the idea of the universal brace is fairly clear, there may be a variety of interpretations in any particular case. Before we can define transformations on these diagrams, we must specify what we mean by the universal brace. We will call this specification the semantics of the brace.

The brace, {, accompanies its hierarchical name and its right side process, for example:

$$X \left\{ \begin{array}{l} \text{begin } X : A \\ Y \left\{ \begin{array}{l} \\ \end{array} \right. \\ \text{end } X : B \end{array} \right.$$

Which we will interpret as:

```
begin X : A
while in-Y do
  Y
end-while
end X : B
```

"A" has to initiate boolean value of in-Y. "Y" has the responsibility to set boolean value of in-Y to false. "B" is responsible for setting boolean value of in-X to false.

This is one possibility for the semantics of the brace. For example, we could have used a do-loop structure for the semantics of the brace. The reason of choosing this semantics is that it is more general. For instance, we don't

have to execute the brace for a fixed times. We set a flag and when the flag changes to false by the program, the execution is terminated.

This example shows the most general form of the brace. It includes a "begin" and an "end" segment. Although in some cases these segments may be null, it is important to consider the most general case with both a begin segment and an end segment. When a segment is null, it can be omitted. This will simplify the transformations.

CHAPTER 3

SPECIFIC TRANSFORMATIONS

3.1 Motivation

The Warnier-Orr Diagrams possess the mathematical properties of associativity and commutativity. Using these two properties, the designer can modify the design structures and compare the different versions of the structures for the same problem.

The purpose of giving this specific transformation example is to give another view of using associative property and commutative property in Warnier-Orr Diagrams and comparing the different design structures that can give the same result.

3.2 Example

3.2.1. Transform Three-Level Diagram into One-Level Diagram

The Warnier-Orr Diagram of figure 1 is a diagram for the problem to sum each row and each column of a two dimensional array, which consists of three levels of nested do-loops. It is desired to transform this three-level diagram into a two-level diagram, then from two-level diagram into one-level diagram.

A necessary condition for these transformation to be correct is that the lowest level process is order-independent. In the example below the activities of the brace ELEMENT do not depend on which values of the array have already been processed.

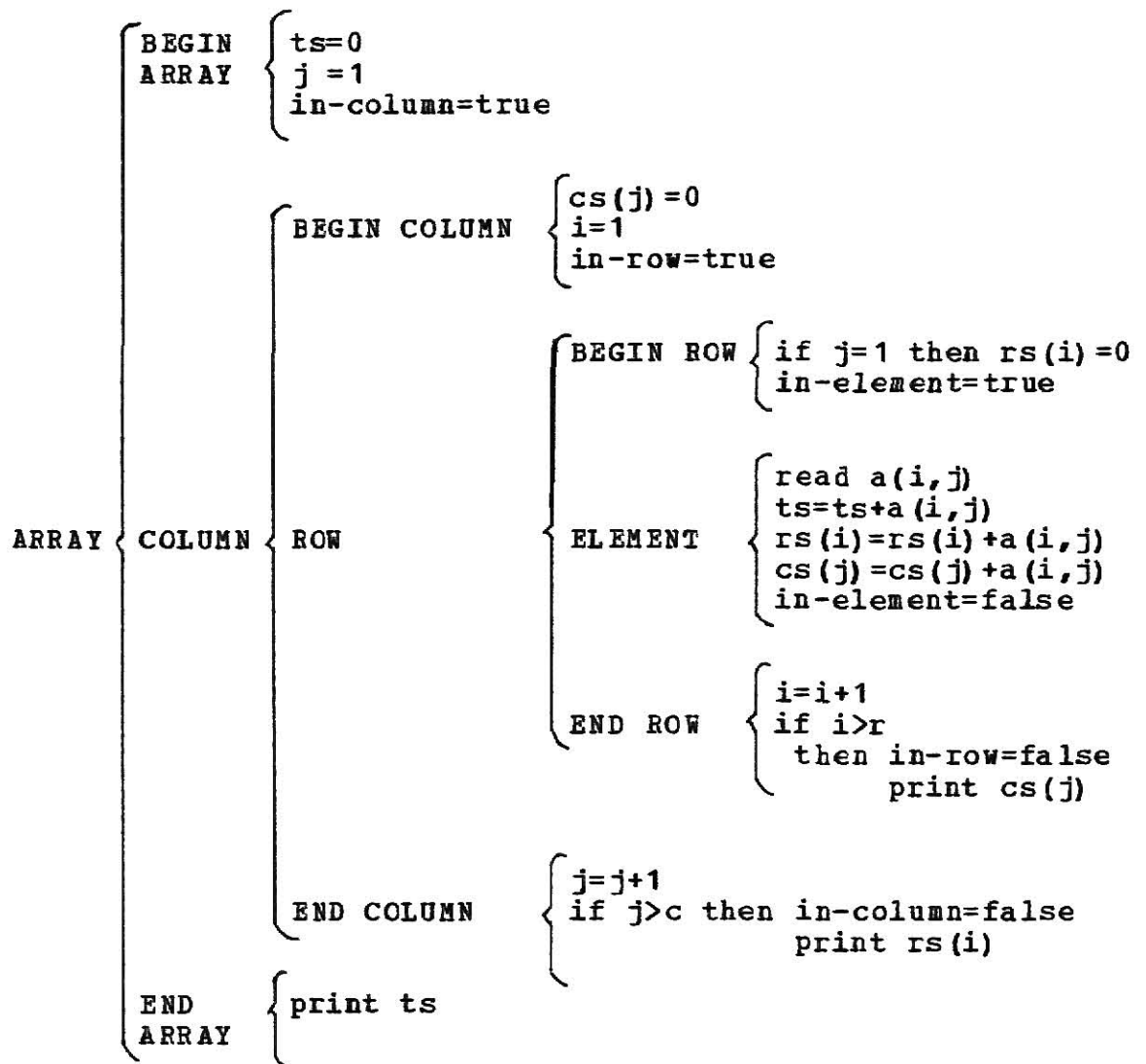


FIGURE-1: Three-level example diagram of Warnier-Orr diagram

By applying the associative property, the ROW level and the ELEMENT level could be associated as one level called ROW-ELEMENT level. Before making this transformation, two procedures must be done: First step, after the last statement of the processes of ELEMENT (i.e., in-element=true) the condition of "if not in-element" is added, then followed by the processes of "END ROW". Second step: the process of "BEGIN ROW" will be added under the last statement (i.e., in-row=true) of "BEGIN COLUMN" section. The segment which is the condition of " if in-row & not in-element" followed by the processes of "BEGIN ROW" is attached to the last statement of the first step has been done. This is accomplished in the diagram of figure 2.

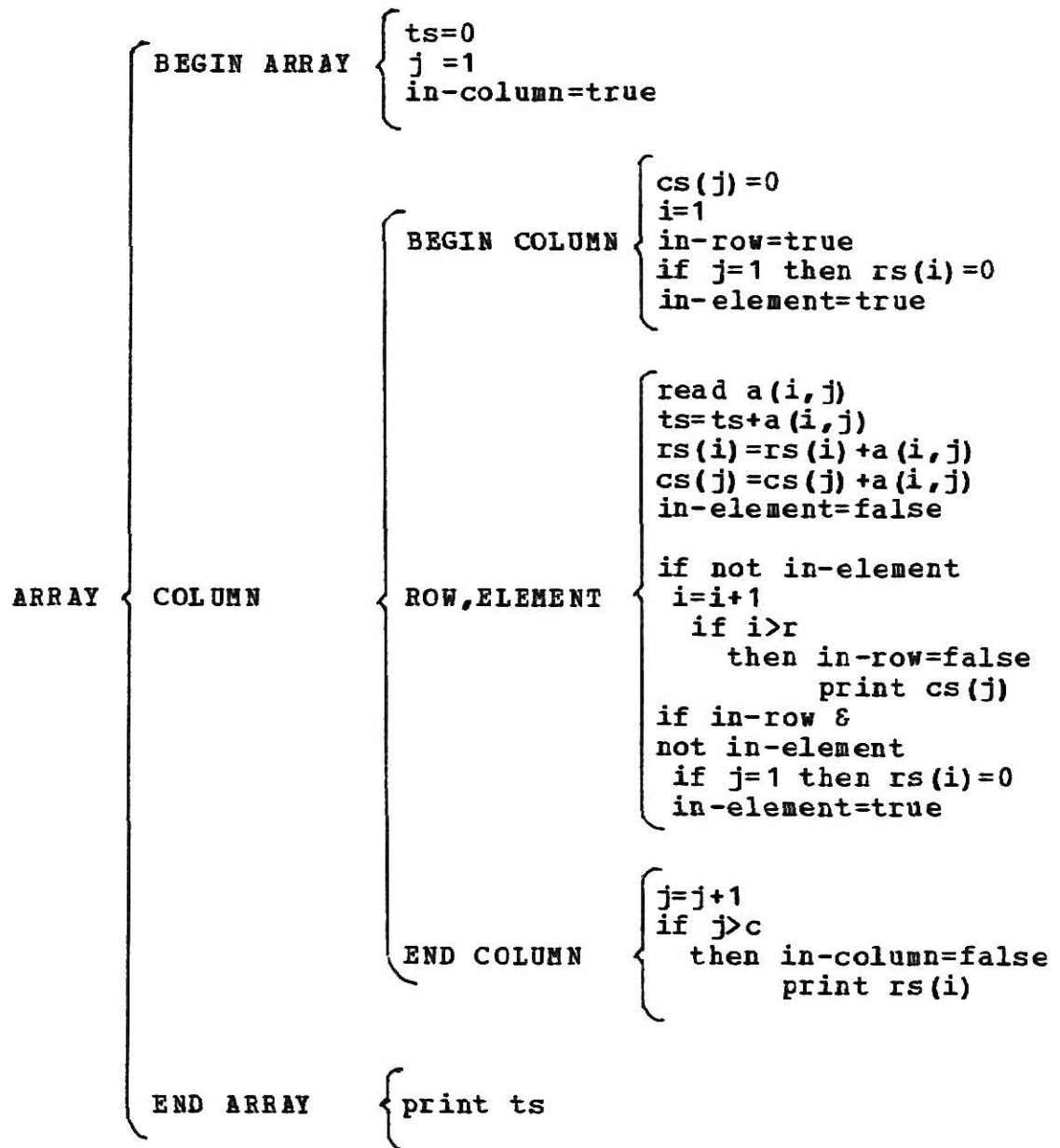


FIGURE-2: Transformed diagram of figure 1

The reason of adding those "if conditions" is for increasing the index of the loops and reinitializing the value of each column and each row. The number of the index will determine the termination of the loop. The reinitialization will make sure the sum of each column and each row is computed.

Figure 2 shows a two-level structure which is derived from figure 1 by applying the associative property of Warnier-Orr Diagram. Continuing the process by associating COLUMN level and ROW-ELEMENT level in figure 2 produces a new level named COLUMN-ROW-ELEMENT level. Following the last statement (i.e., in-element=true) in section of ROW-ELEMENT, add the condition of "if not in-row". After "if not in-row" append all statements in the section of END COLUMN. Then following the condition of "if in-column & not in-row", add all statements in the section of BEGIN COLUMN. Thus forms a new segment of processing in the new level of COLUMN-ROW-ELEMENT. Also append the part of BEGIN COLUMN to BEGIN ARRAY, form a new section, but still named BEGIN ARRAY. The new diagram is shown in figure 3.

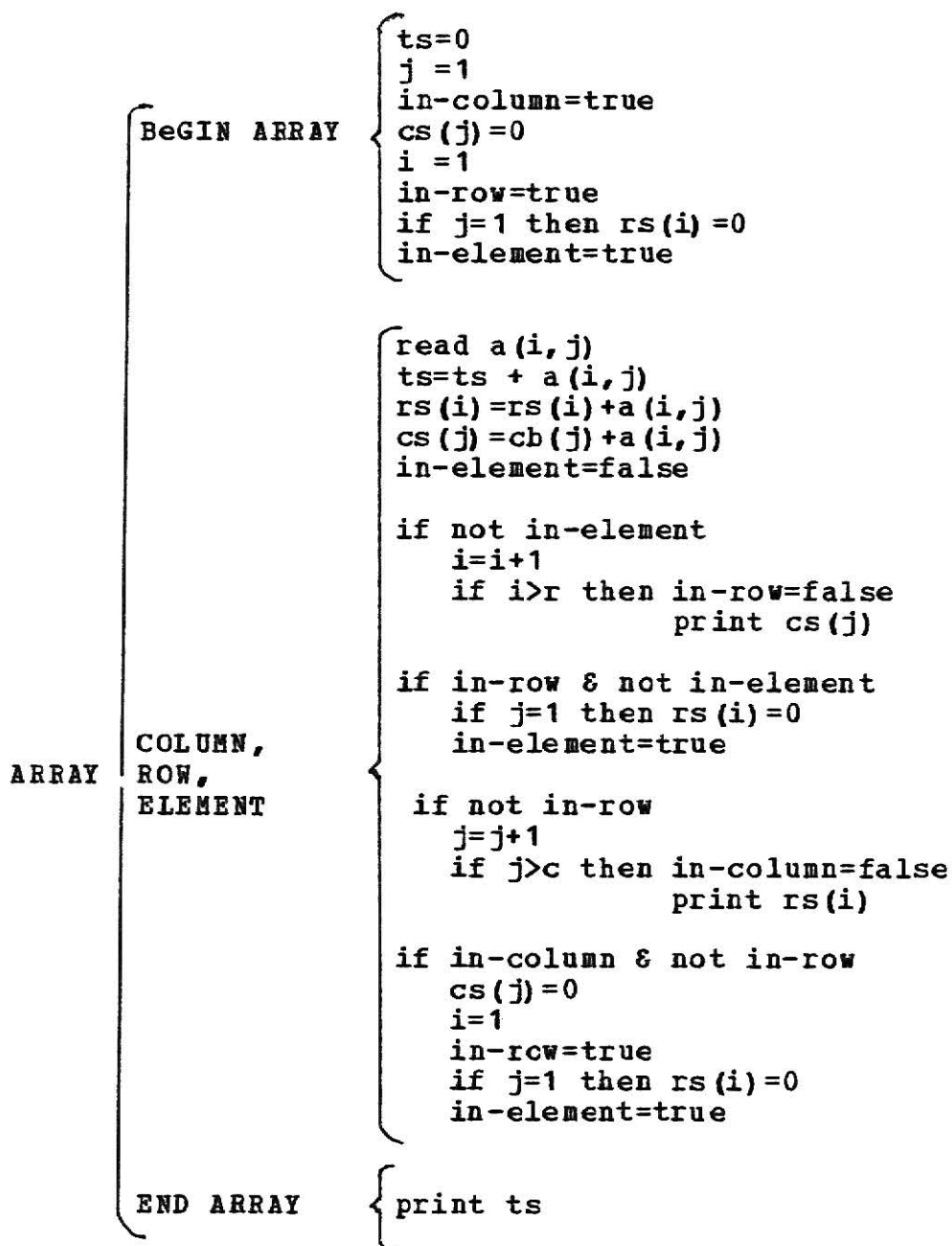


FIGURE-3: Transformed diagram of figure 2

3.2.2. Decompose one-level diagram into more levels diagram

Figure 3 is a one-level diagram. It is possible to decompose this flat structure diagram into a diagram with more levels.

For decomposing the design diagram showing in figure 3, it is necessary to decompose the hierarchical name into two names. That is changing "COLUMN-ROW-ELEMENT" to "ROW" and "COLUMN-ELEMENT". The new structure will be a two-level diagram: ROW level and COLUMN-ELEMENT level. All the statements in section BEGIN ARRAY will stay the same. Move the statement of "if not in-row" and all statements after it from COLUMN-ROW-ELEMENT level to the new section END ROW. The rest of the statements in COLUMN-ROW-ELEMENT will not be changed. The new two-level design diagram is showing in figure 4.

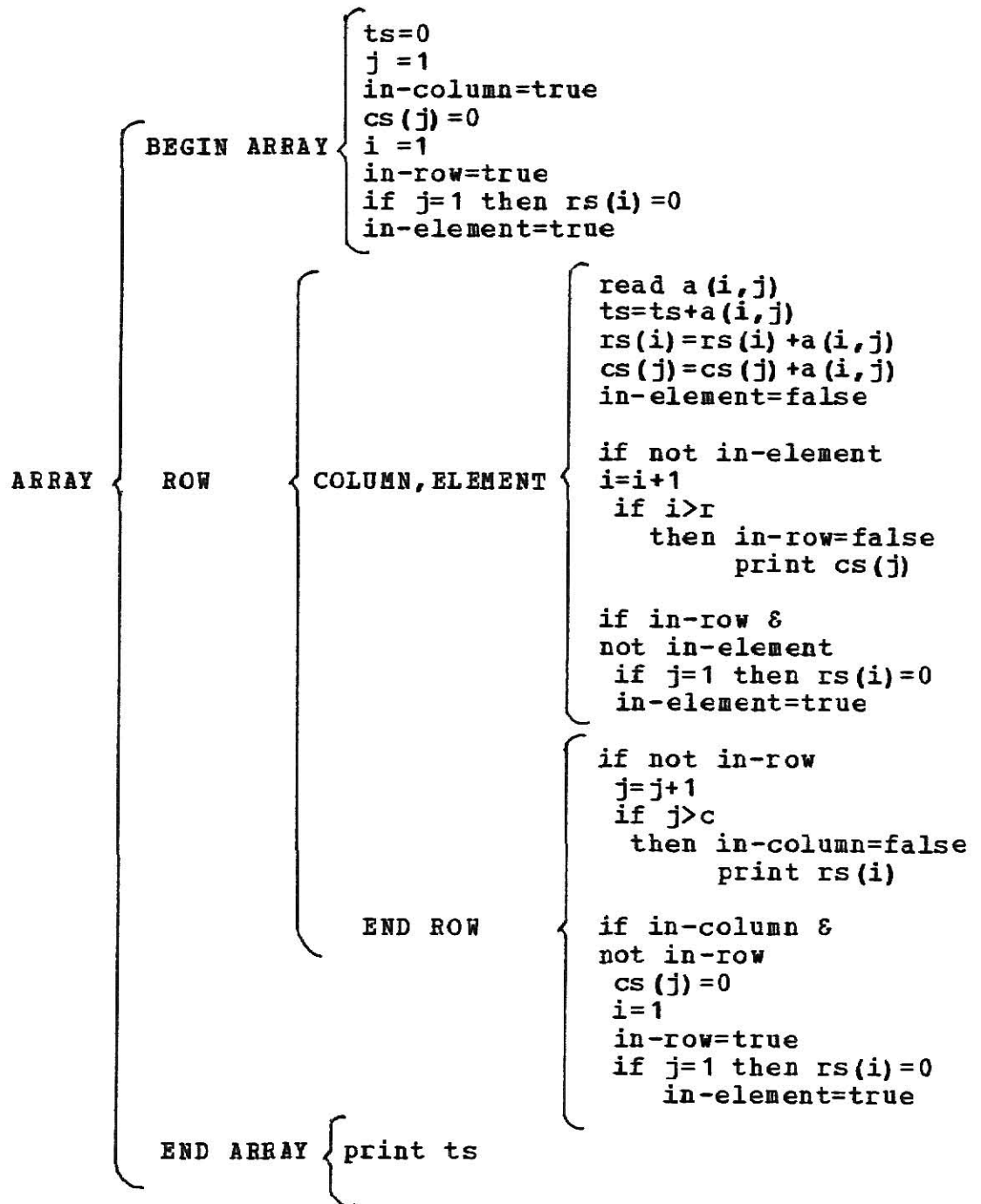


FIGURE-4: Transformed diagram of figure 3

The level of COLUMN-ELEMENT still can be decomposed. First, break down COLUMN-ELEMENT into COLUMN and ELEMENT. Place COLUMN on the left side of ELEMENT, or place ELEMENT on the left side of COLUMN, either way will work out the same result. Then remove the statement of "if not in-element" and all the statements after this statement from section COLUMN-ELEMENT to the new section END ELEMENT or END COLUMN, (if ELEMENT level is on the left side of COLUMN level, then the new section is END ELEMENT, otherwise, END COLUMN). The rest of the statements in COLUMN-ELEMENT will not be changed. Figure 5 shows the new diagram.

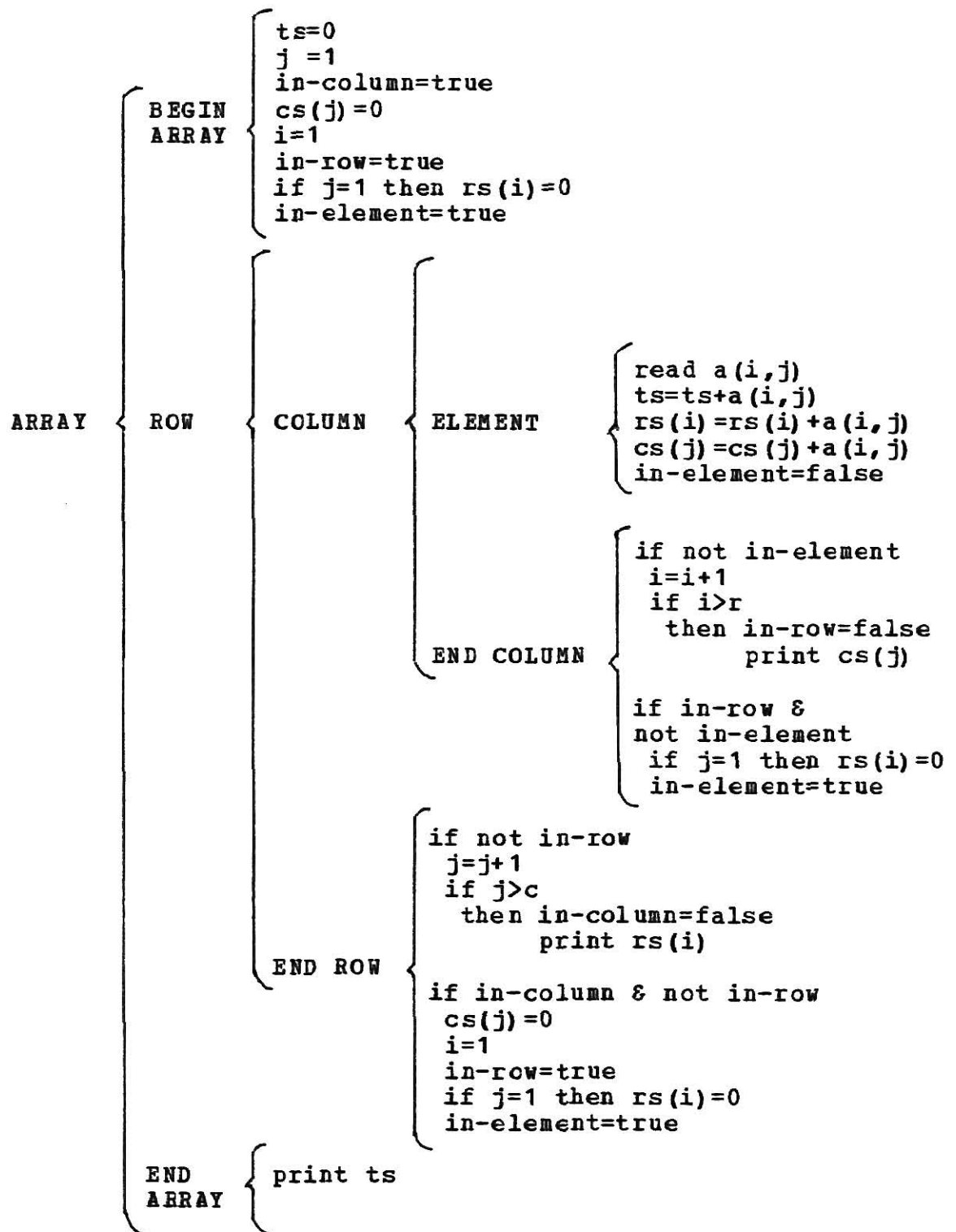


FIGURE-5: Transformed diagram of figure 4

3.3 Transformation Rules : Representation and Correctness

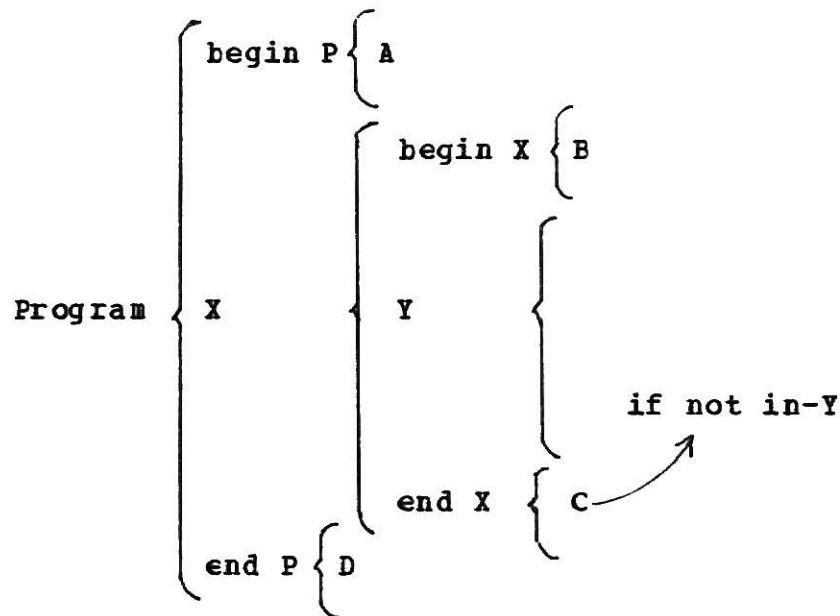
Basically, a transformation of a diagram is the generation of a new diagram from a given one. It is said to be correct if both diagrams are semantically equivalent. A transformation thus is a mapping between set of diagrams. In general, such a mapping is a partial one, as it is only defined for particular kinds of diagrams [6].

3.3.1. Rules for Combining the two right most levels.

Each time, when combine two levels into one level, two conditions are always added. And the BEGIN part and END part are rearranged. Three steps are needed to achieve the combination.

(1) END Transform

The END section of the outer level will be transferred and placed under the added "if condition" of the inner level. The form is denoted below:



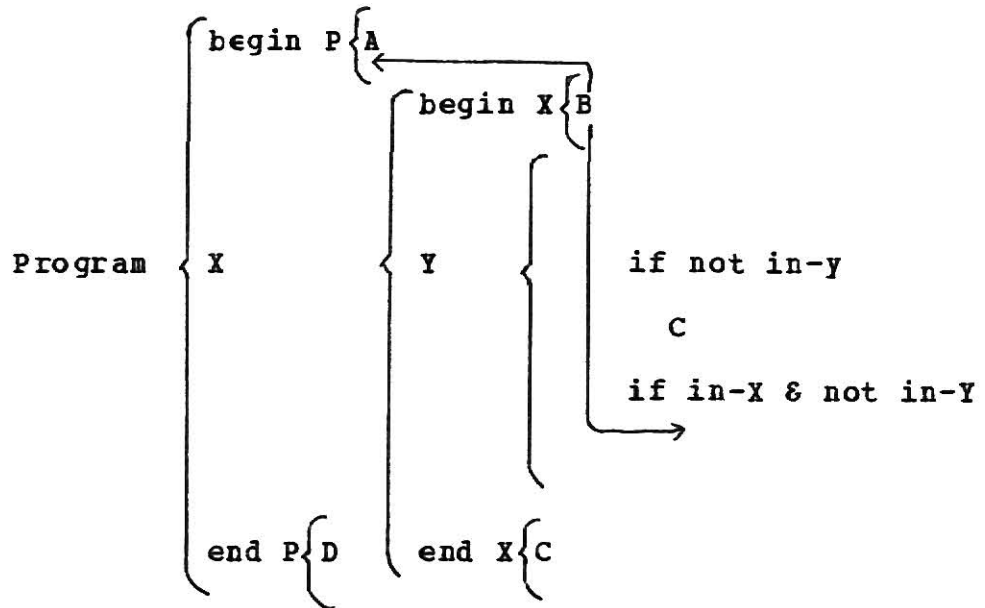
Where "if not in-Y" is the "if condition". "C" is the END section of "X".

When the process Y is finished there are some activities that need to be done. For example, after the status of Y level turns to false then the index-X is needed to be increased, the status of X level is also needed to be checked. Part C (i.e., END X) does these jobs. These activities could be moved into the inner bracket if they are preceded by a condition (if not in-Y). In this position the condition will be checked on each iteration but "C" will only be done when the "Y" is completed.

(2) BEGIN Transform

The BEGIN section of the inner level will be removed

and added to the BEGIN section of the outer level and to the processing segment of the lower level next to inner level. The form is denoted below:

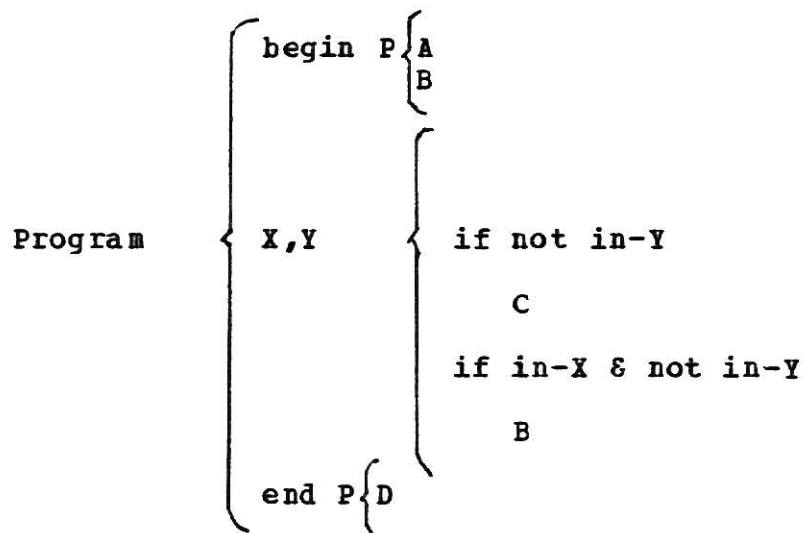


There are some initiations are done in "begin" section. If two adjacent levels are combined into one level then the initiations have to be done before the new level has been executed. Hence, "begin X" (i.e., part B) is moved into the begin section of outer bracket (i.e., part A). Also if the status of outer level is still true but inner level is false, it is necessary to reinitialize the inner level. In this situation the condition of "if in-X & not in-Y" followed by B (begin X) is appended to the process of inner level Y. On each iteration the condition will be checked:

but only when the inner level is finished but the outer level is not finished, will the proper processes be done.

(3) COMPRESSION

The last step is compression. Compress the two adjacent levels into one level. Combine two level names into one name. The new transform is shown below:

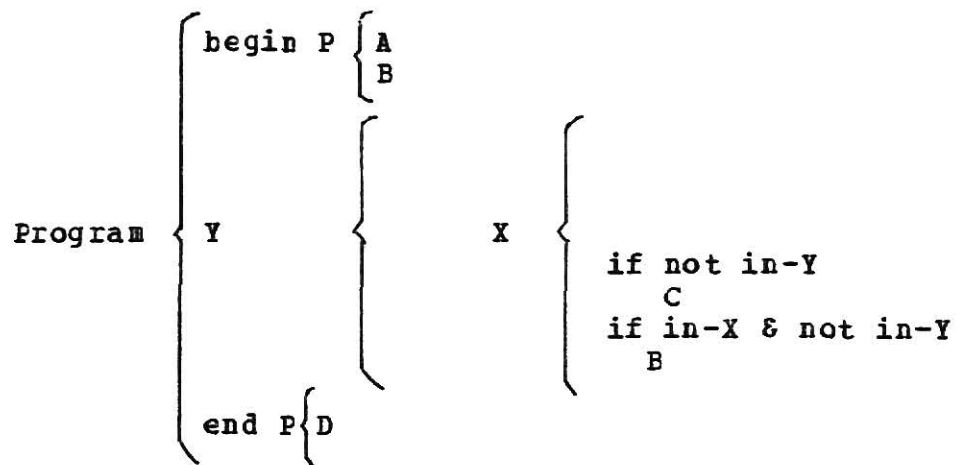


The processes of "X,Y" will not be finished until both in-X and in-Y are false.

3.3.2. Rules for Decomposing one level diagram into diagram with several levels.

(1) Decomposition

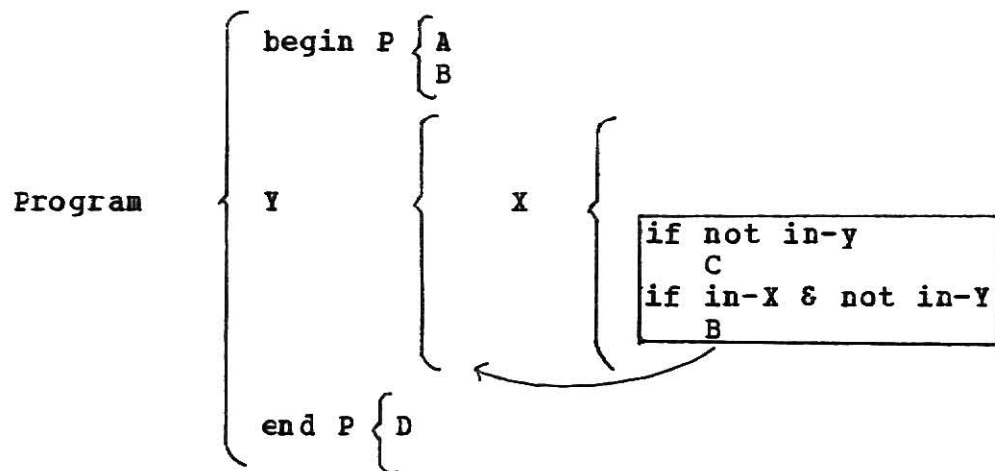
Decompose the level (using the previous diagram) into two levels.



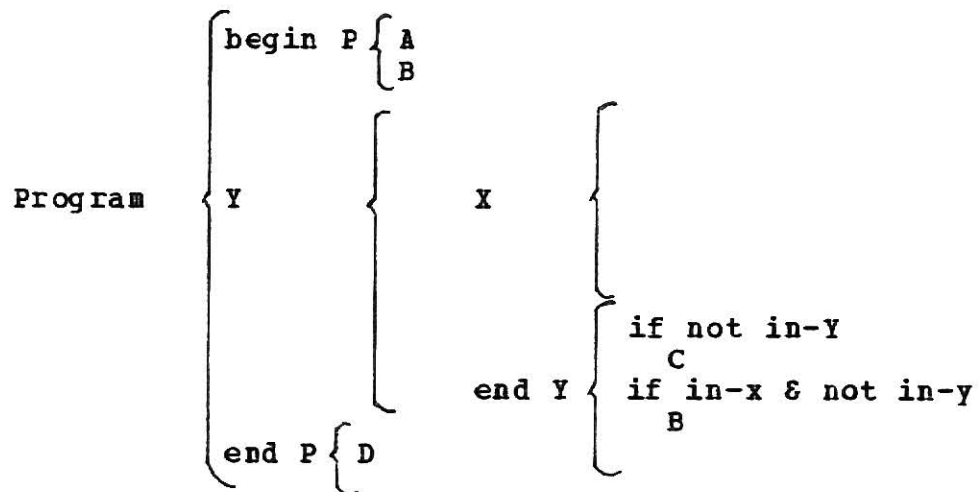
In this diagram the hierarchical name "X,Y" is decomposed into two: X and Y. For demonstrating the commutative property of Warnier-Orr design diagram, Y is going to be outer level, X is going to be inner level.

(2) END reverse Transform

Move the last two additions (i.e., the if conditions) to the END part of the outer level.



(3) BEGIN part is not changed. Form a new diagram.



Since the sequence of the hierarchical levels is changed, the begin section which does the initiations for all the levels will stay the same. This will make the transformation rules easier to follow.

The processes of X level and Y level will not be finished untill in-X and in-y both are false.

CHAPTER 4

INFORMAL PROOF OF THE SPECIFIC TRANSFORMATIONS

In the previous chapter, five different diagrams of the specific example were presented. The rules of the transformation were also illustrated. The transformation is said to be correct if these diagrams can solve the problem and compute the same result.

Figure 6 shows the simplified diagram of figure 1.

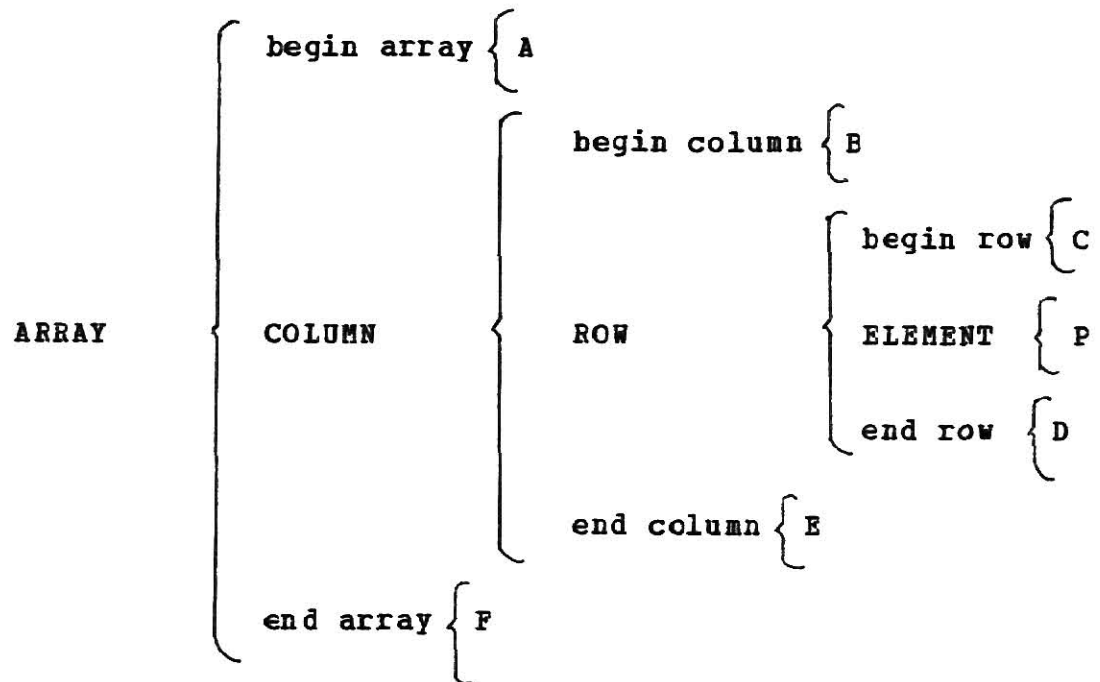


FIGURE-6:Simplified Diagram of Figure 1

The execution order in figure 6 is: A B C P D C P D C P
 D-----C P D C P D E B C P D C P D-----C P D E B C P D C P
 D-----C P D E F.

In chapter 3, figure 1 was transformed to figure 2 by using associative property of Warnier-Orr diagram. The following figure is the simplified diagram of figure 2.

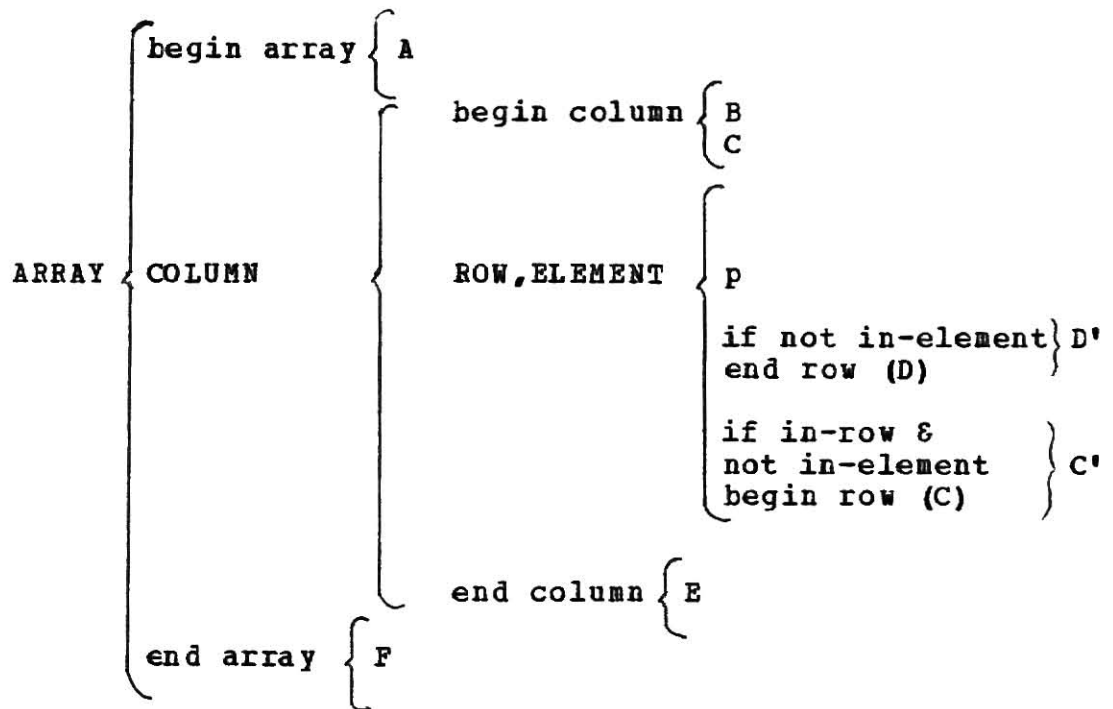


FIGURE-7: Simplified diagram of Figure 2

The primes are used to denote the conditional statement and the operation. "D'" is the segment of if condition (if

not in-element) followed by end row(D). "C'" is the segment of if condition (if in-row & not in-element) followed by begin row(C).

The execution sequence of this diagram is: A B C P D' C' P D' C'----- P D' C' P D' C' E B C F D' C' P D' C'-----P D' C' P D' C' E B C P D' C' P D' C'-- ---P D' C' P D' C' E F.

But when the if condition isn't met, then the following process will not be executed. For example, when in-row turns to false, then the condition of "if in-row & not in-element" can't be met, and "begin column(c)" will be skipped. Hence, the actual execution sequence is : A B C P D' C' P D' C'----- P D' C' P D' E B C P D' C' P D' C' ----- P D' C' P D' E B C P D' C' P D' C' -----P D' C' P D' E F. This shows the same sequence as in figure 6.

In chapter 3, figure 2 was transformed to figure 3 by using associative property of Warnier-Orr diagram. The following figure is the simplified diagram of figure 3.

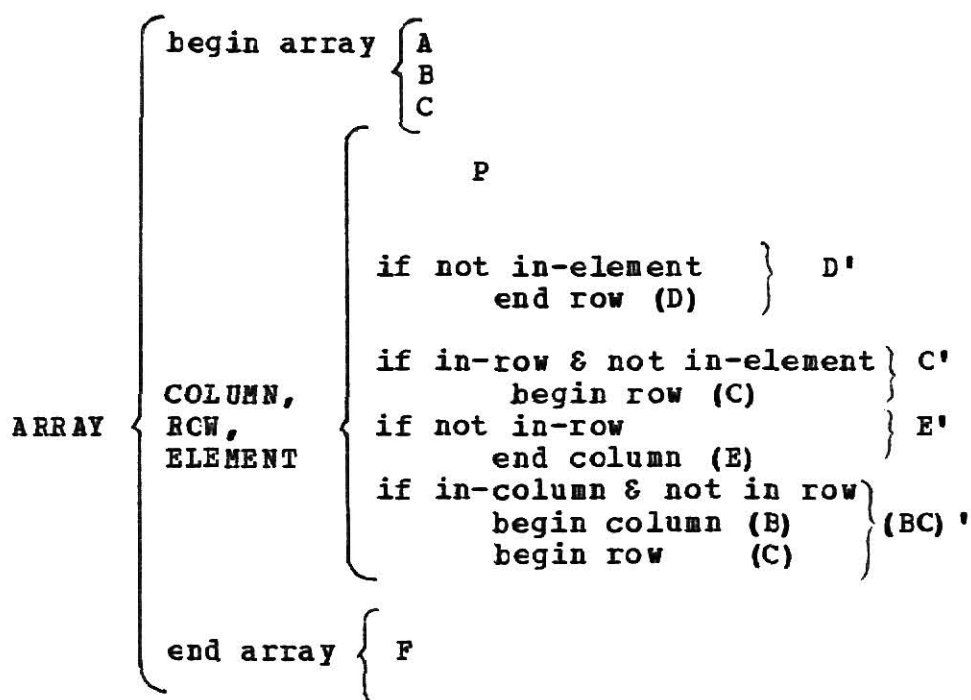


FIGURE-8: Simplified Diagram of Figure 3

The primes are used to denote the conditional statement and the operation. "D'" is the segment of if condition (if not in-element) followed by end row(D). "C'" is the segment of if condition (if in-row & not in-element) followed by begin row(C). "E'" is the segment of if condition (if not in-row) followed by end column(E). "(BC)'" is the segment of if condition (if in-column & not in-row) followed by begin column(B), begin row(C).

The execution sequence of the diagram is: A B C P D' C' E' (BC)' P D' C' E' (BC)' ----- P D' C' E' (BC)' F. But

when the if condition isn't met, then the process following that if condition is skipped. The actual execution order in figure 8 is : A B C P D' C' P D' C' ----- P D' C' P D' E' (BC)' P D' C'-----P D' C' P D' E' (BC)' P D' C'----- P D' C' P D' E' F. This shows the same order as the order in figure 6.

In chapter 3, figure 3 was transformed to figure 4 by using associative property of Warnier-Orr diagram. The following diagram is the simplified diagram of figure 4.

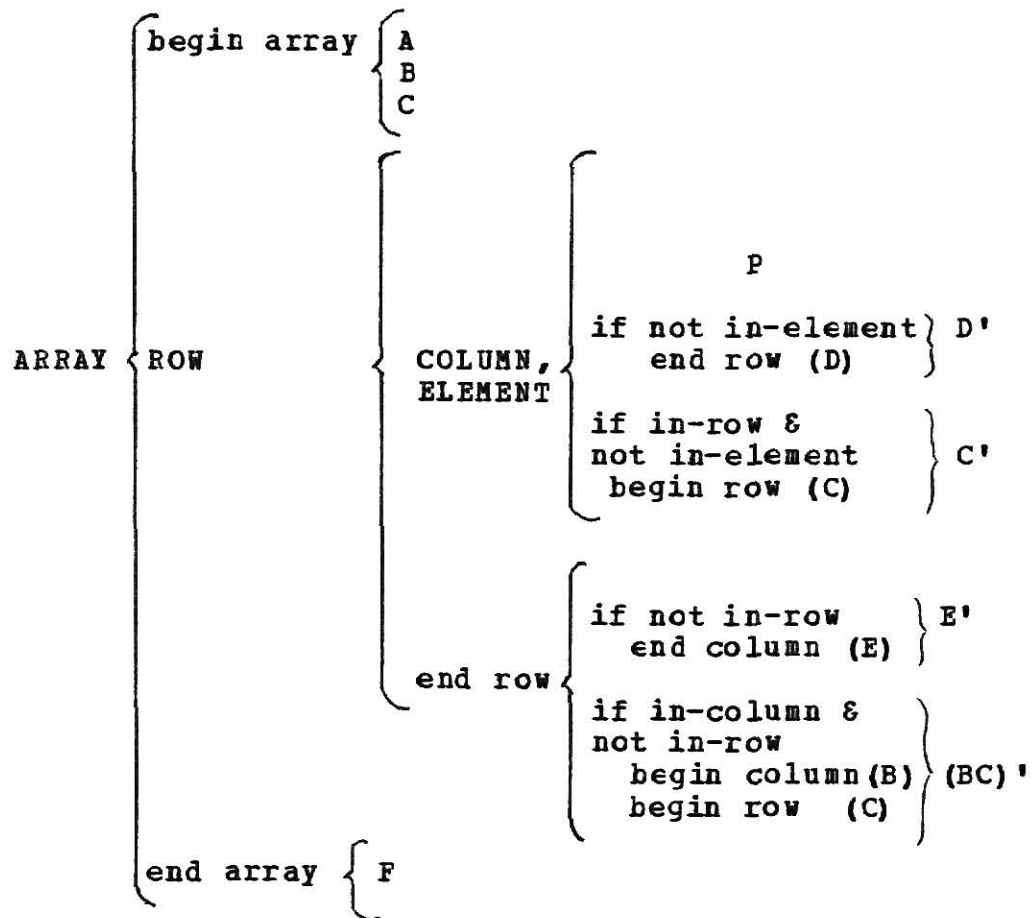


FIGURE-9: Simplified Diagram of Figure 4

The D', C', E' and (BC)' represent the same segment showing in figure 8. The execution order of figure 9 is A B C P D' C' P D' C' ----- P D' C' P D' C' E' (BC)' P D' C' ----- P D' C' P D' C' E' (BC)' P D' C' ----- P D' C' P D' C' P D' C' E' (BC)' F. In the last cycle, the if condition in C' and another if condition in (BC)' are not met, in this case the processes following these two "if condition" are skipped. The actual sequence of this diagram

is: A B C P D' C' P D' C'-----P D' C' P D' C' E' (BC)' P D'
 C'-----P D' C' P D' C' E' (BC)' P D' C'----- P D' C' P
 D' C' P D' E' F. This shows the same order as the order in
 figure 6.

In chapter 3, figure 4 was transformed to figure 5 by
 using associative property of Warnier-Orr diagram. The
 following figure is the simplified diagram of figure 5.

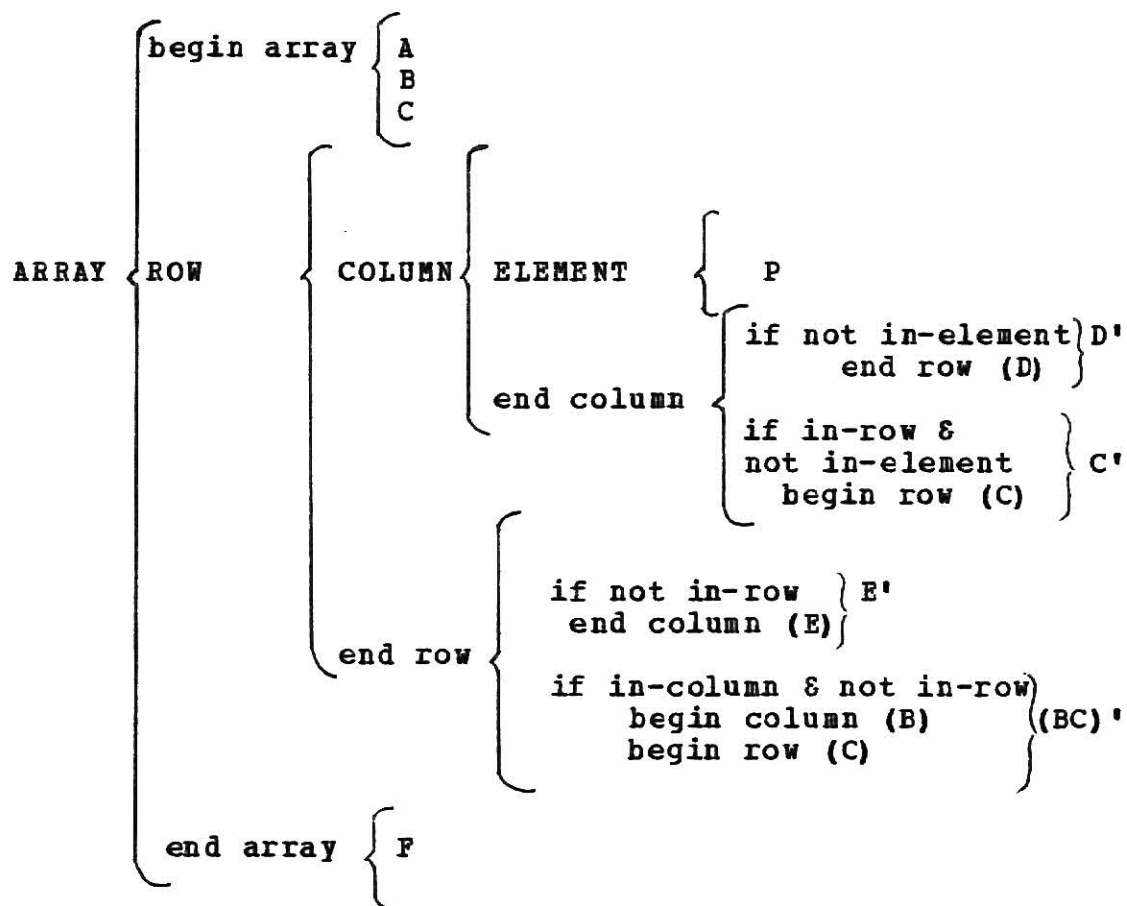


FIGURE-10: Simplified Diagram of Figure 5

In figure 10, the execution sequence is exactly the same as in figure 9. The actual execution sequence of this diagram is: A B C P D' C' P D' C'----- P D' C' P D' C' P D' E' (BC)' P D' C'-----P D' C' P D' E' (BC)' P D' C'----- P D' C' P D' E' F.

Comparing the sequence of the execution of these five Warnier-Orr design diagrams, we can reach a conclusion: Although there are some different conditions added in different sections in each of five diagrams, these sequence of executions is the same in each of the cases. That is, BEGIN ARRAY is always executed before BEGIN COLUMN, BEGIN ROW is executed after BEGIN COLUMN, then section ELEMENT is following BEGIN ROW, then followed by END ROW, END COLUMN and END ARRAY sequentially.

The following picture may represent more clearly:

```
BEGIN ARRAY  -->  BEGIN COLUMN  -->  BEGIN ROW  -->  
ELEMENT -->  END ROW  -->  END COLUMN  -->  END ARRAY
```

Each time when an "BEGIN " is encountered, ROW or COLUMN is reinitiated, this will guarantee that the sum of each row and each column will be computed.

CHAPTER 5

CONCLUSION

The major advantages of the Warnier-orr design methodology are its diagrams and the transformations. The diagrams make Warnier-Orr designs easy to read and understand. The transformation properties allow the designers to modify loop structures, logic structures and data structures.

The transformation design diagrams allow designers to do the comparisons of alternative designs. From the alternative designs, the designers may choose one of the designs which can achieve the requirement of a good design: low cost, reliable, easy to understand and easy to maintain.

In this report the transformation rules of hierarchical structures in Warnier-Orr diagrams have been developed. The rules are for combining two levels into one level, and for decomposing one level into two levels.

Based on this study, two areas for further research are proposed as follows:

(1) . Applying those rules to the other programs. In this report, the rules were applied to a small program that computed the sum of each row and each column of an two dimensional array. Since these rules are for transforming the iterative structures, it will be nice to see these rules applied to some other programs.

(2) . Actual execution of alternative designs. The alternative designs should be converted into computer programs. These programs should be executed on a variety of data. This will verify that the alternatives are actually equivalent.

After expanding these areas, the transformation rules should be more applicable.

SELECTED BIBLIOGRAPHY

- [1]. Barry W. Boehm, "Software Engineering". IEEE Transactions on Computers, December 1976.
- [2]. M. A. Jackson, "Principles of Program Design" New York:Academic, 1975.
- [3]. David A. Higgins, "Structured Programming with Warnier-Orr Diagrams", Byte Publications, Inc., Peterborough, New Hampshire, 1978.
- [4]. David A. Higgins, "Warnier-Orr Diagrams" in "Auerbach Computer Programming Management", Auerbach Publishers Inc.
- [5]. Peter A. Verdegrall, "The Warnier-Orr Diagrams". Proceedings of Comp Con. IEEE Computer Society, Spring 1979.
- [6]. F. L. Bauer, M. Broy, H. Partsch, P. Pepper, H. Wossner, "Systematics of Transformation Rules", in "Program Construction ", edited by F.L.Bauer, Springer-Verlag, New York Inc.

APPENDIX A

pseudo-code for figure 1

```
procedure array ;
    ts = 0
    j = 1
    in-column = true

    while in-column do
        call column
    end-dowhile

    print ts

procedure column;
    cs(j) = 0
    i = 1
    in-row = true

    while in-row do
        call row
    end-dowhile

    j = j+1
    if j>c then do
        in-column = false
        print rs(i)
    od

procedure row;
    if j=1 then rs(i)=0
    in-element = true

    while in-element do
        call element
    end-dowhile

    i = i+1
    if i>r then do
        in-row = false
        print cs(j)
    od
```

```
procedure element;  
  read a(i,j)  
  ts = ts+a(i,j)  
  rs(i) = rs(i) + a(i,j)  
  cs(j) = cs(j) + a(i,j)  
  in-element = false
```

pseudo-code for figure 2

```

procedure array;
    ts = 0
    j = 1
    in-column = true

    while in-column do
        call column
    end-dowhile

    print ts

procedure column;
    cs(j) = 0
    i = 1
    in-row = true

    if j=1 then rs(i)=0
    in-element = true

    while in-row & in-element do
        call row-element
    end-dowhile

    j = j+1
    if j>c then do in-column=false
                    print rs(i)
                od

procedure row-element;
    read a(i,j)
    ts = ts + a(i,j)
    rs(i) = rs(i) + a(i,j)
    cs(j) = cs(j) + a(i,j)
    in-element = false

    if in-element = false
    then i = i+1
    if i>r then do in-row = false
                    print cs(j)
                od

    if in-row & not in-element
    then if j=1 then do rs(i)=0
                        in-element=true
                    od

```

pseudo-code for figure 3

```

procedure array;
  ts = 0
  j = 1
  in-column = true

  cs(j) = 0
  i = 1
  in-row = true

  if j = 1 then rs(i) = 0
  in-element = true

  while in-column & in-row & in-element do
    call column-row-element
  end-dowhile

  print ts

procedure column-row-element;
  read a(i,j)
  ts = ts + a(i,j)
  rs(i) = rs(i) + a(i,j)
  cs(j) = cs(j) + a(i,j)
  in-element = false

  if not in-element
    then i = i+1
      if i>r then in-row = false
        print cs(j)

  if in-row & not in-element
    then if j=1 then rs(i)=0
      in-element = true

  if not in-row
    then j = j+1
      if j>c then in-column=false
        print rs(i)

  if in-column & not in-row
    then cs(j) = 0
      i = 1
      in-row = true

      if j=1 then rs(i) = 0
      in-element = true

```

pseudo-code for figure 4

```

procedure array;

    ts = 0
    j = 1
    in-column = true

    cs(j) = 0
    i = 1
    in-row = true

    if j=1 then rs(i)=0
    in-element = true

    while in-column & in-row & in-element do
        call row
    end-while

    print ts

procedure row;

    call column-element

    if not in-row
    then do
        j = j+1
        if j>c then in-column=false
        print rs(i)
    od

    if in-column & not in-row
    then do
        cs(j) = 0
        i = 1
        in-row = true

        if j=1 then rs(i)=0
        in-element = true
    od

```

```
procedure column-element;

  read a(i,j)
  ts = ts+a(i,j)
  rs(i) = rs(i)+a(i,j)
  cs(j) = cs(j)+a(i,j)
  in-element = false

  if not in-element
    then do
      i = i+1
      if i>r then in-row=false
                print cs(j)
    od

  if in-row & not in-element
    then do
      if j=1 then rs(i)=0
        in-element = true
    od
```

pseudo-code for figure 5

```

procedure array;
  ts = 0
  j = 1
  in-column = true

  cs(j) = 0
  i = 1
  in-row = true

  if j = 1 then rs(i) = 0
  in-element = true

  while in-column & in-row & in-element do
    call row
  end-while

  print ts

procedure row;
  call column

  if not in-row then do
    j = j+1
    if j>c then in-column = false
    print rs(i)
  od

  if in-column & not in-row
  then do
    cs(j) = 0
    i = 1
    in-row = true
    if j=1 then rs(i)=0
    in-element=true
  od

```



```
procedure column;  
  call element  
  
  if not in-element  
  then do  
    i = i+1  
    if i>r then in-row=false  
              print cs(j)  
  od  
  
  if in-row & not in-element  
  then do  
    if j=1 then rs(i)=0  
    in-element=true  
  od  
  
procedure element;  
  read a(i,j)  
  ts = ts+a(i,j)  
  rs(i) = rs(i)+a(i,j)  
  cs(j) = cs(j)+a(i,j)  
  in-element=false
```

TRANSFORMATION OF ITERATION STRUCTURE IN
WARNIER-ORR DIAGRAMS: EXAMPLES AND RULES

by

SHU-MEI LIN

BACHELOR OF LAW, NATIONAL TAIWAN UNIVERSITY, 1969

AN ABSTRACT OF A MASTER'S REPORT

Submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1982

ABSTRACT

Most software errors are made during the design phase. A well-designed system not only is more likely to be correct but also is usually more understandable, modifiable, and easier to maintain. The Warnier-Orr diagrams are design tools that can help to achieve these purposes.

The Warnier-Orr diagrams possesses two mathematical properties, associativity and commutativity. Using these two properties, the designer can modify his design. This report formalizes the transformation of the Warnier-Orr design diagrams. Specific transformations of Warnier-Orr diagrams are introduced and informally proved. A sample program for summing each row and column of an array is presented to illustrate the notions of associativity and commutativity.