70-20,898

Grason, John, 1942-
METHODS FOR THE COMPUTER-IMPLEMENTED SOLUTION
OF A CLASS OF "FLOOR PLAN" DESIGN PROBLEMS.

Carnegie-Mellon University, Ph.D., 1970
Engineering, electrical

University Microfilms, A XEROX Company, Ann Arbor, Michigan

CARNEGIE INSTITUTE OF TECHNOLOGY

COLLEGE OF ENGINEERING AND SCIENCE

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy

TITLE Methods for the Computer-Implemented Solution of a Class of "Floor Plan" Design Problems

PRESENTED BY John Grason

ACCEPTED BY THE DEPARTMENT OF Electrical Engineering (Systems and Communication Sciences)

Herbert A Simon — MAJOR PROFESSOR — April 30, 1970 — DATE

A. G. Jordan — DEPARTMENT HEAD — May 4, 1970 — DATE

APPROVED BY THE COMMITTEE ON GRADUATE DEGREES

CHAIRMAN — 5/4/70 — DATE

ABSTRACT

The work presented is intended as a case study in computer-implemented design. Its purpose is to illustrate the relationship between the representation chosen for a design problem and the methods developed for solving that problem.

A formal class of "floor plan"-type design problems is defined. In these problems a set of rectangular rooms is specified, and an allowable list of dimensions is given for each room. In addition, a set of required adjacencies between rooms, or between a room and an outside wall of the building, is given. The problem is to produce a rectangular floor plan of a building that contains all of the specified rooms, and that satisfies all of the adjacency and dimension requirements.

A linear graph representation for floor plans is developed. This graph is the dual graph of the floor plan, itself treated as a linear graph. Thus, the nodes of the dual graph correspond to rooms, and the edges correspond to adjacencies between rooms. The edges are colored and directed to indicate the different types of adjacencies (north, south, east, or west), and weights are assigned to edges to indicate the length of floor plan wall segments. Rules of well-formedness are developed for the colored, directed graph.

Methods for producing solutions to the "floor plan" design problems are developed, based on the dual graph representation. These methods produce incomplete dual graphs as partial design solutions.

As an aid to handling these graphs, a planar graph grammar is introduced that allows one to keep track of the planarity of a graph, and to generate geometric realizations of any planar graph.

The design methods are implemented in a computer program, GRAMPA, written in IPL-V. Several illustrative problems solved by GRAMPA are discussed. It is concluded that, with proper modifications, GRAMPA could be used either as a research tool or as a practical design program.

ACKNOWLEDGEMENTS

I would like to thank Professor Herbert A. Simon for his thoughtful guidance throughout the research reported in this thesis. A special debt is owed to Professor Saul Amarel (Department of Computer Science, Rutgers University) for his inspiration and advice during the early stages of the research.

Thanks are due Professors C. Gordan Bell, Thomas W. Calvert, and Charles M. Eastman for their helpful comments on the preparation of this document. I am also grateful to Professor Eastman for his many discussions with me about the research.

I would like to thank Mrs. Evelyn L. Adams and Miss Cheryll Conaway for their help in typing the manuscript. My deepest gratitude and affection go to Jennifer, my wife, for both her moral support and her help in preparing the figures and the manuscript.

This work was partially supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-67-C-0058) and is monitored by the Air Force Office of Scientific Research. Portions of the work were also supported by National Institutes of Mental Health grant MH-07722, and by the Air Force Office of Scientific Research of the Office of Aerospace Research under contract AF49(638)-1184.

READING SUGGESTIONS

This thesis is rather long. One reason for this is that it contains a lot of what might be classified as documentation -- documentation of a computer program and documentation of examples run by the program. Another reason is that it contains several long theorem proofs and a good amount of explanation which shows that the program works as claimed.

For the convenience of the reader, most of the documentation is confined to Chapters 3 and 4 and Appendix A.2. The theorem proofs are confined to Appendix A.1. In addition, Chapter 3, which is long, is organized so that most of it can be skimmed by reading only the first few paragraphs of the major sections.

It is not necessary to read the entire thesis to capture its main ideas. The reader is encouraged to use the table of contents as a guide in selecting those sections that he wants to read carefully, keeping in mind what was said above about documentation and proofs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. BACKGROUND AND GENERAL DESCRIPTION

1.0. Introduction

This thesis describes an approach to the solution of a certain class of "floor plan" design problems. "Floor plan" is set off by quotation marks because the approach can be used also on other types of spatial design problems which resemble the floor plan problem. An example of such a problem is the layout of electronic integrated circuits. This has been formulated as a floor plan-type problem by the author in Ref. 13.

Three main topics are developed in the thesis:

(1) A design representation for floor plans, based on the use of linear graphs.

(2) Methods for the solution of floor plan design problems, based on this representation.

(3) A computer program (called GRAMPA, for GRaph Manipulating PAckage) that implements many of these methods.

The main discipline related to all three of these topics is design methodology. However, this is not intended to be primarily a thesis on the theory of design. The description of topics (1), (2), and (3) above provides more than enough material for a complete work, and this description constitutes the major portion of this thesis.

Nevertheless, the description makes clear the relevance of the topics covered to design methodology. This first chapter provides some background on the thesis and establishes a general theoretical framework for the work. It concludes with a brief outline of the remaining chapters of the thesis.

1.1. Choice of a Problem

The investigation described in this thesis grew out of the author's interest in computer-implemented design. This is to be distinguished from computer-aided design, in which the computer is used as an adjunct to the human designer, accomplishing such tasks as bookkeeping, graphical manipulation, or numerical analysis. In computer-implemented design, on the other hand, the computer itself makes all of the design decisions associated with some particular phase of the design process.

For the sake of objectivity, a specific class of design problems was chosen as a working example. Architectural design was chosen because it potentially incorporated both the technical and the artistic aspects necessary to provide some latitude to the investigation. Since architectural design is a rather large subject, the class of problems was further limited to floor plan layout. Floor plan design is comprehensible to designers from a wide range of disciplines, and thus it is a suitable medium for illustrating general concepts of computer-implemented design methodology.[1]

As it turned out, the technical aspects of the floor plan layout problem provided so much opportunity for study that the

[1] Generally comprehensible design examples are desirable if one is to attempt to deal with a "science of design" applicable simultaneously to all design disciplines. See Ref. 24, Chapter III, "The Science of Design".

artistic aspects were never really approached in this investigation. However, the work in this thesis could logically be extended to the artistic aspects of the problem.

1.2. What is Meant by "Design"[2]

It is possible to discuss what is meant by "design" at great length. However, for the purposes of this thesis, a statement by Herbert A. Simon, speaking about a 'science of design', provides an appropriate perspective:

> Historically and traditionally, it has been the task of the science disciplines to teach about natural things, how they are and how they work. It has been the task of engineering schools to teach about artificial things: how to make artifacts that have desired properties; how to design.
>
> Engineers are not the only professional designers. Everyone designs who devises courses of action aimed at changing existing situations into preferred ones.[3]

Thus, a short definition of design might be "how to make artifacts that have desired properties".

From an operational point of view, the design process has three principal phases: specification, structuring, and synthesis. These phases may occur in any sequence, or even concurrently. How they are combined -- either iteratively, recursively, serially, or in parallel -- determines the design strategy. In the

[2] For a more complete treatment of the subject matter in this and the following two sections, see the author's working paper, Ref. 14.

[3] Ref. 24, Chapter III, "The Science of Design".

specification phase the design requirements, or goals, are identified and listed. In the synthesis phase an artifact is created which satisfies these design requirements. The structuring phase is necessary because most interesting design problems are too large for a designer to pay attention simultaneously to the resolution of all design requirements. Therefore, it is necessary to fragment the problem into smaller subproblems which, taken individually, are within the problem solving capabilities of the designer. This fragmentation can be called the structuring of the design problem. More will be said about this phase of the process in the next section.

1.3. Emphasis of the Investigation

Of the three phases just mentioned, the specification phase and the structuring phase lend themselves most readily to computer-aided design.[4] However, it is difficult to justify calling something computer-implemented design unless the synthesis phase is automated. This is the phase which receives the prime attention in this thesis.

The investigation was originally patterned to explore the aspects of a paradigm for a computer-implemented design program (explained in the next section). However, early in the work a very advantageous design representation for floor plan design, based

[4] See, for instance, Refs. 1, 3, 4, 5, 12, and 20.

on planar graph theory, was produced. A great deal of effort was devoted to refining this representation so that it would be useful for the design process. Next, a set of methods, based on this representation, was developed for designing floor plans. Finally, a computer program, GRAMPA, was developed to implement these methods. This implementation served two purposes:

(1) It served as a laboratory for the development of the floor plan design methods.

(2) It provided a working example for the study of computer-implemented design.

The first of these two uses for GRAMPA was given the most study. The second was postponed mostly for future work, and so the design program paradigm was not explored very thoroughly. However, it did serve as a basis for GRAMPA, and so it is described in the next section.

The version of GRAMPA described in this thesis is not really a practical design tool, although it might be made into one, as discussed in Chapter 5. However, the computer implementation of the floor plan design methods still proved to be of great value. This illustrates that the computer is a legitimate research tool in the field of design methodology, whether the end product of the research is a working, practical design program or not. Objectifying a method of problem solving enough to program it for the computer leads to a deeper understanding of that method. But more than this, once a method has been programmed, the computer enables one to explore objectively and with relative ease the effect of changes in the method on the solutions that are produced and the ease with which they are produced. Furthermore, generating and observing large amounts of data under controlled conditions generally leads to the discovery of many things that would have gone uncovered otherwise. This was definitely found to be the case in this thesis. Many of the search tree pruning tests, for instance, came into being because the computer-generated examples showed that they were necessary. Also, it was possible with the computer to measure the amount of effort expended on various aspects of the problem and to optimize the effort in those areas that took the most calculation time. The design methodology described in this thesis would never have been developed as far if it had not been programmed. And conversely, the program would have been more difficult to write if a general model had not been used to guide its construction. This general model, which establishes the vocabulary of this thesis, is described in the next section.

1.4. A Design Program Paradigm

In order to provide a frame of reference a paradigm was developed for a general computer-implemented design program. Since the emphasis of the study is on synthesis, the paradigm embodies the specification phase of the design process only as a set of design requirements (for each design problem) which is provided as an input to the design program. For most cases of interest, however, it is impossible to separate structuring and synthesis in this same simple way. Thus, both of these phases are incorporated in the design program paradigm.

Until recently, a method of structuring very popular among researchers in computer-aided architectural design had been the HIDECS (HIerarchical DEComposition of Systems) approach developed by Christopher Alexander.[5] In this method a design problem is subdivided into smaller problems by identifying subsets of design requirements that interact highly among themselves and minimally with the requirements in other subsets. In the synthesis phase of the design, then, each subset of requirements is supposed to lead to some subcomponent of the completed design.

Just because a design problem can be decomposed according to its requirements, however, does not mean that it is wise to structure a design program in the same way. To utilize the full power of the art a designer must use all pertinent design techniques at his disposal. While a given technique might well be distinguished by the class of design requirements to which it responds, others that are available might be classified in different ways. For instance, a design technique might be oriented toward a particular artifact, regardless of the requirements involved, or toward a particular analytical process, such as cost calculation, or toward a particular optimization algorithm, and so on. Thus, it seems to make sense to structure a design program around the various functions

[5] See Refs. 2 and 3.

that it is capable of performing rather than around sets of design requirements. This approach is the one that is taken in the design program paradigm. A strong argument for this type of structuring of the building design process for computer implementation can be found in Reference 15.

In the paradigm, then, the structuring of the design problem is implicit in a set of fundamental synthesis or analysis operators called <u>design activities</u>. Each design activity is a subroutine of the main program which does something to further the solution of a design problem.[6] A particular design activity could be very well oriented toward the resolution of a certain class of interacting design requirements as in the case of HIDECS. However, any other type of synthesis operator is allowed as well. The design activities related to any particular problem can run the spectrum from simple generate and test routines to sophisticated formally justified algorithms. Their domains of operation can range over such things as design requirements, numerical analysis, or form components of the design solution.

In any non-trivial design problem, the solution does not appear suddenly, but takes shape piece by piece as the design process proceeds. Often the language used to describe a partially completed design conveniently is quite different from that used to describe the finished artifact. This language or set of languages,

[6] The similarity between <u>design activities</u> and the <u>operational methods</u> of GPS (General Problem Solver) is not unintentional. See Ref. 22.

which can be called the design representation,[7] constitutes the second major component of the design program paradigm.

In order for the program to be operational, some decisions have to be made as to which design activities will be applied, when they will be applied, and in response to which design requirements. This control activity is carried out by the third major component of the paradigm, the decision scheme.

In summary then, a design program using this paradigm operates as follows (see Figure 1-1):

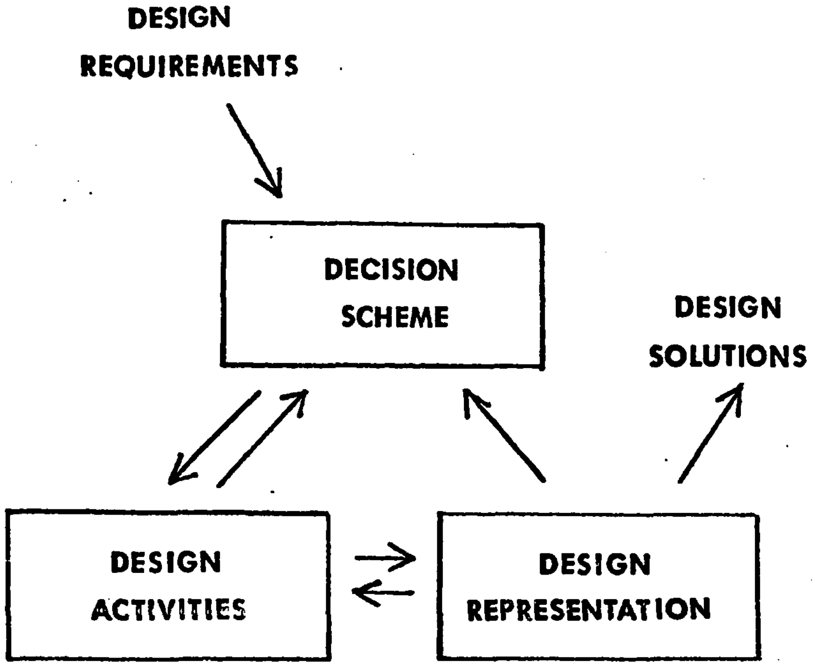
(1) The input to the program is a set of design requirements indicating the desired properties of a design solution.

(2) The output is a (possible empty) set of design solutions each of which satisfies all of the design requirements.

(3) Control is exercised by the decision scheme which looks at the design representation to determine the state of the partially completed design, looks at the set of yet-unsatisfied design requirements, and then decides which design activities to apply next and in response to which design requirements (if any).

(4) A given design activity performs its synthesis or analysis on the basis of information given it by the decision scheme or included in the design representation.

[7] This is a much simplified explanation of what is actually meant by a representation in this thesis. The topic will be developed more fully in Chapter 2. The topic of representations for a large variety of problem solving situations was discussed extensively in a weekly seminar series at Carnegie Institute of Technology in the spring of 1966. Some of the main speakers were Saul Amarel, Allen Newell, and Herbert Simon.



(DIRECTION OF ARROWS INDICATES FLOW OF INFORMATION)

FIGURE 1-1

THE DESIGN PROGRAM PARADIGM

Assuming that, if a solution of the design problem exists,the program can find it in a reasonable amount of time, and assuming that the type of problem is one in which multiple satisfactory solutions are possible, the synthesis action of the program may be terminated in one of three ways:

(1) If no solution can be formed satisfying all design requirements, there is no output, and the problem is <u>over-specified</u>.

(2) If one or a few solutions can be found, they are output, and the problem is <u>well-specified</u>.

(3) If there are many solutions, the first N of them (where N is chosen by the program user) are output in some order determined by the program, and the problem is <u>under-specified</u>.

Any solution which is generated in this way will be called a <u>feasible</u> solution. This model reflects the manner in which GRAMPA generates solutions. However, a design program might also be given some criteria in addition to the design requirements for selecting the <u>best</u> solution out of a set of feasible solutions. The best feasible solution would then be called the <u>optimum</u> solution.

1.5. The General Approach

This paradigm is reflected in the way GRAMPA is constructed. Since the design representation is the single most important item, most of Chapter 2 is devoted to developing and explaining it. Special attention is paid in that chapter to the suitability of the representation for design synthesis.

Chapter 3 describes the design activities and decision scheme that are built around this representation. Some of the design activities are oriented toward a single set of design requirements, while others deal with a mixed set of requirements, and still others test for the feasibility of a design solution.

Some illustrative problems solved by GRAMPA are discussed in Chapter 4. The notions of under-, well-, and over-specified design problems become relevant in that chapter. After reading Chapters 2,3, and 4, it will be clear that the methods and the program presented here are but the foundation for a more complete study of computer-implemented synthesis in floor plan design. Chapter 5, then, proposes extensions and alterations to these and how they might fit into that more advanced study.

2. THE FORMAL PROBLEM AND ITS REPRESENTATION

2.0. Introduction

In the first two sections of this chapter a formal class of floor plan design problems is defined so that the design methods may be developed objectively. Next the linear graph representation is introduced and justified. Properties of the representation that will be useful in the design process are developed extensively. The last two sections of the chapter deal with two references to the literature that are of particular interest.

2.1. The Design Requirements

In this thesis, only a well-defined subportion of the entire architectural floor plan design problem is treated. The first step in defining this special class of problems (hereafter, "The Defined Class") is to determine what form the specification (see section 1.2.) of a problem in The Defined Class takes.

The prime concern here is with the location of spaces relative to one another, rather than with the allocation of activities (such as family living activities, or industrial manufacturing activities) to spaces in the floor plan. It is assumed that as part of the specification of a problem in The Defined Class, a set of rooms is given, and these rooms already have activities assigned to them. Associated with each room can be a set of size requirements determined by the nature of the activities to be carried out in that room. The problem specification can also include requirements for the contiguity and proximity of rooms, as well as ease of communication among them.

Using this initial set of requirements as a guide, it is possible to abstract a set of basic design requirements, such that any requirement can be stated in terms of some combination of requirements from the basic set. These basic requirements are:

(1) Adjacency -- Two rooms or spaces are required to be next to one another.

(2) Communication -- Two rooms or spaces are required to be adjacent and have a doorway joining them.

(3) Physical dimension -- The length or width of a room or space is given.

Since the design representation and design methods to handle this basic set of requirements provide more than enough material to fill this thesis, the discussion of their usefulness in treating higher order requirements such as proximity and area is limited primarily to Chapter 5. Thus, the adjacency requirement and the communication requirement become essentially synonymous. One can always require an adjacency where communication is desired and be guaranteed that he can add doors to the finished floor plan. Therefore, in the rest of this thesis only adjacency and dimension requirements are treated.

2.2. Introduction to the Formalization and the Representation

Overview.- For the sake of objectivity, a simplistic but relatively general class of floor plans is specified in this section for problems of the Defined Class. Next, the term "design representation" is defined and representations currently in use for automated floor plan design are discussed. The idea of using a linear graph for the representation is introduced. In this representation, nodes (or vertices) stand for rooms, and line segments, called edges, connect the nodes and stand for adjacencies between rooms. Some basic graph theory is then explained and the representation used in this thesis is developed. In this representation, the floor plan itself is treated as a linear graph, but it is the dual of the floor plan graph that is used as the design diagram. In this dual graph the nodes stand for rooms and the edges stand for adjacencies, just as stated above. Its chief advantage is that dual graphs can be drawn in which not all room adjacencies are included, and so greater generality of representation is achieved.

The Class of Floor Plans. -- A *building* consists of a set of contiguous *rooms* that fill completely the simply-connected area enclosed by the *walls* of the building. For simplicity, the walls are assumed to have zero thickness. All buildingsand all rooms are rectangles. Buildings are by convention orientated with their walls parallel to the axes of a north-south rectangular co-ordinate system. Thus it is possible to divide the space outside of a building into four rectangular *outside spaces*, called $\mathcal{N}$, $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{W}$ as shown in Figure 2-1. A *space* is defined to be either a room or an outside space. Any two contiguous spaces may have

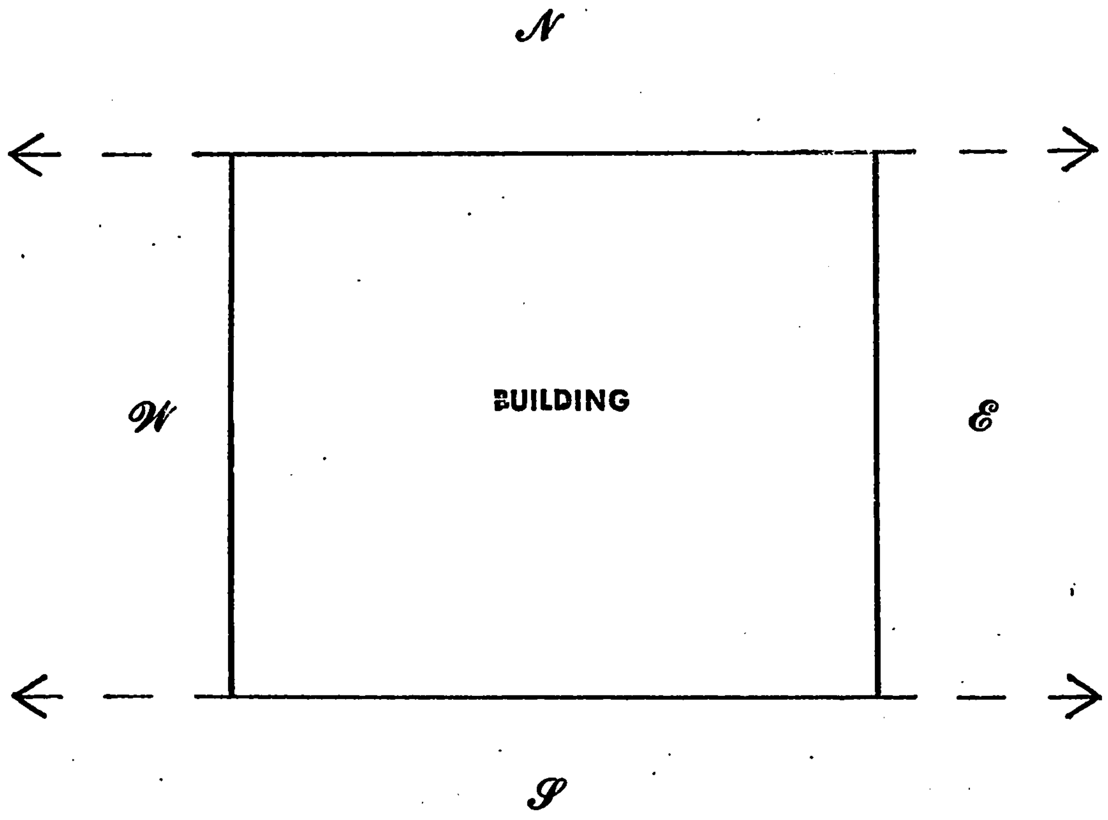


FIGURE 2-1

A TYPICAL BUILDING

at most one door joining them, and the existence of a door is indicated by a double-headed arrow intersecting the wall segment separating the two spaces. A typical floor plan of this class is shown in Figure 2-2. As was mentioned, doors are not specifically treated in this thesis, so hereafter the floor plans will be drawn without them.

What is a Design Representation. -- The next task is to specify a design representation appropriate for the set of basic requirements and the class of floor plans that have been defined. A representation should consist of three parts:[1]

(1) A set of objects suitable for describing the current state of the solution of the problem.

(2) A set of operators for transforming one solution state into another.

(3) A set of prefixes, or tests, for determining whether a given solution state has certain properties (e.g., satisfies certain design requirements).

A supplementary point of view on representations particularly applicable to the design process can be taken from Christopher Alexander's Notes on the Synthesis of Form.[2] Alexander states

[1] The definition of "representation" presented here is taken from a lecture on complex information processing given by Allen Newell at Carnegie-Mellon in the spring of 1967.

[2] See Ref. 3.

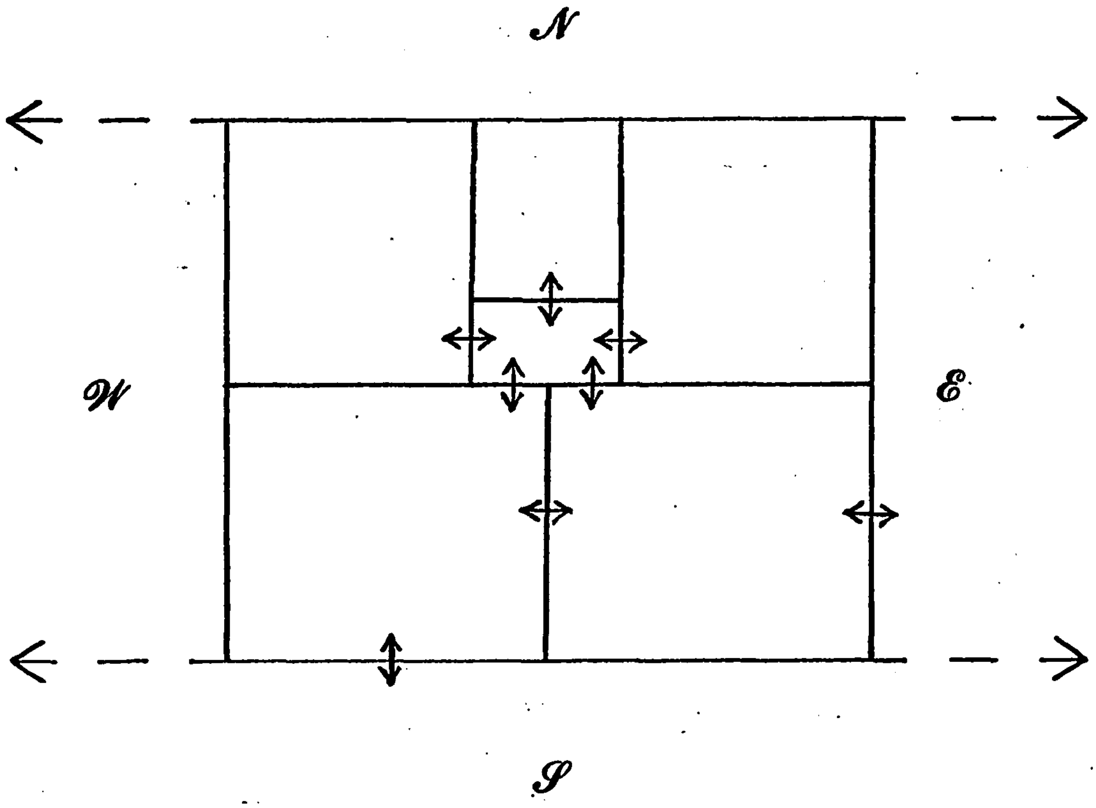


FIGURE 2-2

A TYPICAL FLOOR PLAN

that a diagram, to be useful in synthesis, should be simultaneously both a requirement diagram and a form diagram. That is to say, design requirements should be embodied in a design diagram in terms of the form components of the design solution that satisfy them. Furthermore, if each design requirement maps directly, independently, and uniquely into a single form component of the design diagram, then the synthesis task reduces to simply carrying out the indicated mapping of requirements to form, and the designer is free to experiment with various sets of requirements simply by adding or deleting independent form components at will.

However, some design requirements are not independent of one another and so cannot lead to independent form conponents. Thus, interacting requirements should be represented in the design diagram as well, but requirements which do not interact should not be inadvertently represented as interacting.

Currently Used Representations. -- Most programs for computer implemented floor plan layout do not use representations which have these desired characteristics. A common representation used in such programs written in the last ten years has been a square grid of small modular areas. An activity space is located on this grid by assigning a number of contiguous squares to it. The pattern created by the squares assigned to a given space need not be itself rectangular in shape.

Two different space allocation strategies are generally used with this representation. The first of these, used for instance by CRAFT and by Parson's allocation program,[3] requires that an initial assignment of squares to spaces be made by the program in some arbitrary manner, or by the user. Next the assignments of squares to spaces are methodically interchanged in some manner and a space allocation optimization function evaluated for each new arrangement. The arrangement with the highest value of the optimization function is adopted, the cycle is repeated for this new arrangement, and so on until improvement is no longer possible. This is the familiar hill climbing approach.

The second approach, used by CORELAP, COMSBUL, and the program of Whitehead and Eldars,[4] among others,starts by assigning a single activity space (usually one which has strong proximity requirements with other spaces) to some squares in the center of the grid. It then proceeds by assigning other activity spaces to the squares around the perimeter of this first space according to decision rules that weigh the strengths of the locational relationships among the activity spaces.

Both of these methods deal with design requirements stated in terms of area specifications for each activity space and some sort of relational matrix that assigns a weight to the degree of proximity desired for each pair of spaces. The area requirements are easy to satisfy by assigning squares on the grid to an activity

[3] See Refs. 6, 9, 23.

[4] See Refs. 17, 18, 26.

space, and the relational matrix lends itself well to the evaluation of an optimization function based on the communication costs within a given spatial arrangement.

A second type of representation treats each space to be located as a rectangular area, but does not limit the location of these rectangles to the squares of any grid. Instead, the rectangles are drawn freely in a plane, sometimes partially overlapping one another, but usually conveying the impression of an approximate spatial layout. Such a representation is used in Comprograph III.[5]

These representations do not have the desirable characteristics described earlier because they both deal directly with floor plan drawings -- physical two-dimensional representations of rooms having a definite size and shape and a definite location with respect to one another. Thus, if one wishes to satisfy the design requirement that "room A is adjacent to room B", he cannot indicate that fact unless he also assigns some arbitrary size and shape to each room and places the rooms next to one another along some particular wall segment. But the room sizes and the wall at which the rooms were to be adjacent may not have been specified as design requirements, and arbitrarily determining them just so the rooms can be drawn may introduce unnecessary conflicts later in the design process. Similarly, it would be impossible to satisfy a size requirement on the design diagram without arbitrarily placing the room that was

[5] Comprograph I, II, and III are some of several architectural design programs offered by Design Systems, Inc., Cambridge, Massachusetts.

being sized somewhere on it. Thus, although the literal floor plan representation may allow the direct mapping of a design requirement into form, the mapping is neither unique nor independent of other design requirements.

The Linear Graph Approach. -- The use of a linear graph as a design representation can overcome the major difficulties of the literal floor plan representation. In this representation the rooms can be pictured as labelled nodes possessing certain attributes, such as intended use, area, and shape. Adjacencies between rooms can be indicated by drawing lines connecting the nodes representing those rooms. The room does not have to be physically drawn to indicate that the adjacency requirement has been satisfied.[6] A rudimentary application of this technique as a pencil and paper design method was described by Levin.[7] Casalaina and Rittel[8] in an unpublished paper also discussed the use of a linear graph representation for floor plans. A much more thoroughly developed use of a linear graph representation is described by Krejcirik.[9] Krejcirik's

[6] The author was motivated to use this type of representation because of a suggestion made to him by Saul Amarel at RCA Laboratories in the summer of 1967.

[7] See Ref. 19.

[8] See Ref. 11.

[9] See Ref. 16. Although Krejcirik started his work in 1966, this paper, published in Autumn, 1969, was the first English language description of it available to the author. This was well after the author had completed the theoretical aspects of this work. Although there are marked similarities between Krejcirik's work and the author's, the author doubts that Krejcirik has ever seen a copy of his work.

work was done independently of the author's and during roughly the same period of time. There are strong enough similarities between his work and the author's to warrant special comment in the last section of this chapter.

This section is completed by an introduction to the terminology of the linear graph representation used in this volume. The succeeding sections of this chapter develop this approach much more thoroughly than was done in any of the papers mentioned above.

Introduction to Graph Theory. -- A few terms from the theory of graphs necessary for the understanding of the representation are introduced first. The reader familiar with graph theory can skip this section if he desires. For a more thorough treatment of the theory of graphs, see References 8 and 10. The terminology in graph theory is not standardized, but the author leans toward that used in Reference 10, and has also defined some special terms for his own use. The following explanation draws heavily on the first chapter of Reference 10.

A distinction must be made between the abstract notion of a graph and the specific type of graph known as a geometric graph. It can be shown that, for the purposes of graph theory, any graph is isomorphic with an appropriate geometric graph. Thus geometric graphs give convenient visualizations of all graphs. The following quotation from Reference 10 develops the notion of a geometric graph:

> Euclidean n space... (denoted E^n)... is the space of all sequences of n real numbers $x = (x_1, \ldots, x_n)$, with distance between two points $x = (x_1, \ldots, x_n)$,
>
> $y = (y_1, \ldots, y_n)$ given by $d(x,y) = (\sum_{i=1}^{n} (x_i - y_i)^2)^{\frac{1}{2}}$.
>
> A simple open curve in E^n is a continuous, non-self-intersecting curve joining two distinct points in E^n. A simple closed curve in E^n is essentially a continuous, non-self-intersecting curve whose end points coincide.
>
> A geometric graph in E^n is a set $V = \{v_i\}$ of points in E^n and a set $E = \{e_j\}$ of simple curves satisfying the following conditions:
>
> (1) Every closed curve in E contains precisely one point of V.
>
> (2) Every open curve in E contains precisely two points of V, and these agree with its endpoints.
>
> (3) The curves in E have no common points, except for points of V.
>
> Thus a geometric graph is simply a geometric configuration or structure in E^n consisting of a set of points interconnected by a set of non-intersecting continuous curves.[10]

However, as far as graph theory is concerned, the only essential structural feature of a geometric graph is that each geometric edge has associated with it two (possibly coincident) geometric vertices, or nodes (the word "node" is used in preference to "vertex" in this thesis). Graph theory focuses on these associations and disregards the nature of nodes and edges. Thus, the enumeration of the edges and nodes of a graph, along with a listing of which nodes are incident to which edges (this information can be tabulated in the form of an incidence matrix) provides sufficient information

[10] Ref. 10, p. 4.

to specify that graph. The notion of an abstract graph, then, is separate from its representation as a geometric graph. Usually an abstract graph is simply called a graph.

A geometric graph G is a particular instance of a graph in which the nodes and edges are respectively points and simple curves in E^n and in which the incidence matrix specifies the points between which the simple curves are drawn. Two graphs G = (V,E) and G' = (V',E') are said to be isomorphic to each other if there exist one to one correspondences between V and V' and between E and E' which preserve incidences. If a graph G is isomorphic to a geometric graph G', then G' is called a geometric realization of G. There will be occasion to use this concept in this thesis in reference to geometric realizations of planar graphs. A graph is said to be planar if and only if it has a geometric realization in E^2. It is possible for a given planar graph to have more than one geometric realization (more will be said about this in Section 2.4.1).

Two useful graph structures are the simple chain and the simple circuit. The counterpart of a simple chain in a geometric graph is a sequence of edges joined end to end to form a simple open curve (see definition above). The counterpart of a simple circuit in a geometric graph is a sequence of edges joined end to end to form a simple closed curve (see definition above). In this thesis the terms simple chain and chain will be used interchangeably, as will the terms simple circuit and circuit.

This introduction to graph theory should suffice for present purposes. Additional terminology will be introduced as it becomes relevant to the discussion.

The Dual Graph Representation. -- Setting aside doors, which will not be considered in this thesis, a floor plan of the Defined Class can be drawn in the form of a planar geometric graph, as shown in Figure 2-3. This will be called the floor plan graph, and its edges and nodes will be called wall segments and corners respectively. In a planar geometric graph, the simply connected area bounded by a simple circuit is called a region. The regions of the floor plan graph will be referred to specifically as spaces.

The notion of using nodes to represent rooms, and edges to represent room adjacencies is realized by dealing with the dual of the floor plan graph. The dual of the floor plan graph can be obtained by placing a node inside each space and constructing edges to join the nodes of adjacent spaces. By convention, edges crossing north-south wall segments will be "colored" dotted (......) and directed from west to east, while edges crossing east-west wall segments will be "colored" slashed (------) and directed from south to north. Dimensioning will be accomplished by labelling each edge (except the edges joining the outside spaces to one another) with a number equal to the length of the wall segment that it crosses. Such a dual for the floor plan graph of Figure 2-3 is

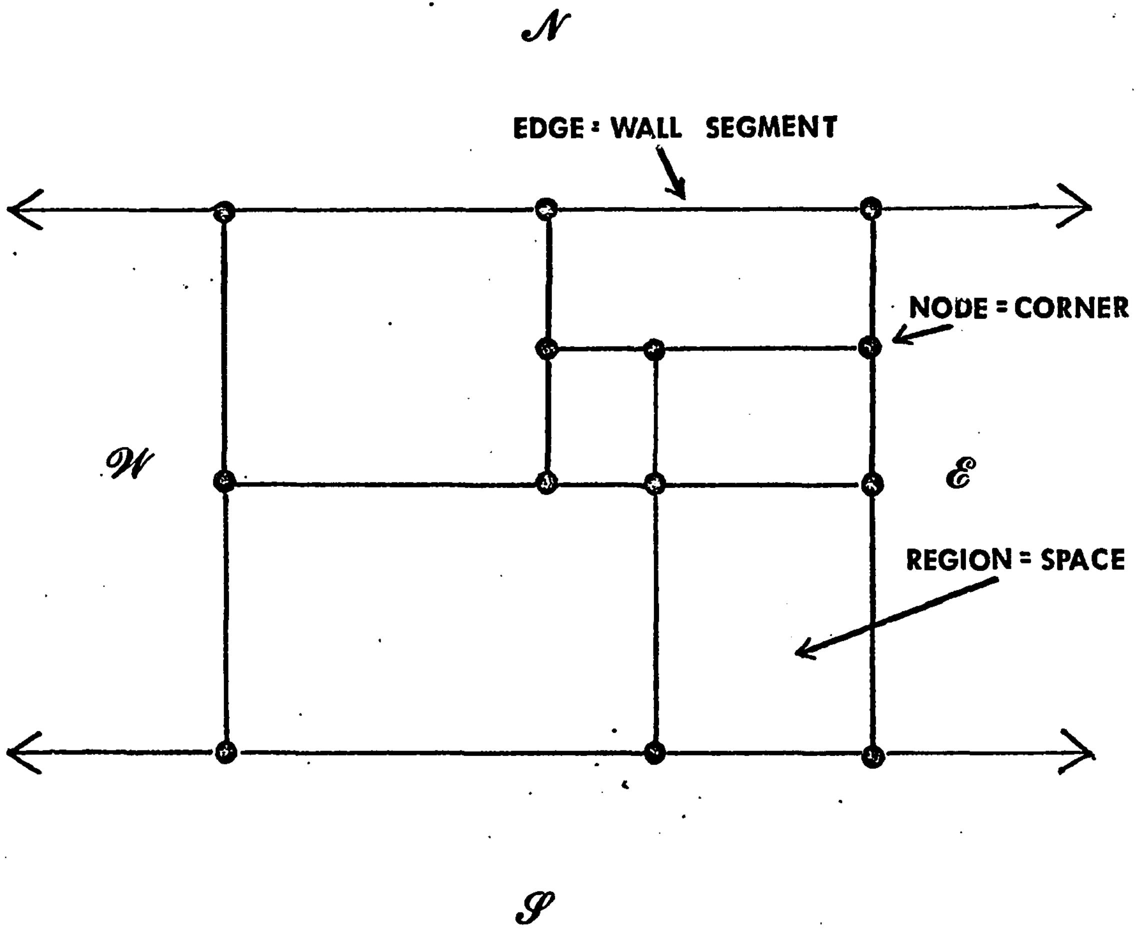


FIGURE 2-3

THE FLOOR PLAN GRAPH

shown in Figure 2-4. In this thesis, the term *the dual graph* will always be taken to refer to this geometric graph which will be used as the diagram portion of the design representation. To distinguish it from the floor plan graph, its local structures will be referred to by the graph theory names of *node*, *edge*, and *region.*

2.3. *Justification of the Representation*

Overview.-- In this section the three components necessary to specify the dual graph representation are identified. The representation is justified in terms of its suitability as a requirement diagram and as a form diagram.

The Components of the Representation. -- The *objects* of the representation are quite easy to identify. They are the *nodes*, *edges*, and *regions* of the *dual graph*. There are *room nodes*, and *outside space nodes*, and the edges can be either *north-south edges* or *east-west edges*.

A *partially completed dual graph* is a dual graph in which not all the nodes and/or edges necessary to specify a complete floor plan are present, and in which some of the edges may lack color and/or direction and/or weight. A dual graph in which the details are all included will be called a *completely specified dual graph.* In using the dual graph representation in the solution of design problems, one starts with the non-dimensioned outline of the building, as specified by the four directed, slashed edges $\mathcal{NE}$, $\mathcal{NW}$, $\mathcal{SE}$, and $\mathcal{SW}$ in Figure 2-4, and then gradually fills in the rest of the

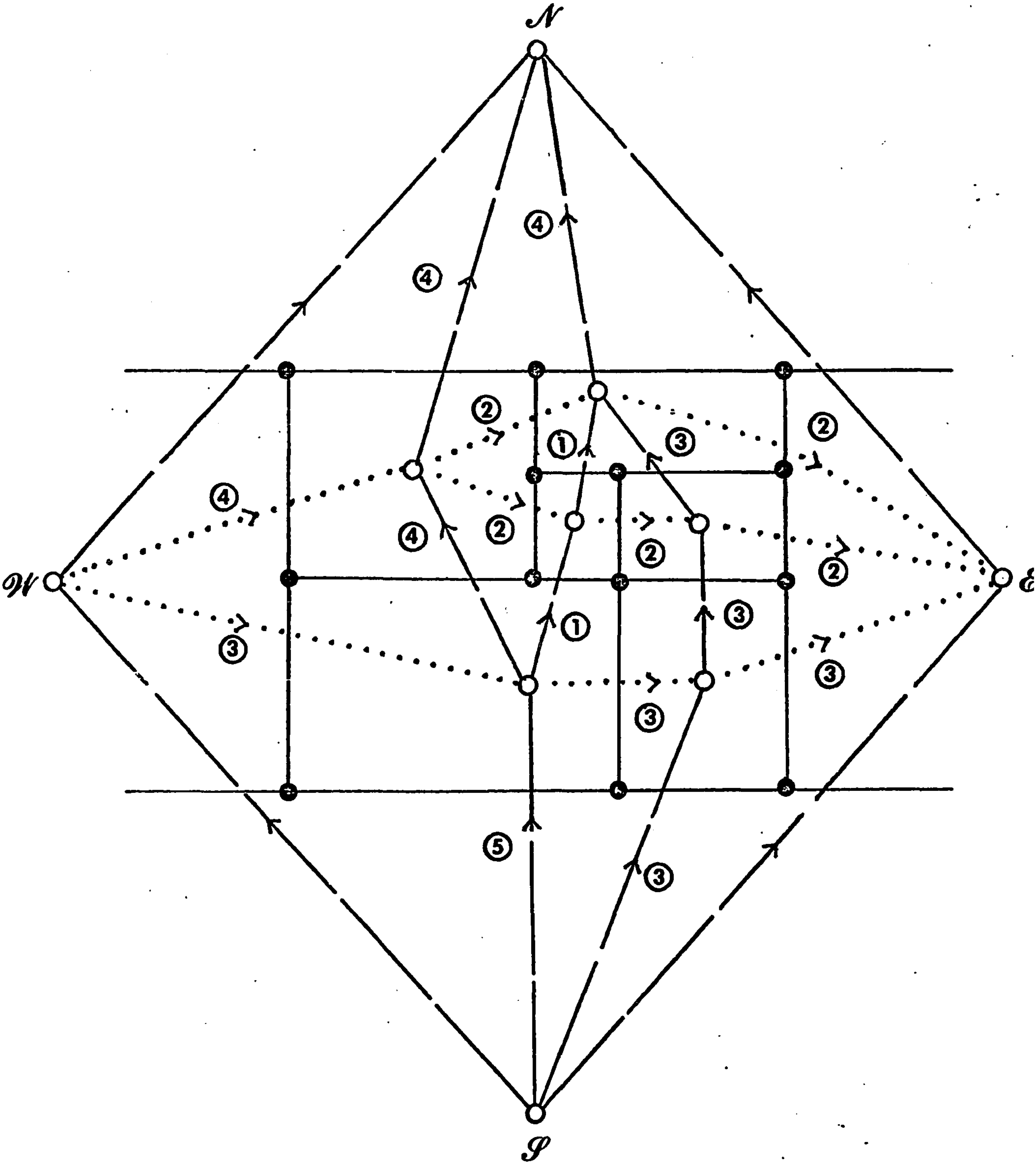


FIGURE 2-4

FLOOR PLAN GRAPH WITH DUAL GRAPH

structure, node by node and edge by edge, in response to the design requirements. Thus a partially completed dual graph represents an intermediate state in the solution to a design problem, and one set of operators transforms one design state into another in response to a given design requirement. The operators corresponding to the two basic types of design requirements are as follows:

(1) Adjacency -- "Space A is adjacent to space B on the north, south, east, west, or 'unspecified' side" becomes "construct an appropriately colored and directed edge between node A and node B," where adjacency on an unspecified side can be indicated by using a non-directed, non-colored edge, i.e. a solid line (______).

(2) Length -- "Wall segment X is L units long" becomes "assign a weight L to edge X."

Since a given set of design requirements will not in general be sufficient to determine a completely specified dual graph, another set of operators is necessary to finish the specification of a partially completed dual graph, according to certain rules of well-formedness. This set of operators is developed in later sections. The rules of well-formedness deal mainly with physical realizability, i.e. whether or not a floor plan can be constructed corresponding to the dual graph that has been generated as the solution to a given design problem. These rules of well-formedness correspond to the prefixes part of the representation. Section 2.4.

deals with several of these prefixes. Another type of prefix is possible: namely a prefix to test a partially completed dual graph for the satisfaction of certain types of design requirements. An example of such a requirement would be the negation of either of the two basic types of requirements for problems in the Defined Class. For instance, one could specify that "room A should not be adjacent to room B." Another example would be a prefix to test whether the total area and/or dimensions of a building fall within certain prescribed limits. Adding such prefixes to GRAMPA is generally easy, and several are used in the examples (see Chapter 4) as an easy way of extending the set of requirements that could be treated. However, the satisfaction of requirements by direct construction, as in the case of adjacency requirements, is much more desirable, when feasible.

The Dual Graph as a Requirement Diagram. -- That the dual graph functions well as a requirement diagram should be quite evident. The adjacency requirements map directly into edges of appropriate color and direction. The physical dimension requirements map into weights on edges, or if the edge has not been constructed yet, they can be treated as attributes of the appropriate space nodes. In all cases the form realizations satisfy exclusively the design requirements which generated them. Any case in which the form realizations interact is immediately picked up by the prefixes testing well-formedness.

The Dual Graph as a Form Diagram -- In order to be useful as a form diagram, the dual graph should be capable of completely specifying a floor plan of the Defined Class. This is indeed the case, since it can be shown[11] that if one takes the dual of a well-formed, completed dual graph, he obtains the floor plan graph. An example of this step for the dual graph of the floor plan of Figure 2-4 is shown in Figure 2-5. It can be shown that the wall segments of the floor plan graph can be straightened and rectangularized appropriately to form a floor plan.

The convention that rooms and buildings be rectangles need not seriously limit the generality of floor plans of the Defined Class. As long as right angled wall intersections are used, irregular buildings can be filled out to rectangular shape by using dummy spaces or "patios", and irregular room shapes can be formed by treating compound spaces as single rooms, as shown in Figure 2-6. To do this with GRAMPA would require some modifications to its present form, but it is not a complicated problem.

A nested hierarchy of structures can be treated by considering the dual graph of a rectangular building as itself a node in a larger graph, or considering a room node as a rectangular space which is to be further subdivided using dual graph techniques. Various attributes other than doors can be assigned to the edges, such as windows, facilities for the distribution of utilities, and so on.

Finally, through the use of uncolored and/or undirected and/or

[11] This topic is developed in greater detail in Section 2.4.

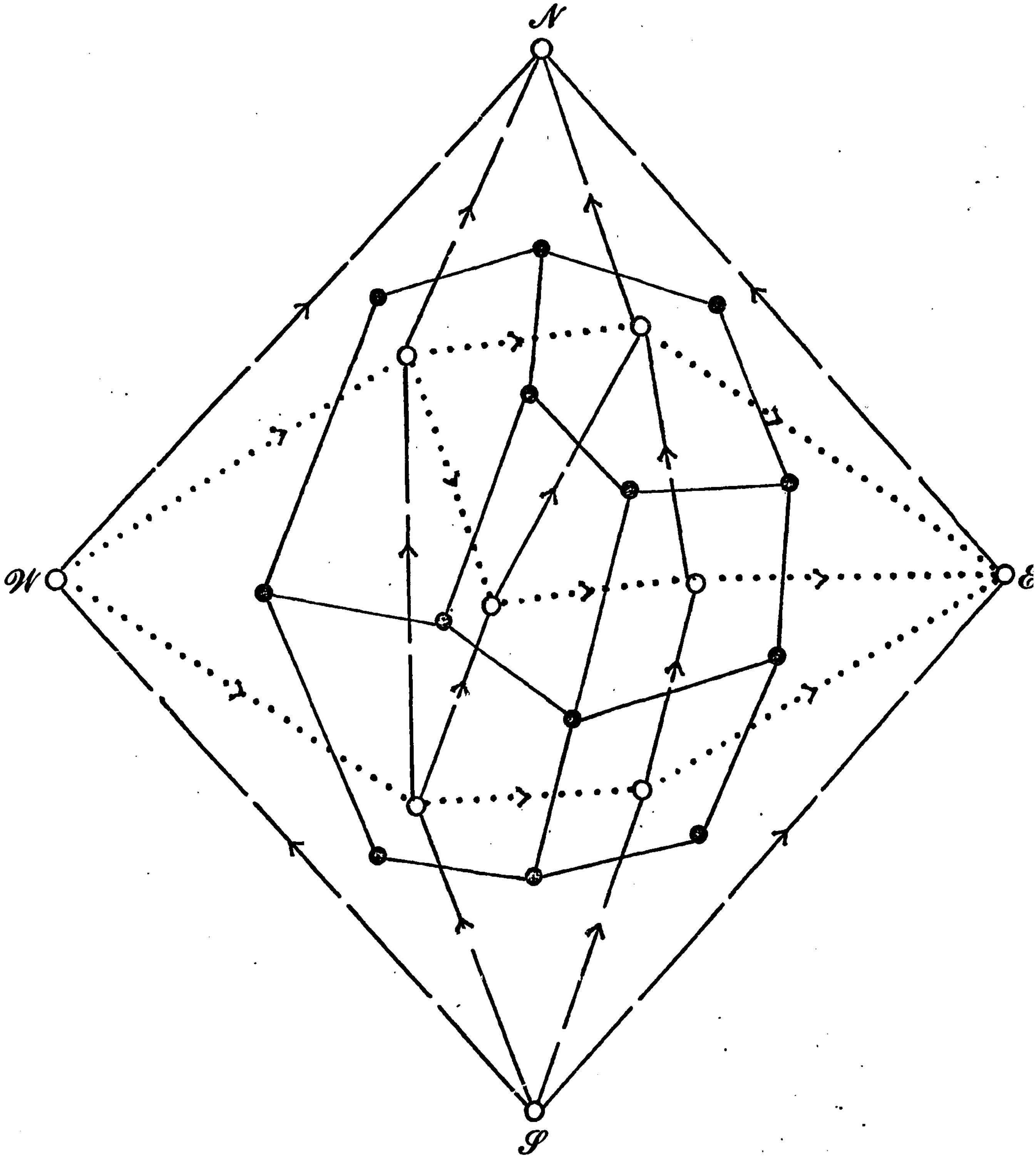


FIGURE 2-5

DUAL GRAPH WITH UNSTRAIGHTENED FLOOR PLAN GRAPH

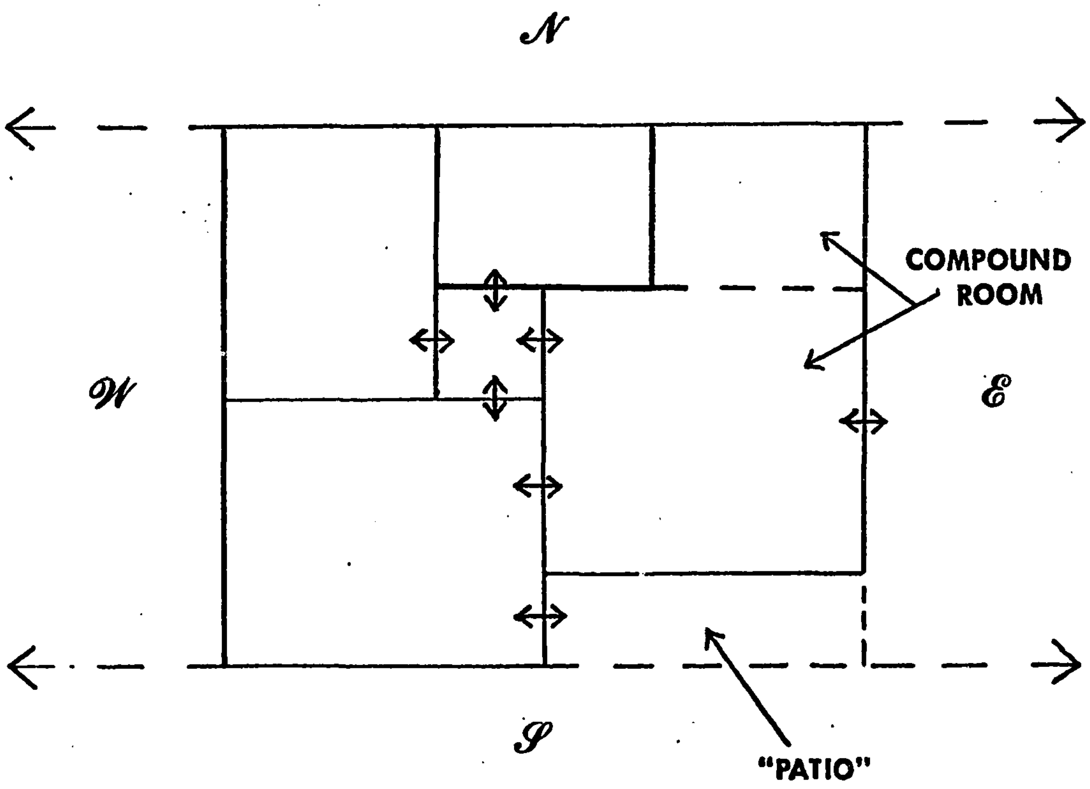


FIGURE 2-6

COMPOUND ROOM AND PATIO

unweighted edges, a partially specified dual graph is capable of representing a whole set of potential floor plans without actually enumerating them, because the attributes of these edges and all missing edges can eventually be assigned in a variety of ways.

2.4. Properties of the Representation

2.4.0. Introduction

When the dual graph representation is used in a design problem, the partially completed dual graph existing at any intermediate stage is assumed to contain form realizations of all design requirements thus far considered. There is no *a priori* guarantee that this dual graph can be extended to a physically realizable floor plan. To make it possible to test whether a dual graph is physically realizable, pertinent properties are derived in this section. Tests based on these properties are described in Chapter 3.

In general, the design requirements do not provide enough information to specify a floor plan completely. Thus, the unspecified details of the dual graph must be provided in a manner that keeps it physically realizable. The properties derived in this section also aid in making this possible.

These properties are concerned mainly with the topological configurations of the dual graph and its related floor plan. The most important property is the planarity of the dual graph. A special grammar is developed which makes it easy to keep track of this planarity. An algorithm, based on this grammar, is given for generating realizations of the dual graph.

The last properties that are developed are related to the fact that the floor plans are composed of rectangular rooms. Several theorems are given concerning the topology of such floor plans.

2.4.1. Planarity

2.4.1.0. Introduction

In the preceding section it was stated that a graph is planar if and only if it has a geometric realization in E^2. In more descriptive terms, this means that a graph is planar if it can be represented on a plane in such a fashion that the nodes are all distinct points, the edges are simple curves, and no two edges meet one another except at their terminals. The two-dimensional floor plan graph is planar, since wall segments do not intersect one another except at corners. Therefore, the dual graph must be planar as well. The dual graph is also simple. A simple graph is one in which no two edges share the same pair of endpoint nodes (no two rooms are adjacent along more than one wall) and no edge has both its ends incident to the same node (no room is adjacent to itself).

In this work the dual graph is used as a representation for floor plans. However it is also necessary to provide a method for representing the dual graph itself to the computer, so that GRAMPA can easily test its planarity and generate realizations for it. In the literature, linear graphs are usually specified by matrices. In one type of matrix, described earlier as the incidence matrix, the rows correspond to nodes of the graph and the columns correspond to edges. An entry of '1' in the matrix indicates that the node of that row and the edge of that column are incident to one another (are connected). An entry of '0' indicates no incidence. Another type of matrix is the circuit matrix, in which the rows stand for circuits and the columns stand for edges. An entry of '1' in this matrix means that the edge of the given column is part of the circuit of the given row.

However, a different representation for the dual graph is developed in this section. This representation exists in a special grammar which is used to hierarchically structure the dual graph. This hierarchical structuring makes it easy to keep track of the planarity of the graph and to generate realizations for it. The realization generation algorithm is developed in detail in this section, while the planarity tests based on this grammar are developed in Chapter 3.

2.4.1.1. The Realizations of a Planar Graph

In Section 2.2 it was stated that a given planar graph may have several different geometric realizations. In graph theory two geometric realizations are not regarded as distinct if they can be made to coincide with one another by plastic deformation of the plane. However, there are other ways in which geometric realizations can be distinct.

Consider the graph specified by the incidence matrix shown in Figure 2-7. One planar geometric realization for this graph is shown in Figure 2-8. Figure 2-9 shows four different realizations

EDGES

NODES	(1,2)	(2,5)	(2,6)	(2,7)	(2,10)	(2,13)	(3,4)	(3,6)	(3,7)	(3,8)	(3,9)	(4,16)	(4,17)	(4,20)	(5,6)	(5,8)	(5,9)	(7,11)	(7,13)	(8,9)	(10,11)	(10,12)	(11,12)	(13,14)	(14,15)	(16,17)	(16,19)	(17,18)	(20,21)	(20,22)	(21,22)	(21,23)	(22,23)	(23,24)	(24,25)	(26,27)	(26,28)	(27,28)	(29,W)	(29,30)	(E,N)	(E,S)	(N,W)	(S,W)
N																																								1		1	
E																																								1	1		
S																																									1		1
W																																							1				1	1
1	1																																											
2	1	1	1	1	1	1																																							
3							1	1	1	1	1																																	
4							1					1	1	1																														
5		1													1	1	1																											
6			1					1							1																													
7				1					1									1	1																								
8										1						1				1																								
9											1						1			1																								
10					1																1	1																						
11																		1			1		1																				
12																								1	1																				
13						1													1					1																			
14																										1	1																		
15																										1																		
16												1														1	1																
17													1													1		1															
18																												1															
19																												1																
20														1															1	1													
21																													1		1	1											
22																														1	1		1										
23																																1	1	1									
24																																		1	1								
25																																			1								
26																																				1	1						
27																																				1		1					
28																																					1	1					
29																																							1	1				
30																																								1				

FIGURE 2-7

A TYPICAL INCIDENCE MATRIX

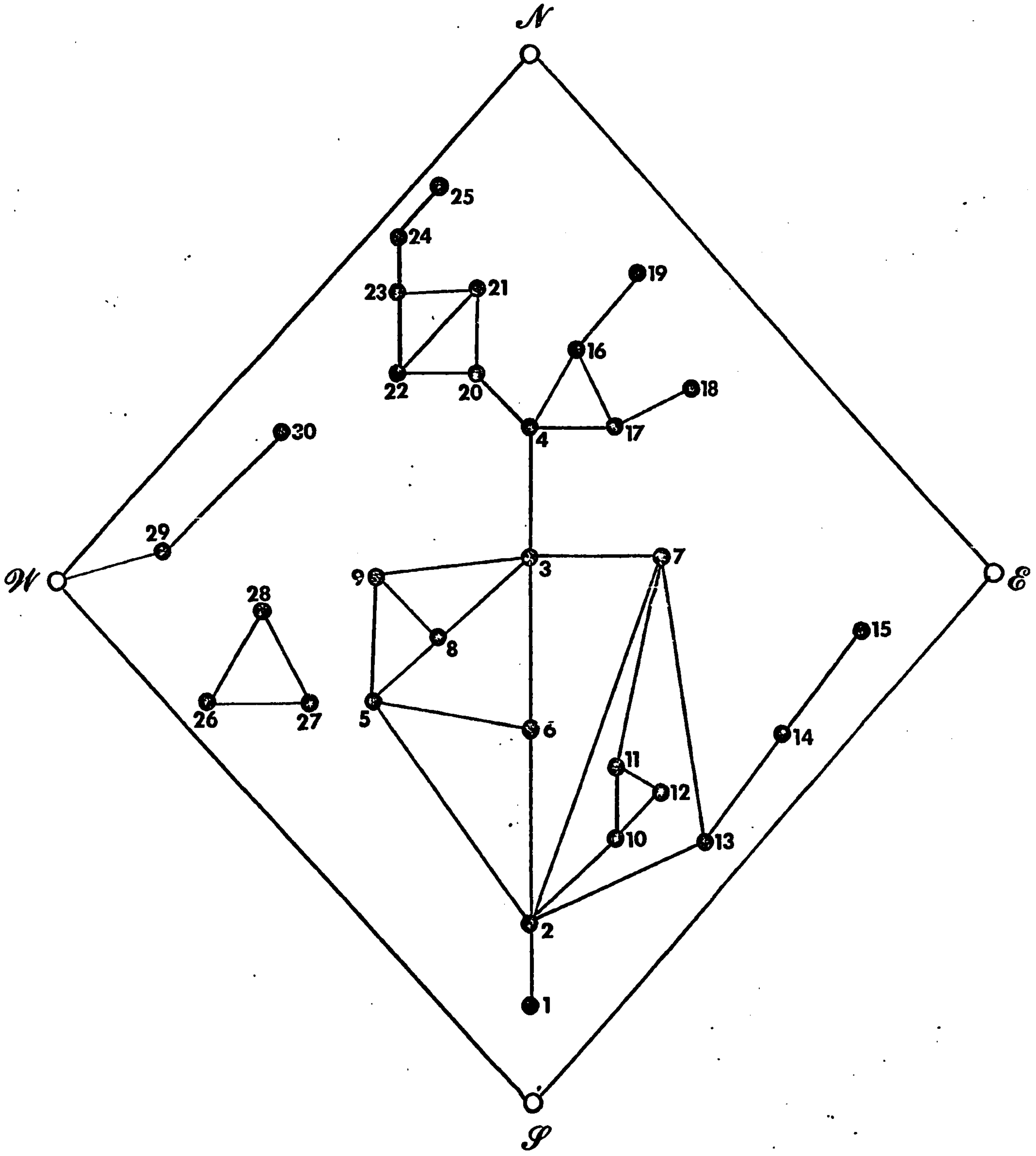


FIGURE 2-8

NON-COLORED, NON-DIRECTED, PARTIALLY SPECIFIED DUAL GRAPH
FOR INCIDENCE MATRIX OF FIGURE 2-7

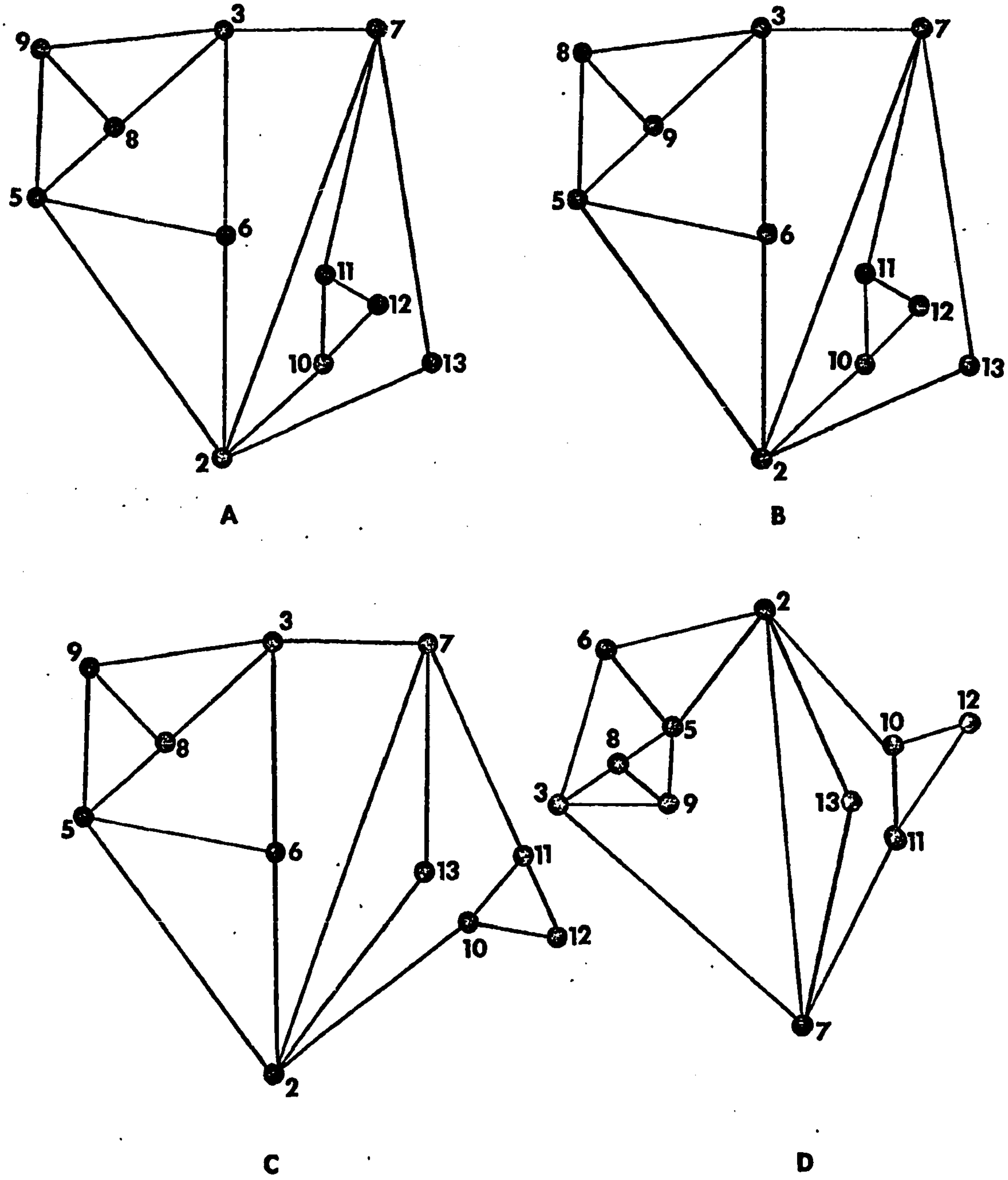


FIGURE 2-9

FOUR REALIZATIONS FOR A SUBGRAPH OF FIGURE 2-8 (the adjective "geometric" will be dropped for brevity) for a subgraph of the graph of Figure 2-8. Each different realization of a dual graph corresponds to a different layout of rooms in the floor plan that can be generated from that graph. Since the number of realizations for a given graph can be quite large (it will be seen that the subgraph shown in Figure 2-8 has 432 different realizations), it is important that a method be provided for determining how many realizations a given graph has, and for enumerating them systematically.

But first a method is needed for testing objectively whether one realization of a given graph is distinct from another. A further discussion of the concept of a region will lead to such a method. Recall from Section 2.2 that a region is defined only for the planar realization of a graph. It is a simply-connected area of the plane bounded by edges of the graph. The boundary of the region is the circuit of edges and nodes that surrounds the region. By convention, the direction of reading the boundary of a region is taken as positive when the interior of the region is to the right as the boundary is traversed. Two regions are said to be adjacent if their boundaries have at least one edge in common; two regions which meet only at a node are not adjacent. The area surrounding a planar graph is called the outside region (or the infinite region). The remaining regions are finite regions.[12] By convention the boundary of the outside region is the outside boundary of the dual graph, read so that the outside region is always to the right.

[12] The definitions in this paragraph are taken from Ref. 8, although the terminology has been changed to that of Ref. 10 and the author.

Only edges of a graph that are part of the boundary of some region are involved with planarity. Thus, one planar realization of a connected graph will be considered as distinct from another if its set of region boundaries, read in the positive direction, is distinct from the set of region boundaries of the other realization, and/or if its outside region differs from that of the other realization. A connected graph is one in which there exists at least one chain linking each pair of nodes.

2.4.1.2. A Planar Graph Grammar

Overview. -- A grammar for describing a planar graph is developed in this section to answer the needs raised in the preceding paragraphs. The general intent of this grammar is to divide a graph into a set of hierarchically organized subgraphs. Each of these subgraphs can be treated as a separate entity, and if it is connected to the rest of the graph, it is by at most two nodes. These subgraphs are free to rotate about these nodes to create the various realizations of the graph. Other such "degrees of freedom" in creating realizations are also possible.

This grammar allows one to deal with planarity in an inductive manner. That is, given a graph that is planar and described in terms of the planar graph grammar, if a new edge is to be attached to that graph, a test can be developed to tell whether the resultant graph will also be planar. This test, which is described in Chapter 3, assumes that each of the special subgraphs is already planar. It then checks to see if the proposed edge can be connected to its two endpoints without crossing an edge of one of these subgraphs.

The terminology of linguistics will be used in describing this grammar. However, it must be kept in mind that this is intended to be a grammar for graph structures rather than for strings. The planar graph grammar developed here (call it the PGG) is intended for describing the *deep structure* of a planar graph. That is, the PGG is used to describe an abstract planar graph and not the geometric realization of that graph. From the PGG description, a *generation algorithm* can be used to create the various possible geometric realizations of a planar graph. Each geometric realization will be considered to be a possible *surface structure* for the graph.

A surface structure generated from the grammar is comprised of *terminal structures*, arranged in some configuration determined by the syntax of the grammar and the generation algorithm. These terminal structures are the nodes and edges of the geometric graph.

In the PGG the deep structure of a graph is described in terms of both terminal and *non-terminal structures*. In addition to the terminal nodes and edges, there are non-terminal structures called *tree structures* (abbreviated TS), *region structures* (abbreviated RS), and *pivot pair sets* (abbreviated PP). Each non-terminal structure corresponds to some subgraph of the dual graph under description.

Each non-terminal structure *is composed of* one or more terminal and/or non-terminal structures. These structures are said to *be a part of* (or, collectively, *to comprise*) that non-terminal structure, and any given structure can be part of at most one non-terminal structure. Quite separate from this, there is also a parent-descendant relationship. The structures which are a part of a non-terminal structure are its direct descendants, and it is their parent structure. However, it is also possible for a non-terminal structure to have direct descendants which are not a part of it. A type of TS known as an anchored TS is the only non-terminal structure which is not a part of its parent structure. All terminal structures are a part of their parent structures. Each structure has at most one parent structure. A structure which has no parent structure is called floating.

Briefly speaking, (1) an RS is a connected subgraph consisting entirely of adjacent circuits. A typical RS is shown in Figure 2-10-A. (2) A TS is a chain which has been generalized so that RS's can be freely substituted for edges, as shown in the TS of Figure 2-10-B. The parent-descendant relationship described above results because these subgraphs can be connected to one another in well-defined ways. RS's and TS's can be connected to certain other subgraphs at either one or two nodes, and the TS has the additional possibility of not being connected to anything. In Figure 2-10-C, TS2 and TS3 are connected to TS1, and RS1 is also connected to TS1 (in fact, it is a part of TS1).

(3) A PP is a subgraph composed of a set of TS's which are connected to an RS at two common points, as shown in Figure 2-11-A. The details of the connection rules for these structures are given

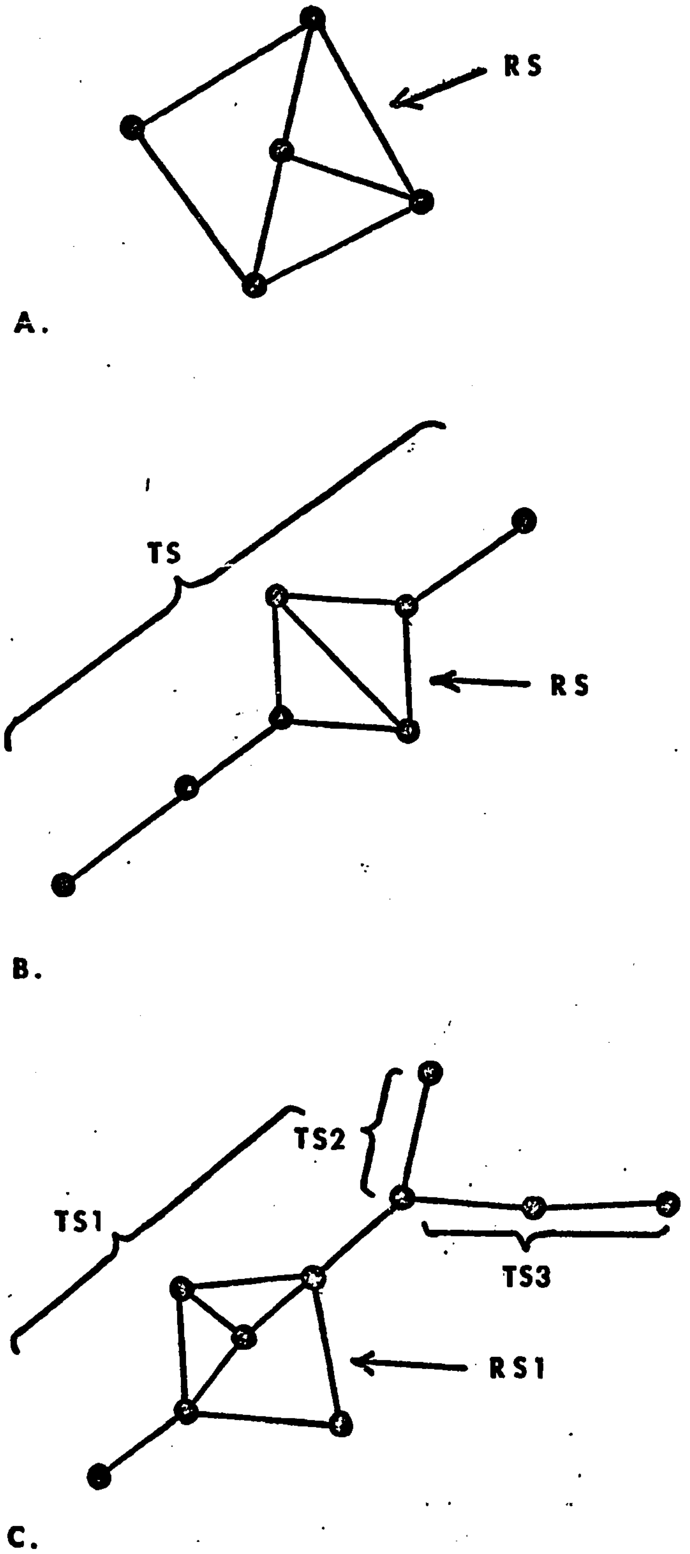
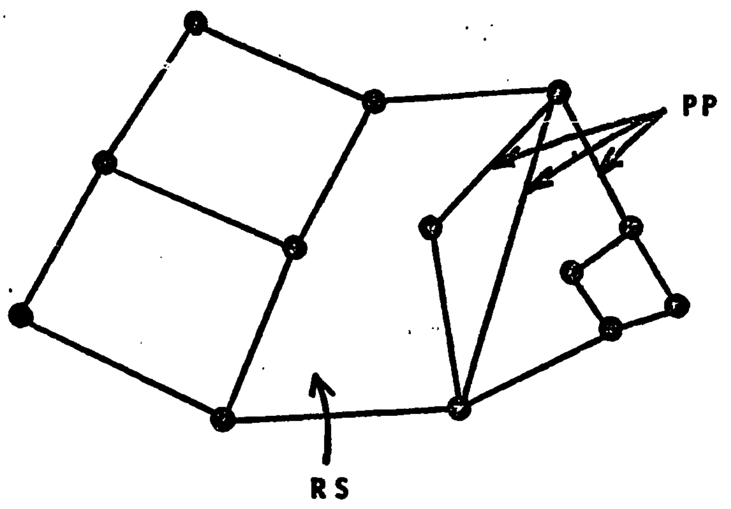
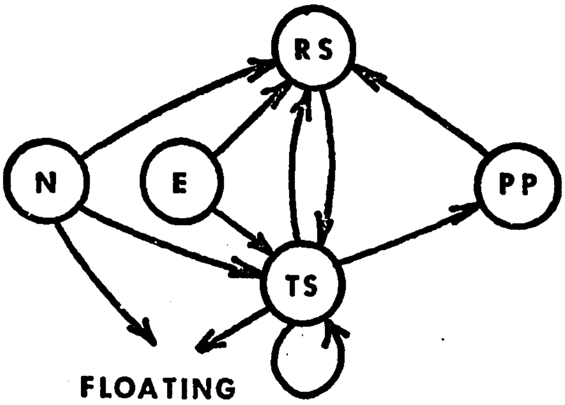


FIGURE 2-10

EXAMPLES OF PGG STRUCTURES



A. EXAMPLE OF A PP



KEY

N = NODE
E = EDGE
TS = TREE STRUCTURE
RS = REGION STRUCTURE
PP = PIVOT PAIR SET

B. THE PARENT-DESCENDANT HIERARCHY

FIGURE 2-11

A PP, AND THE PARENT-DESCENDANT HIERARCHY in the following paragraphs, but a rough idea of the allowable hierarchy of parent-structure relationships is given in Figure 2-11-B. In this figure, the arrows join allowable parent-descendant pairs, and point from descendant to parent. As was mentioned, different realizations of the graph can be achieved by rotating the various structures about their points of connection.

The following paragraphs describe in detail the syntax for each of the non-terminal structures and establish some terminology used elsewhere in the thesis in speaking of the grammar. Extensive examples of the non-terminal structures are given at the end of this section.

Introduction to the RS. -- Recall that one realization of a given connected planar graph is distinct from another if its set of region boundaries, read in the positive direction, is distinct from that of the other, and/or if its outside region is different from that of the other.

Now consider a planar graph G which has the following properties:

(1) G is simple and connected

(2) Every edge of G is part of a circuit.

(3) Every circuit of G shares at least one edge with at least one other circuit of G.

(4) All realizations of G contain exactly the same set of region boundaries. That is, any given realization of G, call it R, differs from any other realization of G, call it R', only in one or both of the two following ways:

(a) R' has a differen+ outside region boundary from R.

(b) The region boundaries of R' are all inverted from those of R. That is, although the boundaries contain the same structures as R in the same sequence, the sequence must be read in inverse order to keep the interior of the region to the right.

Call any graph which has these properties an R-graph. An example of such a graph is Figure 2-10-A. Besides its unique manner of generating realizations, an R-graph has the property that its component structures can be enumerated by listing the region boundaries of any of its realizations, since, subject to inversion, the region boundaries of all its realizations are identical. This type of graph forms the model for the non-terminal structure called a region structure (RS).

Roughly speaking, an RS is a graph that satisfies the same rules as an R-graph, but for nodes and g-edges instead of nodes and edges. A g-edge is either an edge or the non-terminal structure called a pivot pair set (PP). An RS which contains a PP was shown in Figure 2-11-A. The rules that applied for generating realizations of an R-graph can be applied for generating realizations of an RS, but only down to the level of detail of nodes and g-edges. These can be called the generalized realizations (g-realizations). To differentiate between regions with boundaries composed of nodes and edges, and regions with boundaries composed of nodes and g-edges, the former will be called conventional regions (c-regions) and the latter generalized regions (g-regions). It will become clear later that the class of graphs describable as RS's is the class that satisfies, for nodes and edges, the first three rules given for R-graphs. This is much larger than the class of R-graphs. However, to generate conventional realizations of RS's, more must be known about generating realizations for PP's. Information on this follows later.

The rules which specified R-graphs implied the knowledge of an algorithm for generating realizations, and thus for generating the g-realizations of an RS. However, there is an alternate way for specifying the syntax of an RS which avoids the necessity of using a generation algorithm at this stage. This is given in the following paragraphs.

Syntax of the RS. -- Let L be the list of region boundaries of an RS. The set of structures which comprise an RS is equal to the union, S, of the sets of structures which occur on L, with one or possibly two structures of S excluded. One of these excluded structures is the anchor node of the RS, and the other, if any, is either its brace node or its pseudo brace node. These structures are described in the syntax of the tree structure later. In order for an RS to be well-formed, the list of boundaries is subject to certain limitations:

(1) Each boundary of L must share at least one g-edge with some other boundary of L.

(2) No two nodes of S may occur in more than two boundaries of L in common.

(3) Define a boundary portion to be a connected sequence of g-edges and nodes on a boundary. If two structures of S appear on more than one boundary in common, both structures must appear on a single boundary portion in each boundary.

These conditions can be tested for without generating g-realizations for the RS. They guarantee that no g-realizations exist with boundaries other than those on L (read all positive or all inverted).

The parent structure of an RS is always a tree structure, as described in the following paragraphs.

Syntax of the TS. -- In graph theory a graph is said to be a tree if it is connected and has no circuits. In the PGG, just as an RS is a collection of generalized circuits, a collection of connected tree structures is a generalized tree. A single tree structure (TS) is a generalized chain. These two types of structures were shown in Figures 2-10-C and B, respectively. The structures which comprise a TS are enumerated by means of a map of the TS, to be explained in detail below. Whereas a conventional tree is composed of nodes and edges, a TS is composed of nodes and t-edges, where a t-edge is either an edge or a region structure. A brief introduction to the hierarchical organization of TS's is given below, before going into detail on TS syntax.

To keep the concept of a map simple, no map is allowed to have any branches. Since trees normally do branch, this means that each branch of a generalized tree becomes a distinct TS. These TS's are organized in a hierarchical parent-descendant relationship. One TS may be a descendant of another TS and if so it is said to be anchored to its parent, and is called an anchored TS. (A TS may also be anchored to an RS, but more about that later.) Recall that an anchored TS is the only structure that is not a part of its parent structure. An anchored TS is attached to its parent structure at a node called the anchor node of the anchored TS. The anchor node is a part of the parent structure. Several TS's may be anchored to the same node. All TS's anchored to a given node occur on the structures attached list (SAL) of that node. The SAL's provide the means for keeping track of the branching of the generalized trees described by the TS's. Remember that an anchored TS is the descendant of the parent structure of its anchor node, but it does not belong to its parent structure.

In addition to anchored TS's, there are floating TS's and braced TS's. The syntax for all three is given below:

(1) A floating TS is a TS that has no parent structure and that is not a part of any structure. TS1 in Figure 2-10-C is a floating TS. The map of a floating TS is a list of nodes and t-edges that comprise the TS. The nodes and t-edges are listed in alternating order, indicating the actual connectivity of these structures in the abstract graph that the TS represents. Several formation rules hold for the map of a floating TS:

(a) The first structure on the map must be a node. This node is called the anchor node of the TS, although its only similarity with the anchor node of an anchored TS is that it dictates the order of the TS map.

(b) The structures after the anchor node must be alternating t-edges and nodes, subject to the termination conditions listed in (c) and (d) below.

(c) The last structure on the map may be an RS. If so, it is the only RS on the map that may have direct descendant nodes that have non-empty SAL's. As a matter of convenience, an RS has as an attribute a master SAL that contains the names of all anchored TS's that appear on the SAL's of the nodes that are part of it.

(d) The last two structures on the map may be an edge and a node, in that order. If this is the case, this last node is the only node of the TS that may have a non-empty SAL. Thus rules (c) and (d) guarantee that only the last structure on the map of a floating TS may have descendant anchored TS's branching out from it. Notice that a node immediately precedes an RS on the map, and if there is any structure after an RS on the map it is also a node. These nodes are those mentioned above which belong to the set S of structures that occur on the boundary list of the RS, but that are not a part of the RS. Instead they are a part of the TS, and they represent the only graphical connection that the RS has with the TS. The node of these two that is closer to the anchor node end of the map is called the anchor node of the RS. The other node of these two, if it exists (i.e. if the RS is not the last structure on the map), is called the pseudo brace node of the RS. An RS on the map of a floating TS is called an anchored RS.

(2) An anchored TS is a TS that has an RS or another TS as its parent structure. (TS2 and TS3 in Figure 2-10-C are anchored TS's.) Its anchor node is a node that is a part of its parent structure. Otherwise an anchored TS is exactly like a floating TS in every respect but one. The first structure on the map of an anchored TS must be a t-edge rather than a node. This t-edge is incident to the node to which the TS is anchored. Even though this node is not a part of the TS and consequently does not appear on its map, it is still called the anchor node of the TS.

(3) A braced TS is a TS which is graphically connected at both ends of its map to two nodes of an RS. (The PP of Figure 2-11-A is comprised of three braced TS's.)

These two nodes are nodes which occur immediately before and after the occurrence of a pivot pair set (PP) in some boundary of that RS. These nodes are called the anchor node and brace node of the PP, and also of the braced TS. The PP is the parent structure of the braced TS and the braced TS is a part of the PP. The map of a braced TS enumerates the structures that belong to it and is constructed according to slightly different rules than the map of a floating or anchored TS.

(a) The first structure on the map must be a t-edge and that t-edge must be graphically connected to the anchor node of the TS. Again the TS anchor node is not on the map and is not a part of the TS.

(b) The structures following the first structure must be alternating nodes and t-edges as in the floating and anchored TS's and the last structure on the map must be a t-edge that is graphically attached to the brace node of the TS. Like the anchor node, the brace node does not appear on the map and is not a part of the TS. It is possible for a braced TS map to consist of a single t-edge.

(c) Any node that is part of a braced TS or that is part of an RS that is part of a braced TS may have a non-empty SAL. The node that occurs before an RS on the map of a braced TS is called its anchor node, and the node occuring after it is called its brace node. If the RS is the first or last structure on the map, it takes the anchor node or brace node of the TS as its anchor node or brace node respectively. An RS on the map of a braced TS is said to be a braced RS.

Syntax of the PP. -- Just as the RS appears as a generalized edge in a TS, the pivot pair set (PP) appears as a generalized edge in an RS. (A typical PP was shown in Figure 2-11-A.) The RS is the parent structure of the PP, and the PP is a part of the RS. Depending on the context, a PP is comprised of one or more braced TS's. The purpose of the PP is to enlarge the class of abstract graphs that an RS is capable of describing. The braced TS's of a given PP can be thought of as parallel connected graph structures, since they share the same anchor node and brace node. The degree of variability that a PP introduces to the generation of graph realizations is that of the order in which its TS's are constructed in the plane. Each permutation of the order of the braced TS's corresponds to a different set of region boundaries, and hence, a different graph realization. More will be said about this in the next section.

As has been mentioned, the structures that comprise a PP are enumerated as a list of braced TS's In all contexts but one the list must contain two or more TS's. The sole exception is that a PP may consist of a single braced TS if that TS has a map consisting of a single RS, and the RS which is the parent of the PP has three or more boundaries.

Two restrictions are placed on the syntax of the PGG to insure that all parallel tree structures between two nodes belong to the same PP. First, no two separate PP's may have the same anchor and brace node. Second, if a braced TS has a single structure on its map, and if that structure is an RS, then one and only one region boundary of that RS may contain both the anchor and the brace node of that RS. This prevents the occurrence of a single braced TS that can in fact be structured as two braced TS's that are part of the same PP.

The Main RS. -- When the program described in this thesis applies the PGG to the dual graph design diagram, one RS is considered to be the main RS. This is the RS which contains the four outside space nodes $\mathcal{N}$, $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{W}$. This RS is not considered to be anchored or braced, and it has no parent structure. In a completely specified dual graph, all nodes and edges are a part of the main RS.

Any anchored TS or TS that has the main RS as its ancestor in the parent-descendant hierarchy is considered to be *positively anchored*.

Note.-- It has not been proved in this section that all planar graphs can be described in terms of the PGG. This issue is taken up in Chapter 3. Examples of the various non-terminal structures of the PGG are listed in Table 2-1, which refers to the dual graph in Figure 2-8.

TABLE 2-1

THE STRUCTURES OF FIGURE 2-8

KEY

* An edge is denoted by its endpoint nodes, e.g. (1,2). Order is not important, e.g. (1,2) is equivalent to (2,1).

* A region is defined by its boundary circuit, taken in clockwise order.

* A region structure is defined by its regions.

* A tree structure is defined by its map.

* A pivot pair set is defined by its set of tree structures.

NODES

Name	Parent Structure	Name	Parent Structure
N	RS7	14	TS12
E	RS7	15	TS12
S	RS7	16	RS5
W	RS7	17	RS5
1	TS1	18	TS4
2	TS1	19	TS5
3	TS1	20	TS3
4	TS1	21	RS6
5	RS1	22	RS6
6	RS1	23	TS14
7	RS1	24	TS15
8	RS2	25	TS15
9	RS2	26	TS6
10	TS9	27	RS4
11	TS9	28	RS4
12	RS3	29	TS7
13	TS10	30	TS7

(Continued on next page)

TABLE 2-1

(Continued)

EDGES

Name	Parent Structure	Name	Parent Structure
(1,2)	TS1	(11,12)	RS3
(2,5)	RS1	(13,14)	TS12
(2,6)	RS1	(14,15)	TS12
(2,7)	TS8	(16,17)	RS5
(2,10)	TS9	(16,19)	TS5
(2,13)	TS10	(17,18)	TS4
(3,4)	TS1	(20,21)	RS6
(3,6)	RS1	(20,22)	RS6
(3,7)	RS1	(21,22)	TS13
(3,8)	RS2	(21,23)	TS14
(3,9)	RS2	(22,23)	TS14
(4,16)	RS5	(23,24)	TS15
(4,17)	RS5	(24,25)	TS15
(4,20)	TS3	(26,27)	RS4
(5,6)	RS1	(26,28)	RS4
(5,8)	RS2	(27,28)	RS4
(5,9)	RS2	(29,W)	TS7
(7,11)	TS9	(29,30)	TS7
(7,13)	TS10	(E,N)	RS7
(8,9)	RS2	(E,S)	RS7
(10,11)	RS3	(N,W)	RS7
(10,12)	RS3	(S,W)	RS7

REGIONS

Name	Boundary	Parent Structure
R1	3,(3,6),6,(6,5),5,PP2	RS1
R2	3,(3,7),7,PP1,2,(2,6),6,(6,3)	RS1
R3	6,(6,20,2,(2,5),5,(5,6)	RS1
R4	9,(9,8),8,(8,5),5,(5,9)	RS2
R5	3,(3,8),8,(8,9),9,(9,3)	RS2
R6	9,(9,5),5,(5,8),8,(8,3),3,(3,9)	RS2
R7	7,(7,3),3,PP2,5,(5,2),2,PP1	RS1
R8	11,(11,12),12,(12,10),10,(10,11)	RS3
R9	11,(11,10),10,(10,12),12,(12,11)	RS3
R10	28,(28,27),27,(27,26),26,(26,28)	RS4
R11	28,(28,26),26,(26,27),27,(27,28)	RS4
R12	16,(16,4),4,(4,17),17,(17,16)	RS5
R13	16,(16,17),17,(17,4),4,(4,16)	RS5
R14	21,(21,20),20,(20,22),22,PP3	RS6
R15	21,PP3,22,(22,20),20,(20,21)	RS6
R16	N,(N,E),E,(E,S),S,(S,W),W,(W,N)	RS7

(Continued on next page)

TABLE 2-1

(Continued)

REGION STRUCTURES

Name	Region List	Parent Structure
RS1	R1,R2,R3,R7	TS1
RS2	R4,R5,R6	TS11
RS3	R8,R9	TS9
RS4	R10,R11	TS6
RS5	R12,R13	TS2
RS6	R15,R15	TS3
RS7	R16	none

TREE STRUCTURES

Name	Map	Parent Structure	Comments
TS1	1,(1,2),2,RS1,3,(3,4),4	none	floating
TS2	RS5	TS1	anchored
TS3	(4,20),20,RS6	TS1	anchored
TS4	(17,18),18	RS5	anchored
TS5	(16,19),19	RS5	anchored
TS6	26,RS4	none	floating
TS7	(W,29),29,(29,30),30	RS7	pos. anch.
TS8	(2,7)	PP1	braced
TS9	(2,10),10,RS3,11,(11,7)	PP1	braced
TS10	(2,13),13,(13,7)	PP1	braced
TS11	RS2	PP2	braced
TS12	(13,14),14,(14,15),15	TS10	anchored
TS13	(22,21)	PP3	braced
TS14	(22,23),23,(23,21)	PP3	braced
TS15	(23,24),24	TS14	anchored

PIVOT PAIR SETS

Name	Tree Structure List	Parent Structure
PP1	TS8,TS9,TS10	RS1
PP2	TS11	RS1
PP3	TS13,TS14	RS6

2.4.1.3. The Realization Generation Algorithm

Overview. -- An algorithm for generating the realizations of an RS is developed in this section. The RS was chosen because regions are the factor that allow one to distinguish one graph realization from another.

The algorithm works by rotating and twisting RS's, TS's, and PP's about their connection points in all possible ways. Care must be taken to insure that this is done in an exhaustive, non-redundant way. This section explains these aspects of the algorithm.

Some existing work in the graph theory literature describes a realization generation technique that is similar to this one. However, that technique was developed for a somewhat different problem formulation. Details are given in Section 2.6.

Degrees of Freedom. -- The realization generation algorithm is centered around the notion of the degrees of freedom of an abstract graph. These correspond to the categories of ways in which the graph can be varied in generating geometric realizations. The two degrees of freedom for g-realizations of an RS have already been introduced. These are the orientation of all g-region boundaries of the RS, and the selection of an outside g-region boundary from among the boundaries of the RS. Note that there are exactly two different orientations possible for an RS, one the inverse of the other. The number of possible outside g-regions is equal to the number of boundaries of the RS, although this degree of freedom will be somewhat modified later.

The generation of a g-realization for an RS still leaves the possibility of non-terminal structures, the PP's which are a part of the RS. These PP's are composed of braced TS's that may have other RS's on their maps. These RS's may have other PP's, and so on. This makes it necessary to introduce the PP's into the generation of realizations if conventional realizations are to be generated, and also to make special rules for including the realizations of the braced RS's which are descendants of PP's.

The degree of freedom that the PP introduces is the particular order in which the braced TS's that comprise the PP are drawn in the plane. Recall that each braced TS shares the same anchor and brace node as the PP, so the set of TS's get drawn as a sequence of parallel structures in the plane. These TS's then become parts of the boundaries of g-regions for the RS which is the parent of their PP. (Assume that any RS that was part of one of these TS's is taken care of properly. More will be said about this below.) Each different sequence produces a different set of g-region boundaries for the parent RS. This degree of freedom will be referred to as the permutation of a PP. Let N be the number of braced TS's that comprise the PP. Since there are N! ways of arranging N things in a row, there are N! permutations for each PP. Since each realization must be planar, the N! permutations represent all the different ways that variation of the PP can produce distinct sets of g-region boundaries.

As was noted, RS's which are descendants of PP's need special consideration in the generation of realizations for the RS that is their ancestor. These descendant RS's will have realizations themselves, but they are limited by the fact that they are joined to another structure at their anchor and brace nodes. These points of attachment necessitate a special definition for the outside region of a braced RS. Strictly speaking, a braced RS has no outside region, since its attachment to another structure at two points breaks up the space outside of it into many regions, not just one. However, it is still possible to speak of a generalized outside region as if it existed: The outside region boundary of the RS is the circuit that when traced in the positive direction will bear all regions outside the RS to its right. The only limitation is that an outside region of a braced RS must contain the anchor and brace node of that RS in its boundary. If it did not, it would be possible to generate a realization of the braced RS that could not be connected to its parent structure without crossing edges in the plane.

Thus it would seem that given an RS from which realizations are to be generated, call it RS', one could first generate g-realizations of RS' and then produce conventional geometric realizations from these by generating permutations for all PP's which are descendants of RS' and generating realizations of all RS's which are descendants of RS', in any order, subject to the outside region constraint just mentioned. This is a rough description of what is actually done, with one main exception, as explained in the following paragraphs.

The Algorithm. -- To justify a realization generation algorithm of the type just described it must be shown that it is capable of producing all distinct sets of region boundaries for the abstract graph described by RS', and all possible outside regions for each distinct set of boundaries.

The algorithm just sketched does not quite satisfy these requirements, because the PP permutations and the realizations of the RS's which are part of the TS's of the PP's would be generated after the outside regions for the g-realizations of RS' were selected. The regions created inside of the PP permutations and the inside (finite) regions of the descendant RS realizations could not be outside regions of a conventional realization of RS' under this algorithm, and they should be. This problem can be remedied by selecting the outside regions for any RS realization after its c-regions have all been specified, rather than after only its g-regions have been specified. This leads to the realization algorithm explained below. It will be shown in subsequent paragraphs that this algorithm satisfies the requirements stated at the beginning of this subsection.

(1) First choose orientations for RS' and each of its descendant RS's. If there are K RS's, counting RS', this means that there are 2^K different combinations of RS orientations, since there are two possible orientations for each RS. There is one exception to

this rule. An RS with only two g-region boundaries can produce all of its g-realizations using either the orientation degree of freedom or the outside region selection degree of freedom alone. To use both of these would produce redundant realizations. Thus only a single orientation is generated for RS's with two g-regions.

(2) Next, for each combination of RS orientations, choose and implement permuations for all PP's that are descendants (direct or indirect) of RS'. (Notice that this means that permutations are generated for PP's that are descendants of RS's which are descendants of RS'.) The braced TS's that were part of these PP's become part of the boundaries of g-regions. Any RS's that were on the maps of these braced TS's are temporarily converted into single TS PP's, where the RS is the only structure on the map of the TS. No permutations are generated for single TS PP's. If there are L_i TS's that belong to the i^{th} PP, and if there are M PP's in all, there are $\prod_{i=1}^{M} (L_i!)$ different combinations of permutations.

(3) In the configurations generated by the above step, all braced RS's are in the form of single TS PP's, and each RS has a set of g-regions generated for it. At this point, generate outside g-regions for all braced RS's. Then merge all braced RS's with their great-grandparent RS's in any order (the parent structure of a braced RS is a TS, and its parent structure is a PP, but these two structures disappear after the merger). The next subsection will show that these two steps can be validly carried out. After

these steps are done, RS' is composed entirely of c-regions. If the j^{th} braced RS has P_j outside regions and there are Q braced RS's, then there are $\prod_{j=i}^{Q} P_j$ possible combinations of outside regions.

(4) Finally, if there are now N c-regions making up RS', each one of these can be the outside region, accounting for N different outside regions.

Thus if RS' has K-1 descendant braced RS's with three or more g-region boundaries, M PP's with L_i braced TS's comprising each PP, Q braced RS's with P_j outside g-regions each, and if RS' has a total of N c-regions then the total number of realizations for RS' is

$$R = 2^K \cdot N \cdot (\prod_{i=1}^{M} L_i\,!)\,(\prod_{j=1}^{Q} P_j)$$

For the region structure of Figure 2-9, the total number of realizations is calculated as

$$R = 2^2 \cdot 9 \cdot (1!3!)\,(1 \cdot 2) = 432.$$

This algorithm is justified in the following paragraphs, but one observation can be made here. If RS' is the main RS of the dual graph, its orientation is already chosen and it can have only one outside region. In this case, the algorithm gets suitably modified, as explained in Chapter 3.

Proof that the Algorithm Works. -- This proof can be divided into two parts.

I. First, it is proved that all different combinations of c-regions are generated for RS'. Consider the generation of permutations for the PP's which are direct descendants of RS'. It has already been observed that these produce all possible distinct sets of g-regions that can be due to PP variation. Next, consider the RS's that are direct descendants of the TS's that comprise these PP's. Realizations are generated for these RS's by following the same rules as for RS' (taking into account the special definition of an outside region). Thus, if the generation algorithm works for RS', it works for these RS's as well. Since these RS's are each attached to RS' at exactly two points in the second phase of the realization generation process, if all realizations for these RS's are generated they will produce all possible sets of c-regions for RS'.

II. The second part of the proof shows that all of the c-regions that are generated for RS' can be outside regions. This is guaranteed by the last step in the realization generation algorithm, provided that all realizations of the braced RS's that are part of the TS's that comprise the PP's that are part of RS' (call these the second level RS's) can indeed be generated. It has already been shown that all combinations of c-regions can be generated for these braced RS's by the algorithm. It remains to be shown that their outside regions can be generated correctly, since the special definition of an outside region for a braced RS makes it a separate case from RS'.

In the algorithm, only one difference exists between the way outside regions are generated for RS' and the way they are generated for the second level RS's. The difference is that in the first case the outside regions for the second level RS's must be generated before those for RS' are generated, while in the second case no such restriction exists on the braced RS's that descend from the second level RS's. Recall that in the case of RS' this restriction was introduced because the regions inside the PP permutations and the inside (finite) regions of the second level RS's had to be allowable outside regions for RS'. Now in the algorithm, the regions inside of PP permutations are considered as possible outside regions for the second level RS's since outside region selection takes place after PP permutations have been generated. Furthermore, it is easy to show that since the second level RS's and their descendant RS's are braced, none of the inside regions of their descendant RS's can be outside regions for the second level RS's. In a recursive manner, then, no inside region of a braced RS can be an outside region for an ancestor braced RS.

The proof is as follows: For an inside region of a descendant RS to be a possible outside region of an ancestor RS, it must contain both the anchor node and the brace node of the ancestor RS in its boundary. Since the descendant RS is attached to the ancestor RS at only two nodes, the anchor and brace nodes of the ancestor and descendant RS's would have to be one and the same pair. However, it was stated in the syntax of the PP that a braced RS that comprises the single structure of a braced TS can have only one region that has both the anchor and brace node in its boundary. This region must be the outside region of the braced RS, so no inside region can contain both the anchor node and the brace node. Since all braced RS's are the single structures of braced RS's when the outside regions are being selected in the realization generation algorithm, the proof is completed.

Hierarchical Structuring as an Abstraction. -- The hierarchical structuring of the dual graph that is achieved through the use of the PGG serves as a valuable aid to the design solution search. Just as the partially completed dual graph, with its missing adjacencies and its uncolored and/or undirected and/or unweighted edges serves as an abstraction of the floor plan, the PGG provides an abstraction of the dual graph. That is to say, many possible floor plans can be generated from a partially completed dual graph, and many topologies for a partially completed dual graph can be generated from a single PGG description. Thus the application of a single analytical test to a hierarchically structured dual graph provides results which apply to a whole class of potential design solutions. This enables one to search exhaustively a solution space that would be uneconomical to search if each potential solution had to be enumerated in its completed form before it could be tested. Allen Newell has stated that in the final analysis all problem solving programs resort to some sort of generate and test procedure. What differentiates one approach from another is the level at which the generate and test takes place. The ability of this representation to allow generate and test procedures that deal with whole classes of potential solutions rather than single solutions is one of its strongest points.

2.4.2. Physical Realizability

Overview. -- The ultimate goal of the design procedure described in this thesis is to produce a dual graph that corresponds to a physically realizable floor plan satisfying the various design requirements. This section continues the development and presentation of properties of the dual graph that insure that this goal can be achieved. Most of the properties don't consider the weights of edges, because much of the design is done with unweighted graphs. For unweighted edges, the notion of physical realizability is defined as follows:

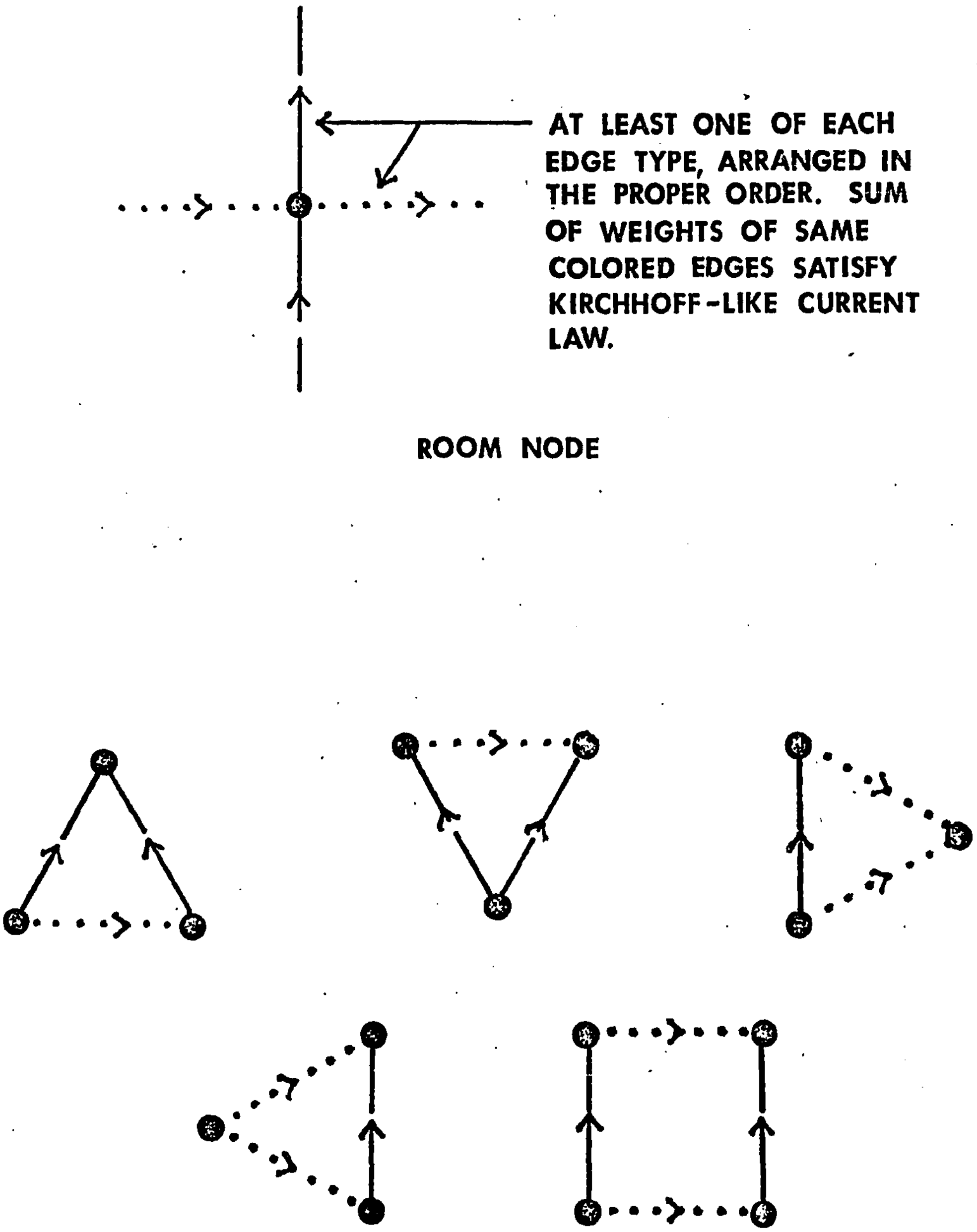
Definition.-- A subgraph of a dual graph (or in the limit, an entire dual graph) is physically realizable if it is possible to construct a set of non-overlapping rectangles of arbitrary size, corresponding to the nodes of the dual graph subgraph, that bear the contiguity relationships to one another indicated by the edges of the dual graph subgraph.

The properties developed in this section deal with those aspects of physical realizability that are a result of the use of rectangular rooms in the floor plan. Formation rules are given for nodes and regions of the dual graph. A special abstraction, the "turn" concept, is introduced to generalize these rules. At the end of the section several theorems are provided that give conditions that must be satisfied for the extension of a partially completed dual graph to be physically realizable.

Well-Formed Nodes and Terminal Regions. -- A few definitions will clarify the notion of well-formedness for the dual graph. A *terminal dual graph* is a dual graph that corresponds to a floor plan in which each room is contiguous to at least one other room and which completely fills a simply connected area, such as the floor plan of Figure 2-2. If the simply connected area is a rectangle, then the terminal dual graph is also a *completed dual graph*. A substructure of a dual graph is considered to be *well formed* if it will lead to a physically realizable floor plan. Two well-formedness rules are as follows:

(W1) Well-formed room nodes -- since a retangular room has four walls, it is necessary in a completed dual graph, that a node corresponding to a room have connected to it at least one each of the four edge types: slashed directed outward, dotted directed outward, slashed directed inward, and dotted directed inward, taken in clockwise order as shown in Figure 2-12. If the physical dimensions of the room are known, then for a node the sum of the weights of the slashed-inward edges must equal the sum of the weights of the slashed-outward edges, and this must equal the length of the room. A similar condition holds for the dotted edges. Any partially completed room node must not have its edges and/or weights arranged in such a way that the above conditions cannot be satisfied when the node is completed.

(W2) Well-formed terminal regions -- There are five types of *terminal regions* possible in a terminal dual graph,



AT LEAST ONE OF EACH EDGE TYPE, ARRANGED IN THE PROPER ORDER. SUM OF WEIGHTS OF SAME COLORED EDGES SATISFY KIRCHHOFF-LIKE CURRENT LAW.

ROOM NODE

THE TERMINAL REGIONS

FIGURE 2-12

WELL-FORMED NODES AND REGIONS

corresponding to the five different types of corners possible in a floor plan graph which fills a simply connected area. The terminal regions are shown in Figure 2-12. (Well-formedness rules for non-terminal regions are given later in this section.)

The four outside space nodes, $\mathcal{N}$, $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{W}$, require special treatment with respect to well-formedness, since they are only incidental to edges that attach them to one another and to the nodes of rooms along the outside walls of a building.

The "Turn" Concept. -- In order to reduce greatly the number of cases that have to be enumerated in stating and proving the theorems which follow, the concept of a turn is now introduced. Imagine the two types of directed edges of the dual graph to be in their respective north-south (for slashed edges) and east-west (for dotted edges) orientations, as seen in the first part of Figure 2-12. Then if one imagines himself walking along a chain, in going from one edge type to another while passing through a node, it is possible to think of making either a 0°, $\pm 90^{\circ}$, or $\pm 180^{\circ}$ turn, according to the turn matrix shown in Figure 2-13. Since the nodes are thought of as eventually being mappable into rectangles, it is seen that the $+180^{\circ}$ turn is an "about face" to the left. Similarly the -180° turn is an "about face" to the right.

In terms of the turn concept, properties W1 and W2 can now be stated more generally as W1* and W2*.

SECOND EDGE FIRST EDGE	TIP OF SLASHED	TAIL OF SLASHED	TIP OF DOTTED	TAIL OF DOTTED
TIP OF SLASHED	±180°	0°	+90°	-90°
TAIL OF SLASHED	0°	±180°	-90°	+90°
TIP OF DOTTED	-90°	+90°	±180°	0°
TAIL OF DOTTED	+90°	-90°	0°	±180°

FIGURE 2-13

THE TURN MATRIX

(W1*) Generalized well formed nodes -- It is possible to look at the edges connected to a node and determine whether property W1 is satisfied with respect to unweighted, uncolored, undirected edges, solely by inspecting the turns between adjacent edges. Consider Figure 2-12 A. The turns indicated are -180° in walking along edge a through the node to edge b, -90° in walking from edge b to edge c, and -90° in walking from edge a to edge c. Now imagine walking from edge a to edge c by way of edge b. One starts by making a -180° turn from edge a to edge b. But then, to get headed back to edge c, one makes a virtual +180° turn at the end of edge b, causing one to reverse direction and walk back along edge b. Finally, one makes a -90° turn from edge b to edge c. Notice that the sum of these three turns (-180°, +180°, and -90°) exactly equals the -90° turn from edge a to edge c. This property of the sum of turns (with a virtual turn for each intervening edge) being equal will hold for all well formed nodes, regardless of whether they are partially or completely specified. Of course, if turns are ever traced in the reverse direction, their signs are changed. Furthermore, edges that are encountered while tracing edges in the clockwise direction around a node require -180° virtual turns. In Figure 2-14 B for example, the four +90° turns minus three -180° virtual turns gives the necessary -180° turn that it takes to start at the end of edge a, walk toward the node, walk out and back on every edge in succession, and finally wind up walking away from the node on edge a again.

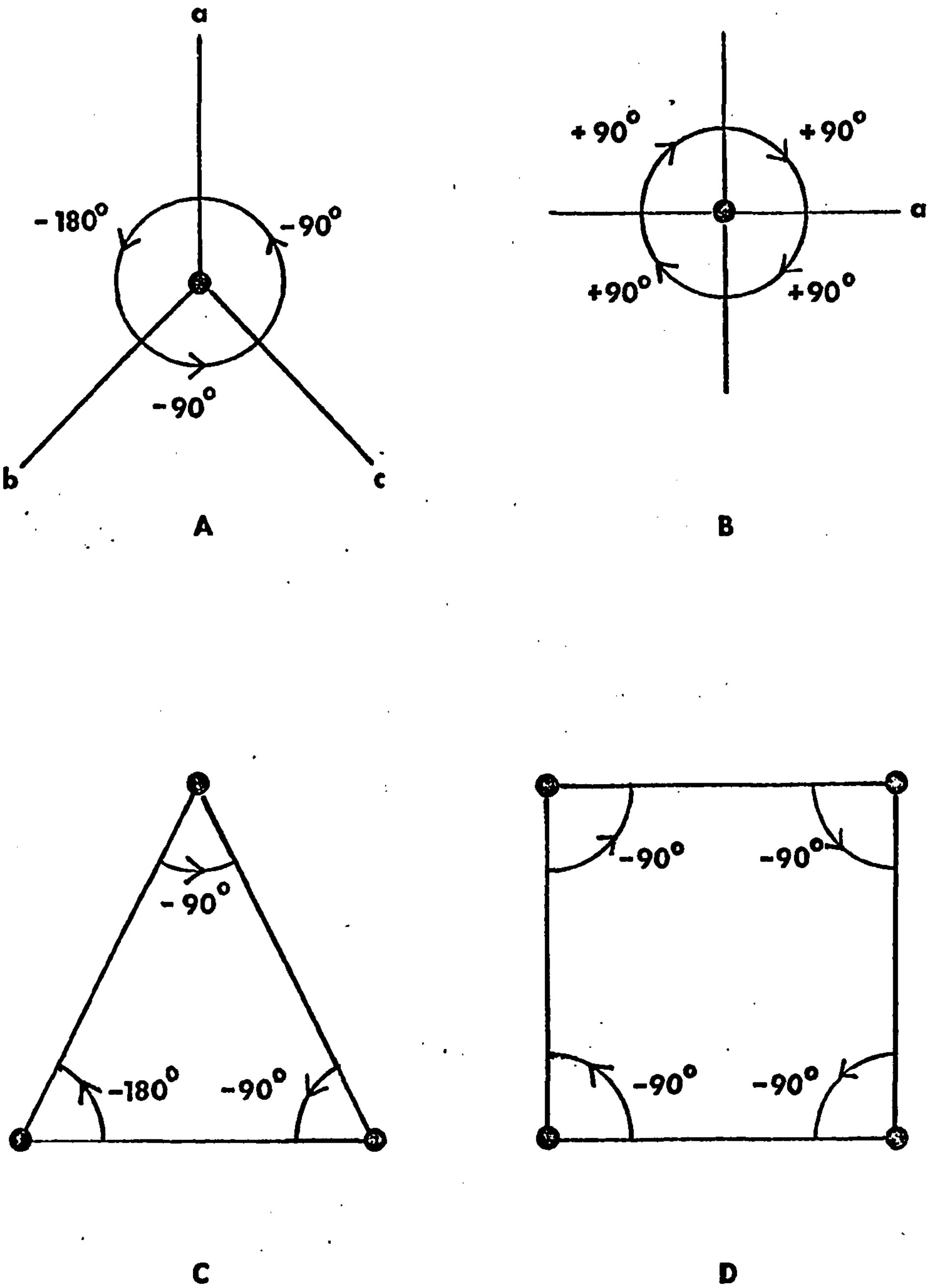


FIGURE 2-14

EXAMPLES FOR PROPERTIES W1* AND W2*

(W2*) Generalized well formed terminal regions -- Speaking in terms of unweighted, uncolored, undirected edges and turns, the four well-formed three sided terminal regions reduce to the generalized one shown in Figure 2-14-C, and the well-formed four sided terminal region becomes that shown in Figure 2-14-D.

A note should be added here about three-sided terminal regions. Any three sided region in a dual graph realization must in fact be treated as terminal, even if the color and direction of its edges have yet to be specified. This is because any time three rectangular spaces meet in the plane to form a triangular dual graph region, their adjacencies must form some sort of "T" shaped corner. Consequently, no structures can be placed inside of a triangular region in a dual graph realization.

Three Theorems on Physical Realizability -- The two well-formedness rules just given deal with single nodes and terminal regions. However, often in solving design problems, chains and non-terminal regions (regions with four or more edges in their boundary) occur as subcomponents of a partially completed dual graph. The three theorems in this section deal with the physical realizability of such structures. But first, a definition is necessary:

Definition -- A "±180° (±190°) ±180° sequence" is a sequence of turns consisting of either two -180° turns with an arbitrary number of -90° turns between (including possibly no -90° turns) or two +180° turns with an arbitrary number of +90° turns between (including possibly none).

Theorem 1:

A necessary and sufficient condition for an unweighted isolated dual graph chain to be physically realizable is that it contain no $\pm 180^\circ$ ($\pm 90^\circ$) $\pm 180^\circ$ sequence.

The proof of this theorem is given in Appendix A.1.1.

Definition -- A non-terminal region bounded by unweighted edges will be called fillable if its interior can be entirely filled with well-formed three sided terminal regions, without the addition of any new nodes and such that the resultant dual graph is physically realizable.

Theorem 2:

A non-terminal region bounded by unweighted edges is fillable if and only if the sum of the turns of its boundary circuit taken in the clockwise direction is -360° and the circuit contains no $\pm 180^\circ$ ($\pm 90^\circ$) $\pm 180^\circ$ sequences.

The proof of this theorem is given in Appendix A.1.2.

Theorem 3:

A circuit of unweighted edges is physically realizable if and only if the sum of its turns taken in the clockwise direction is -360° and it contains no $\pm 180^\circ$ ($\pm 90^\circ$) $\pm 180^\circ$ sequences.

Th. proof of this theorem is given in Appendix A.1.3.

Corollary 3.1:

The region corresponding to a physically realizable circuit of unweighted edges is fillable.

The proof of this corollary is given in Appendix A.1.4.

Theorems 1 and 3 provide the necessary well-formedness tests for chains and non-terminal regions in a partially completed dual graph. Once some of the edges have weights attached to them, the theorems provide necessary but not sufficient conditions for physical realizability. Some tests for physical realizability involving weighted edges are discussed in Section 3.3.3.

Besides accomplishing their designated purpose, the proofs of the above theorems provide insight into the dual graph representation and are recommended to the interested reader. Section 3.3.2 draws upon these proofs to justify some of the design tree search techniques.

2.5. Comments on Krejcirik's Work

As was mentioned earlier, the work of M. Krejcirik represents the closest approach that the author has been able to find to the dual graph representation. The author feels that his own work has carried the theory of the dual graph representation much farther than Krejcirik's work. This is substantiated in the following paragraphs, in the course of giving a rough description of Krejcirik's approach.

Krejcirik uses the dual graph in two separate design programs. The first one, called the RG program, is chiefly concerned with the topology of the floor plan. The second one, called the RR program, attempts to fit room dimension requirements into a fixed topology as specified by a dual graph. The reader can see in Chapter 3 how this approach compares to the one used in GRAMPA.

The input to the RG program is a list of room nodes and a list of adjacency requirements. These requirements are weighted according to the relative desirability of each adjacency. The direction of an adjacency (north, south, east,or west) may not be specified as it can in GRAMPA. Nevertheless, in RG it may be stated if a room is to be adjacent to one or more of the outside walls of the building. As in the case of GRAMPA, rooms and buildings are assumed to be rectangular.

The output of RG is a dual graph consisting entirely of three sided terminal regions, but in which the edges are neither colored, directed, nor assigned dimension weights. It can be shown, using the theory in this thesis, that such a dual graph may not always be physically realizable. Furthermore, there exist dual graphs which do not fit into this class which are physically realizable -- namely dual graphs which include four sided terminal regions. Instead of having outside space nodes to establish the orientation of the floor plan, Krejcirik uses the four corner rooms (the northwest, northeast, southeast, and southwest) to establish the "cardinal points" of the floor plan. (An example of such a dual graph is shown in Figure 2-15.) This has some advantages and some disadvantages compared with the use of outside space nodes, and it is not clear which method is best. It is interesting, however, that in the RR program Krejcirik switches to the use of outside space nodes.

The dual graph which is the output of the RG program is chosen to maximize the total weight of the adjacencies which are satisfied. It is not clear from Krejcirik's article just how this is done, but it is stated that the four corner rooms are chosen first, and then the interior of the graph is filled in. Highly weighted adjacencies are preferentially satisfied. The output graph is described by enumerating its three sided regions. Apparently, from the article, the graph specifications are not always unique and the user may have to make some arbitrary

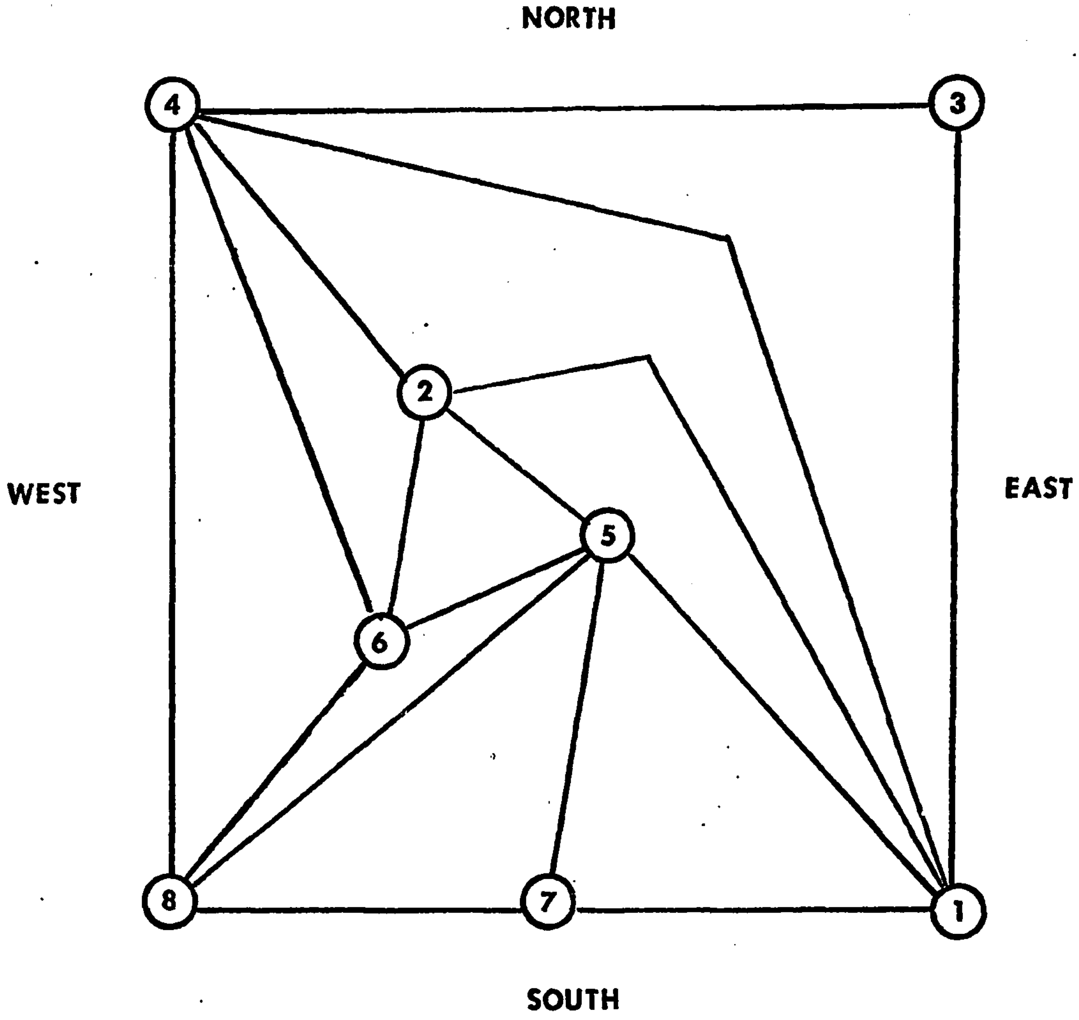


FIGURE 2-15

A DUAL GRAPH FROM THE RG PROGRAM

decisions in order to translate the region descriptions into a drawing of a graph.

In contrast to the RG program, GRAMPA satisfies a set of discrete unweighted adjacencies in the first phase of the design process. In general it will produce a partially completed dual graph, rather than one composed entirely of terminal regions as RG does. However, rather than leave the ambiguity of the partially completed graph up to the user, GRAMPA is capable of generating all geometric realizations for a given graph.

Krejcirik's second program is the RR program, which is intended to assign dimensions to the rooms of a dual graph which is topologically completely specified. Thus, one of the inputs to RR is a dual graph composed entirely of three sided terminal regions with colored and directed edges. In contrast to GRAMPA, the colors and directions of the edges must be introduced by the user rather than by the program. For some reason, Krejcirik chooses to treat the dual graph as two separate graphs -- one for the east-west adjacencies and one for the north-south adjacencies. The author views this as a disadvantage unless the interactions between the two graphs are taken into account.

In addition to the dual graph, the second input to the RR program is a list of area requirements for the various rooms, along with maximum and minimum dimension limits. This set of requirements is more general than the one used in GRAMPA, but it is possible to extend GRAMPA to cover such requirements, as explained in Chapter 5. Furthermore, RR does not do a complete job of satisfying these requirements anyway, as can be seen by observing a typical output. Such an output is shown in Figure 2-16[13], where it is seen that only some of the adjacencies of the topologically completely specified dual graph which was input have been maintained, thus leaving large holes in the floor plan. Contrast this with GRAMPA, which produces completely filled floor plans.

Again it is not clear from Krejcirik's article, but the RR program apparently satisfies the dimension requirements by some sort of iterative searching. Rudimentary well formedness tests are explained, but they are not as complete as those of this thesis, especially in the area of topological physical realizability.

Nevertheless, Krejcirik's programs clearly indicate some possible directions for the generalization of GRAMPA. The use of weighted adjacency requirements and the more general dimension requirements are samples of these generalizations. However, Krejcirik's programs proceeded to the more general types of design requirements without first thoroughly developing the theory of the dual graph representation. The emphasis of this thesis on the other hand has been to develop theory first and to write a package of program subroutines capable of handling a basic set of design requirements. This basic package can easily be expanded to treat a more ambitious class of problems, as discussed in Chapter 5.

[13] This Figure and Figure 2-15 were redrawn from Krejcirik"s article, Ref. 16.

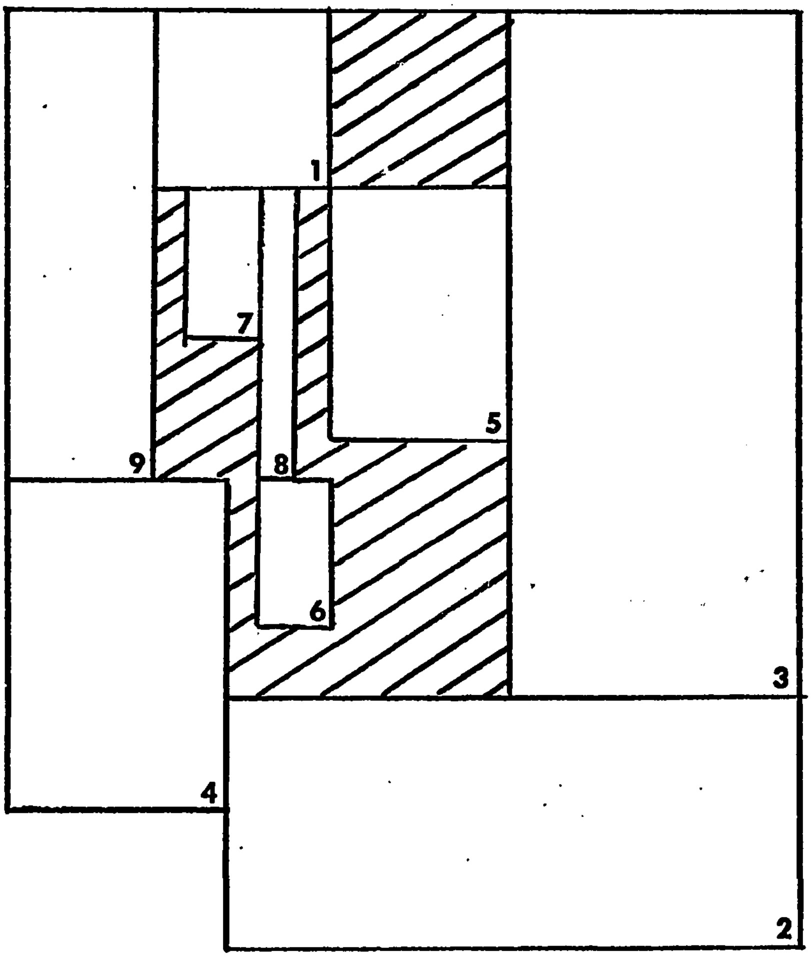


FIGURE 2-16

AN OUTPUT FROM THE RR PROGRAM

2.6 The Realization of a Linear Graph From Its Circuit Matrix

When linear graphs are used in systems design, an intermediate stage in the design process is often the circuit matrix (see Section 2.4.1.3, Overview) of a graph. It must be determined if this matrix is realizable as a linear graph (not necessarily a planar linear graph). Once realizability is established, the next task is to actually construct a realization of the graph.

A technique for constructing a realization of a linear graph, given its circuit matrix, is given by Auslander and Trent.[15] The problem which they deal with differs from the problem of generating realizations for planar graphs in several ways:

(1) The circuit matrix does not uniquely specify all edge-node incidences, while the PGG does. This is because the circuit matrix specifies only which edges are in which circuits, but not the order of the edges. This is intentional, of course, for the circuit matrix may be representing a network in which the serial order of the elements in a given branch is immaterial.

(2) The graph specified by the circuit matrix is not necessarily planar, while the graph specified by the PGG is.

(3) Auslander and Trent are interested in constructing only one realization from a circuit matrix, while the generation algorithm of this thesis must generate all

[15] See Ref. 7.

realizations of a planar graph.

In spite of these differences, with the possible exception of (3), Auslander and Trent's technique is adaptable to the generation of planar graph realizations. The PGG and the generation algorithm of this thesis, however, provide several advantages for computer implementation, over the circuit matrix and the construction technique of Auslander and Trent. Before discussing these advantages, that construction technique is briefly outlined.

In principle, the technique is quite similar to the one used in this thesis, the main difference being that it operates off of a circuit matrix representation instead of the PGG. A theorem, called the Decomposition Theorem, is used to divide the graph hierarchically into a set of subgraphs that strongly resemble the RS's of the PGG. Special matrices, called reduced incidence matrices, are used to store the information needed to reconstruct a graph from its subgraphs, after their realizations have been constructed. These matrices contain generalized edges corresponding to the subgraphs that were produced, just like the RS's can be generalized in the PGG.

The subgraphs are then reduced to their simplest form before realizations are generated for them. This is done by recognizing degrees of freedom (nonuniquenesses), such as parallel branches, and nonspecified serial order of edges. For instance, parallel branches are replaced by a single generalized edge, much as PP's are used in the PGG.

After the simplification has been done, the realization of the simplified subgraph is constructed and the realizations of the various nonuniquenesses are introduced one at a time. This results in a sort of realization for the subgraph. The words "sort of" are used because, when the subgraphs are combined into the final realization of the original graph (this is the final step in the construction process), the subgraph realizations get altered somewhat. This recombination of the subgraphs is guided by the reduced incidence matrices, mentioned above.

Now the advantage of the methods used in this thesis, for computer implementation, can be listed.

(1) The PGG has built into it, in the form of the nonterminal structures with their degrees of freedom, most of the information that the realization generation algorithm needs to operate. With the circuit matrix representation, the construction algorithm must generate this information.

(2) The PGG is especially organized to facilitate the planarity test. Planarity tests using the circuit matrix would be generally more lengthy.

(3) The subgraph realizations generated in the Auslander and Trent technique do not recombine into the final graph in as simple a manner as they do in the method used in this thesis.

In summary, the Auslander and Trent technique is interesting because of its similarity to the methods used in this thesis, but the PGG seems to be a more advantageous representation than the circuit matrix for computer implementation.

3. THE DESIGN ACTIVITIES AND THE DECISION SCHEME

3.0 Introduction

The purpose of this chapter is: (1) to describe the design method for floor plans, based on the dual graph representation (this is the decision scheme of the paradigm discussed in Section 1.4), and (2) to describe the computer program, GRAMPA, that was written to implement this method. This program is organized according to design activities, as discussed in Section 1.4.

For the sake of brevity, the design method is described in terms of its program implementation. The reader should not let this obscure the fact that the design method exists separately from the program, and GRAMPA is but one of several possible implementations of the method.

GRAMPA itself consists of approximately 12,000 instructions written in the computer language IPL-V, as implemented on the IBM 360/67 computer at Carnegie-Mellon University. IPL-V[1] is a fixed format list processing language that has primitives that greatly resemble assembly code. This accounts for the large number of instructions.

The input to GRAMPA consists of: (1) a list of the rooms that the desired floor plan should have, along with dimension requirements for these rooms, and (2) a list of adjacency requirements for

[1] For further details on IPL-V, see Information Processing Language-V Manual by Allen Newell et al., second edition, Prentice-Hall, 1964.

the rooms. The individual requirements take the following form (see Section 2.1):

(1) For each room, a list of allowable sizes is specified. For instance, room X1 may be required to be either 3 x 3, 3 x 2, or 3 x 1 units in size (both orientations of a given size are considered by the program).

(2) One room may be required to be next to either another room or one of the outside walls of the building. If desired, one room may be required to be north, south, east or west of another room.

It is possible to extend this set of design requirements to include such things as overall area and overall dimensions for the building. Thesε extensions are discussed in Chapter 4.

Once GRAMPA has read the design requirements, its operations can be divided into two phases. The design activities associated with these two phases are listed below:

(1) Phase I deals solely with topological considerations and includes:

(a) determination of graph planarity

(b) satisfaction of adjacency requirements

(c) generation of dual graph realizations

(d) tests of well-formedness for nodes and regions

(2) Phase II deals with both topological and dimensional considerations and includes:

(a) the decision scheme for the topological and dimensional completion of dual graph realizations

(b) satisfaction of dimension requirements

(c) local tests of dimensional well-formedness

(d) global tests of dimensional well-formedness

(e) b and d of Phase I

Briefly speaking, Phase I of the program creates a dual graph in which all of the adjacency requirements are satisfied. This graph represents a partially completed design solution. If the adjacency requirements lead to a contradiction (a non-planar graph) no design solution exists and the program terminates. If the dual graph is planar, Phase I then generates realizations of this graph for Phase II.

Phase II of the program has two jobs: (1) it must satisfy the dimension requirements, and (2) it must fill in the adjacencies of the dual graph which were not specified by the design requirements. If these jobs can be carried out successfully, a completely specified floor plan is produced and is printed out in coded form.

A complete documentation of GRAMPA would take over 200 pages. However, Appendix A.2. contains paraphrased flow charts for all major routines. These flow charts should be consulted frequently while reading this chapter.

3.1 Feasibility

The description of the program centers around the feasibility of a design solution. The notion is developed as follows:

For this program a final solution consists of the completely specified dual graph corresponding to an actual floor plan, or the statement that there is no floor plan which satisfies the given requirements. In general, the program progresses toward a final solution by slowly adding information to a partially completed solution in response to both specific design requirements and certain rules for adding those details not explicitly requested by the requirements. The search for a final solution is considered to be proceeding in a forward direction as long as the solution space which remains to be searched grows progressively smaller. If some sort of tree searching is used by the program, then the list of explicit design requirements which have already been considered at a given stage in the solution search may fluctuate up or down as various combinations of structures are tried to fill in the remaining detail of the design. However, the search is proceeding in a forward direction in this case.

At a given stage in the search for the solution of a design problem, the partially completed solution is considered feasible if:

(1) it satsifies all explicit design requirements considered up to that point in the search, and

(2) from it can be generated at least one physically realizable floor plan.

The version of GRAMPA described in this thesis is set up to search the design solution space exhaustively[2]. In terms of feasibility, this means that:

(1) no infeasible final solutions are generated, and

[2] See Section 3.3.2

(2) all feasible final solutions are generated.

Thus, in explaining the program, it must be shown that these two conditions hold. It would be ideal if in addition to the final solutions, all partially completed solutions generated by the program were feasible too. This goal is not quite achieved by the program, but efforts are taken to minimize the number of infeasible partially completed solutions.

3.2 Phase I of the program (Topological)

3.2.0 Introduction

Phase I handles all topological considerations up to and including the generation of dual graph realizations. This includes routines A8, S1, and R150 up to the execution of S2. The general flow of Phase I is shown in Figure 3-1. Notice that routine A8 satisfies the adjacency design requirements, first testing each one against the planarity requirement of the dual graph. Routine S1 then generates dual graph realizations for Phase II of the program. Routines S1 and R150, and several of the generators used by R150 are written recursively. This is to allow the push-down storage capabilities of IPL-V to guide the tree search automatically. For more detail on those aspects of Phase I see the paraphrased flow diagrams in Appendix A.2.

The following two theorems concern the feasibility of partially completed design solutions in Phase I, as discussed in Section 3.1:

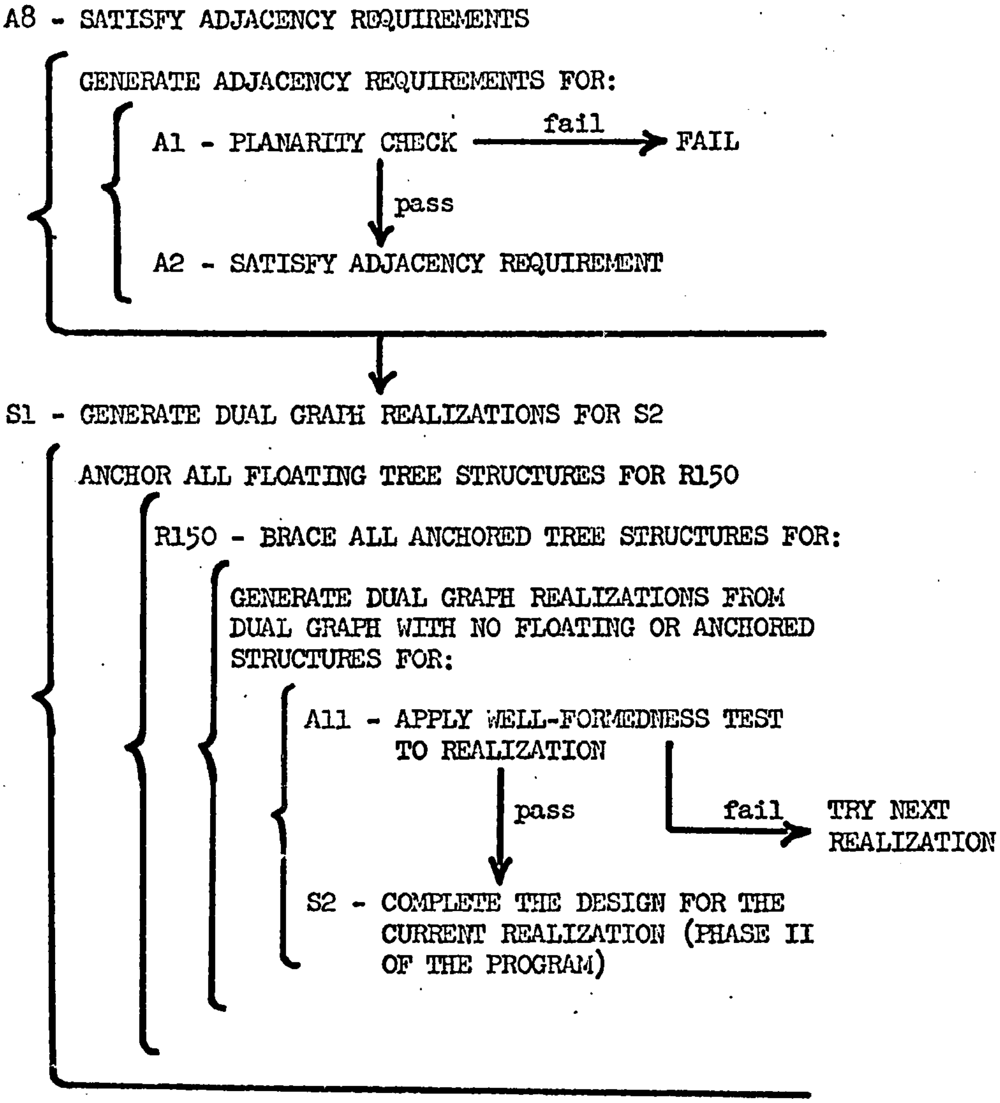

A8 - SATISFY ADJACENCY REQUIREMENTS
GENERATE ADJACENCY REQUIREMENTS FOR:
A1 - PLANARITY CHECK
fail
FAIL
pass
A2 - SATISFY ADJACENCY REQUIREMENT
S1 - GENERATE DUAL GRAPH REALIZATIONS FOR S2
ANCHOR ALL FLOATING TREE STRUCTURES FOR R150
R150 - BRACE ALL ANCHORED TREE STRUCTURES FOR:
GENERATE DUAL GRAPH REALIZATIONS FROM DUAL GRAPH WITH NO FLOATING OR ANCHORED STRUCTURES FOR:
A11 - APPLY WELL-FORMEDNESS TEST TO REALIZATION
pass
fail
TRY NEXT REALIZATION
S2 - COMPLETE THE DESIGN FOR THE CURRENT REALIZATION (PHASE II OF THE PROGRAM)

FIGURE 3-1

GENERAL FLOW OF PHASE I

Theorem F1:

Three necessary conditions foɾ Phase I of the program to generate no infeasible partially completed design solutions are:

(a) all adjacency requirements which have been considered must be satisfied.

(b) the dual graph must be planar and simple.

(c) all nodes and regions of the dual graph must be well-formed.

The proof of this theorem is given in Appendix A1.5. Since Phase I never generates final design solutions, no sufficiency conditions for feasibility are required at this time. However, it will be shown that the program attempts to minimize the number of infeasible partial solutions that leave Phase I.

Theorem F2:

Three necessary and sufficient conditions to guarantee that Phase I of the program generates all feasible partially completed design solutions are:

(a) no planar graph may be rejected by the planarity check, and it must be impossible to construct a planar graph by adding structures to a nonplanar graph.

(b) all planar realizations of the dual graph must be generated.

(c) no well-formed node or region may be rejected by the node and region checks, and it must be impossible to construct a well formed node or region by adding structures to one that is not well formed.

This theorem is proved in Appendix A1.6. It must be shown that Phase I of the program satisfies the conditions of these two theorems. This will be done in the course of explaining the main routines of Phase I. Conditions Fl-b and F2-a are covered in Sections 3.2.1 and 3.2.2; Fl-a is covered in 3.2.2; F2-b is covered in 3.2.3; and Fl-c and F2-c are covered in 3.2.4. Attention is drawn to a condition in these sections only if it is not obvious that it is satisfied.

3.2.1 The Planarity Routine

3.2.1.0 Introduction

Overview. - The planarity routine is used to test whether a partially completed dual graph is planar. This introductory section is used to discuss some basic issues related to the routine. The main point of these is that the planarity routine works in an inductive manner. It assumes that a given graph is planar, and then tests to see if it will remain planar after the addition of some edge desired by the program.

Some Basic Issues. - An introductory discussion of the planarity of the dual graph was given in Section 2.4.1. Several methods exist for determining whether or not a given graph is planar (see Refs. 8 and 10). However, two facts influence the manner in which this program treats planarity;

(1) Because the dual graph is constructed one edge at a time, the planarity question is phrased as: "Given that graph G_N of N edges is planar, if edge E is added to this graph will the resultant graph of N+1 edges, G_{N+1}, be planar?"

(2) The answer to the question in (1) is facilitated because the dual graph has already been structured hierarchically to keep track of planar realizations.

The question arises as to what happens if G_{N+1} is not planar. Is it possible that some graph G_{N+1+M} built by adding M edges to G_{N+1} will be planar? This is clearly not possible, because of a theorem of the planarity of linear graphs due to Kuratowski[3]. A few definitions are necessary before stating the theorem.

(1) The Kuratowski subgraphs, types I and II are shown in Figure 3-2.

(2) Suppose two graphs, A and B, are altered so that in both graphs, any chain composed of nodes having only two edges incident to them is replaced by a single edge. If A and B are isomorphic in this altered form, they are said to be isomorphic to within vertices of degree two.

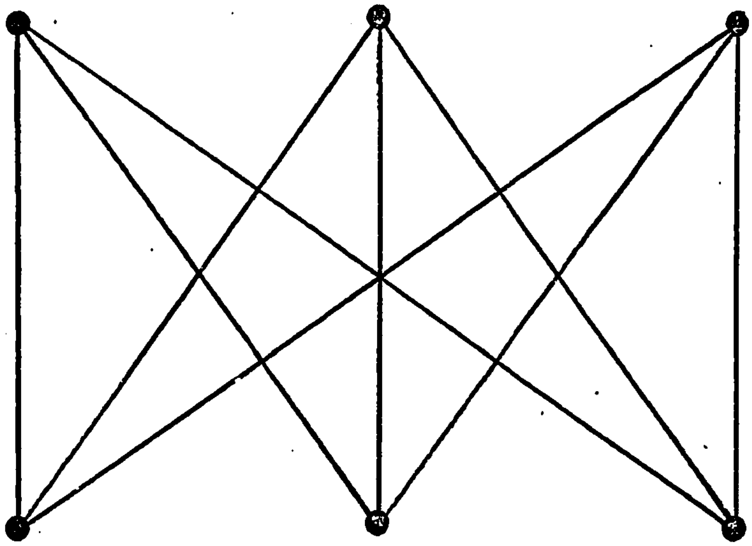
Theorem K;

A graph is planar if it does not contain any subgraph which is isomorphic, to within vertices of degree two, to either of the Kuratowski subgraphs.

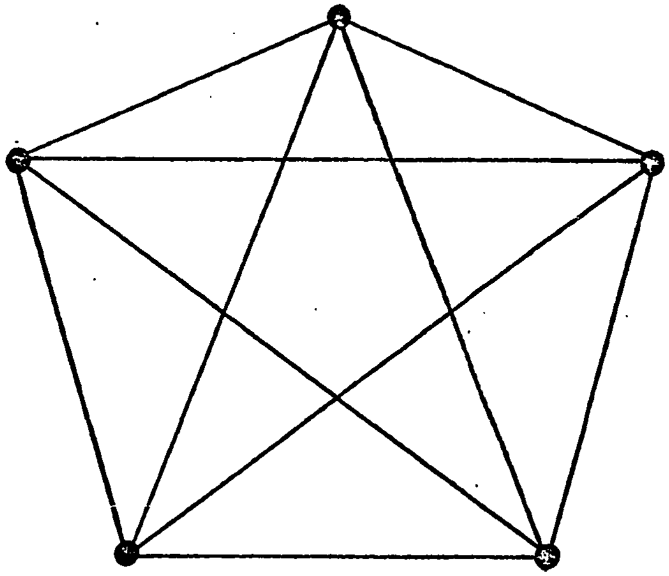
Once a graph contains a subgraph which is isomorphic, to within vertices of degree two, to a Kuratowski subgraph, it is impossible to make it planar again without removing one of the edges of the subgraph. Hence, G_{N+1+M} cannot be planar. This satisfies the second part of condition F2-a.

Combining the above considerations with the definition of planarity, the planarity question can be rephrased as: "Given that the

[3] See Ref. 10, p. 70



I.



II.

FIGURE 3-2

THE KURATOWSKI SUBGRAPHS

hierarchically structured graph G_N of N edges is planar, if edge E is to be added to this graph, does there exist a realization of G_N such that E can be added in a planar way, i.e. so that E crosses no edges of G_N?" This is the question that the program asks, and if the answer is yes it goes ahead and adds the edge to the graph. Otherwise, the required adjacency is considered unrealizable. An objective criterion for "adding E in a planar way" is needed, however. This is given in the following subsection.

The Fundamental Planarity Test. - A necessary and sufficient condition for E to be added to G_N in a planar way is that there exist a realization of G_N in which the following condition is satisfied:

> Condition P1: Both endpoint nodes of E must either lie within the same region, or they must border at least one region in common.

Clearly, if this condition is satisfied the graph G_{N+1} will be planar, and if this condition is not satisfied, then there is no way to add E to G_N without crossing an edge of G_N and G_{N+1} will not be planar.

The planarity test routine must be able to determine whether or not condition P1 is satisfied within the framework of the hierarchical structuring of the graph, provided that it can be shown that any planar graph G_N can be described hierarchically. The latter is established in Section 3.2.2. The former is best illustrated by explaining the flowcharts of the planarity routine.

3.2.1.1. The Planarity Routine Flowcharts

3.2.1.1.0. Introduction

In the planarity routine the hierarchical structuring of the dual graph achieved by the PGG is used to great advantage. The various subgraphs that constitute the non-terminal structures are known to be planar. When one node of a known planar graph is to be connected to another node of the same graph by the addition of an edge, the planarity routine checks through the hierarchy of planar subgraphs to see if this addition can be made in a planar way. Recalling condition P1, this means that the subgraphs are twisted and rotated to try to manipulate the graph into a form in which the two designated nodes lie within or border the same region. Please note, however, that this twisting and rotating is done on the abstract graph represented by the PGG. Actual geometric realizations of the graph are not produced by the planarity routine.

Chart F3 shows routine A1, the executive routine for the planarity test. First R53 is applied to test if an edge already exists between N1 and N2, the two nodes to be connected. This and routine R23, applied two steps later to test if N1 = N2, insure that the dual graph will remain simple (see Section 2.4.1). This satisfies the second part of condition F1-b.

After R53 is applied, the parent structure hierarchy lists (abbreviated PSL) of N1 and N2 are constructed. The parent structure hierarchy list of a structure is a list consisting of the structure itself preceded by its parent structure (if any), preceded by its parent structure's parent structure (if any), and so on, with the top structure on the list being the first one in the hierarchy to have no parent structure (see Chart F4, routine R22). A pair of PSL's is considered to be truncated if the top structure of one is identical to the top structure of the other and if they share no other structures in common. The convention will be adopted that a structure is below its parent structure in the hierarchy. Thus, if the first structures on the truncated PSL's of N1 and N2 are identical, that structure represents their lowest common ancestor in the hierarchy.

Depending on whether the PSL's of N1 and N2 have a common ancestor, different action is taken by A1.

3.2.1.1.1. No Common Ancestor on PSL

If there is no common ancestor, this means that N1 and N2 belong to disconnected subgraphs of the dual graph. Thus the edge that is to connect N1 and N2 could not be part of a subgraph, isomorphic to within vertices of degree two, to a Kuratowski subgraph, since it will be the only edge connecting the two subgraphs. Thus, if the two subgraphs are planar, the new graph with E added will also be planar.

Note that it was unnecessary to call upon condition P1 here to establish the planarity of the resultant graph.

3.2.1.1.2. Common Ancestor on PSL

If there is a common ancestor, routine R28 is applied. R28 does use condition P1. The reader will recall the realization generation algorithm for region structures (see Section 2.4.1). The planarity test requires that it be shown whether or not a realization of G_N exists in which condition P1 is satisfied. This does not mean that a complete realization actually has to be exhibited, however. Many of the substructures of G_N that have degrees of freedom with respect to realizations may not be directly involved with the potential planarity of G_{N+1}. In particular, if a given pivot pair set does not appear on the PSL of either N1 or N2, it might as well be an edge as far as planarity is concerned, and it is sufficient to recognize that a planar realization exists for it, without generating one. The same is true of any region structure which is not the PSL of N1 or N2 and which has none of the tree structures on either PSL as its parent structure[4].

Thus, only part of the graph G_N need be considered by the planarity test. This means that in explaining the test, part of the dual graph will be considered as if it were a realization, and part will still be considered as if it were hierarchically structured. Since the terms g-region and c-region were used in Chapter 2 to distinguish between the formation rules for hierarchically structured region structures and their realizations, some confusion could result if both terms were used here. Thus, in the following sections only the term "region" is used, since under the mixed formation rules neither of the other terms is exactly correct. Consider two definitions:

Definition - The first structure on a PSL (the highest structure in the hierarchy) will be called the base structure of that PSL.

Definition - The second structure on a PSL (the second highest structure in the hierarchy), or the anchor node of that structure if it is an anchored tree structure, will be called the takeoff structure (TOS) of that PSL.

Quite generally speaking, in order to show that a realization of

[4] This illustrates one of the prime motivations for selecting the particular hierarchical structuring which is used.

G_N exists which satisfies condition P1, the following must be shown:

(P2) These exists a realization of the base structure of the PSL's of N1 and N2 (PSL1 and PSL2) in which the TOS's of the two PSL's either border or exist within at least one common region, and

(P3) the structures of PSL1 and PSL2 can be unfolded in a planar way so that N1 and N2 border or exist within the same region as their respective TOS's.

These two conditions will be made more explicit when the three possible types of base structures are discussed. What they basically say, however, is that since N1 and N2 are connected to their respective TOS's through a continuous hierarchy of connected graph structures, if TOS1 and TOS2 do not border or exist within one common region, then neither can N1 or N2 without an edge of G_N being crossed. If TOS1 and TOS2 do border or exist within a common region, then the only way G_{N+1} will be planar is if N1 and N2 also border or exist within that same region, else again an edge of G_N must be crossed to connect N1 and N2.

Thus, the planarity test for the common ancestor case can be divided into two separate parts, that for condition P2 and that for condition P3. Two routines are used in common by these two parts. They are explained in the following paragraphs.

The Planar Unfolding of a Tree Structure. - It can be deduced from the planar graph grammar (Section 2.4.1) that if an anchored or a floating tree structure contains region structures on its map, it may not be possible to generate a realization of that tree structure in which all of the structures on its map lie within the same region (i.e. a realization in which the TS is "stretched out"). This is the case if the two nodes connecting an RS to the TS map do not border a common region of the RS, since one or the other end of the TS map proceeding away from the RS could not lie "outside" of the RS.

Routine +33 (see Chart F8) determines whether a given tree structure can be unfolded in a planar fashion from one of its map structures, S1, to another, S2. In other words, +33 determines whether a realization of the TS exists in which S1 and S2 lie within the same region. It does this simply by testing each RS between S1 and S2 on the TS map and determining whether the two nodes connecting the RS to the TS map share a common region of the TS. If S1 or S2 itself is an RS the net result is that the point of attachment of the PSL hierarchy to the RS is tested to see if it shares a region with the point of attachment of the RS to the TS map. +33 usually performs this test, but if the RS under consideration is the farther of S1 and S2 from the anchor node of the TS, this test is deferred to routine R27.

The Planar Unfolding of a PSL. - The testing of condition P3 is accomplished by routine R27 (see Chart F6). R27 simply considers the structures of a PSL a pair at a time (a pair being a structure and its succeeding structure on the PSL) and determines in each case whether that pair can be unfolded in a planar way. Since "unfolding in a planar way" means that the pair of structures can be located in the region, transitivity applies and if each pair can be unfolded the entire PSL can be unfolded.

Routine R26 (see Chart F7) acts as a subroutine to R27 and does the actual testing of each structure pair. The pair are called the previous structure (the higher structure in the hierarchy) and the current structure (the descendant of the previous structure). There are four possible cases as can be verified by checking the allowable parent-descendant pairs of Figure 2-11-B.

(1) If the previous structure is a pivot pair set, the current structure must be a tree structure, as this is the only descendant a PP can have. It is possible by definition to permute the locations of the TS's of the PP so that the current structure TS is on the outside of the set, and hence bordering the same region that the PP borders.

(2) If the previous structure is a region structure, then the current structure must be part of the boundary of one of the possible outside regions of the RS in order to be located within the same region. Routines R24 and R25 test this.

(3) If the previous structure is a braced tree structure, the current structure, located on its map, by definition bounds the same region.

(4) If the previous structure is an anchored tree structure, then it must be attached to the PSL at its anchor node; so routine +33 is used to determine whether the tree structure can be unfolded in a planar way from its anchor node to the current structure.

Now the three cases of R28 corresponding to the three different common ancestors (base structures of the PSL's) of N1 and N2 can be considered in detail.

Base Structure is a Pivot Pair Set. - In this case the takeoff structures of PSL1 and PSL2 are both tree structures of the PP. By definition these two TS's can be permuted into positions adjacent to one another, and therefore bordering the same region. Thus it simply remains to apply R27 to PSL1 and PSL2 to test condition P3.

Base Structure is a Tree Structure. - Here several possibilities exist. If the base TS is braced, the takeoff structures of PSL1 and PSL2 are by definition located in the same region. If the TOS's of PSL1 and PSL2 are the same node, they are obviously in the same region. If neither of the above cases is true, the base TS is anchored or floating and TOS1 does not equal TOS2. In this case routine +33 is applied to see if the base TS can be unfolded in a planar way from TOS1 and TOS2. Then, again, R27 is applied to PSL1 and PSL2.

Base Structure is a Region Structure. - Here the base region structure is tested to see of TOS1 and TOS2 border a common region. If so, R27 is applied to PSL1 and PSL2.

The Feasibility Conditions. - In summary, the second halves of conditions F1-b and F2-a were covered earlier in this section. The first halves of these conditions were covered by showing that the planarity routine does not pass any non-planar graphs and does pass all planar graphs.

3.2.2 The Adjacency Requirement Satisfaction Routine

3.2.2.0. Introduction

The routine that satisfies adjacency requirements is A2 (see Chart F9). There are strong parallels between A2 and A1, the planarity test routine. Both operate on the parent structure hierarchy lists of N1 and N2, and both split their work between the base structure of the PSL's and the PSL's themselves. However, while A1 tests whether a realization exists in which the two nodes can be connected in a planar way, A2 actually implements the connection and restructures the graph according to the planar graph grammar.

A2 treats several different cases, corresponding to the different types of structures that can be the common base of PSL1 and PSL2. The PSL's can be thought of as representing chains of hierarchically organized graph structures, and these chains are attached to the common base structure at one end by their takeoff structures. A2 must twist and rotate the base structure so the takeoff structures of these chains border or fall within the same region. Then it must unravel the chains so that the nodes to be connected, which are at the extreme ends of the chains, can be located in the same region as the takeoff structures. The routine that does this unravelling is R68, which bears the same relationship to A2 that R27 did to A1.

In the following sections, first R68 will be described, and then the four routines corresponding to the various possible base structures of the PSL's will be described.

Reviewing the feasibility conditions, it must be shown that any planar graph can be structured hierarchically, and that the adjacency requirement satisfaction routine does in fact satisfy all adjacency requirements presented to it. The latter item is trivial, since the satisfaction of an adjacency requirement is synonymous with the addition of the desired edge to the graph. The first item is dealt with by mathematical induction. It will be shown that if no graph exists, two nodes may be connected to form a graph describable by the PGG. Then, given a graph that has been hierarchically structured by the PGG, it will be shown by enumeration of possible cases that if an edge is added to this graph, the resultant graph can also be hierarchically structured.

3.2.2.1 Routine for Restructuring a PSL

Overview. - Routine R68 (see Chart F12) has the title, "create a 'new tree structure' from parent structure list PSL." This reflects the basic philosophy of the restructuring of the dual graph when an edge is added. That is, if PSL1 and PSL2 have a common base structure, then the structures of each of these PSL's are unfolded and added to a "new tree structure" (NTS) map just as if they were going to be part of the map of a braced tree structure. The new tree structures of the two PSL's (NTS1 and NTS2) are then blended with the dual graph in a manner that depends upon what the base structure of their PSL's was. If PSL1 and PSL2 have no common base structure, R68 is employed somewhat differently, as will be explained in detail later. The following brief sketch of the way R68 works should aid in the understanding the more detailed description which follows it.

Outline of R68. - R68 proceeds down a PSL in the same manner as R27, considering a pair of structures at a time, naming them the previous structure (PS) and the current structure (CS). It constructs its new tree structure map as it goes. If the CS is a region structure, it is added to the NTS map. If the CS is a pivot pair set, action is deferred to the following step, when the PP will be the PS and a braced tree structure will be the CS. If a braced TS is the CS, a partial realization of its PP will be generated and the resultant new region merged with the RS that is the parent structure of the PP. If an anchored TS is the CS, its map, between its anchor node and the location on its map of the following structure on the PSL, is added to the NTS map. Any remaining part of its map is pruned off and converted to a new anchored TS having the NTS as its parent structure. If the CS is a node, it is added to the NTS map, and since this is the last structure on the PSL, R68 is terminated.

R68 has a number of important subroutines. Their main function is to update the IPL-V description lists of the various structures on the PSL as they are restructured. Because of the large number of special cases, going into detail on their routines would be counterproductive at this level of description. Thus, the reader will be referred to the paraphrased flow charts for details.

Now a more detailed explanation of R68 is presented, in which each possible current structure is considered separately. The large number of possible combinations precludes giving an example for each case. However, a single representative example will be developed. The situation is presented in Figure 3-3-A, in which the braced tree structure TS1 is the base structure of PSL1 and PSL2, and it is desired to connect the two darkened nodes N1 and N2. Only certain

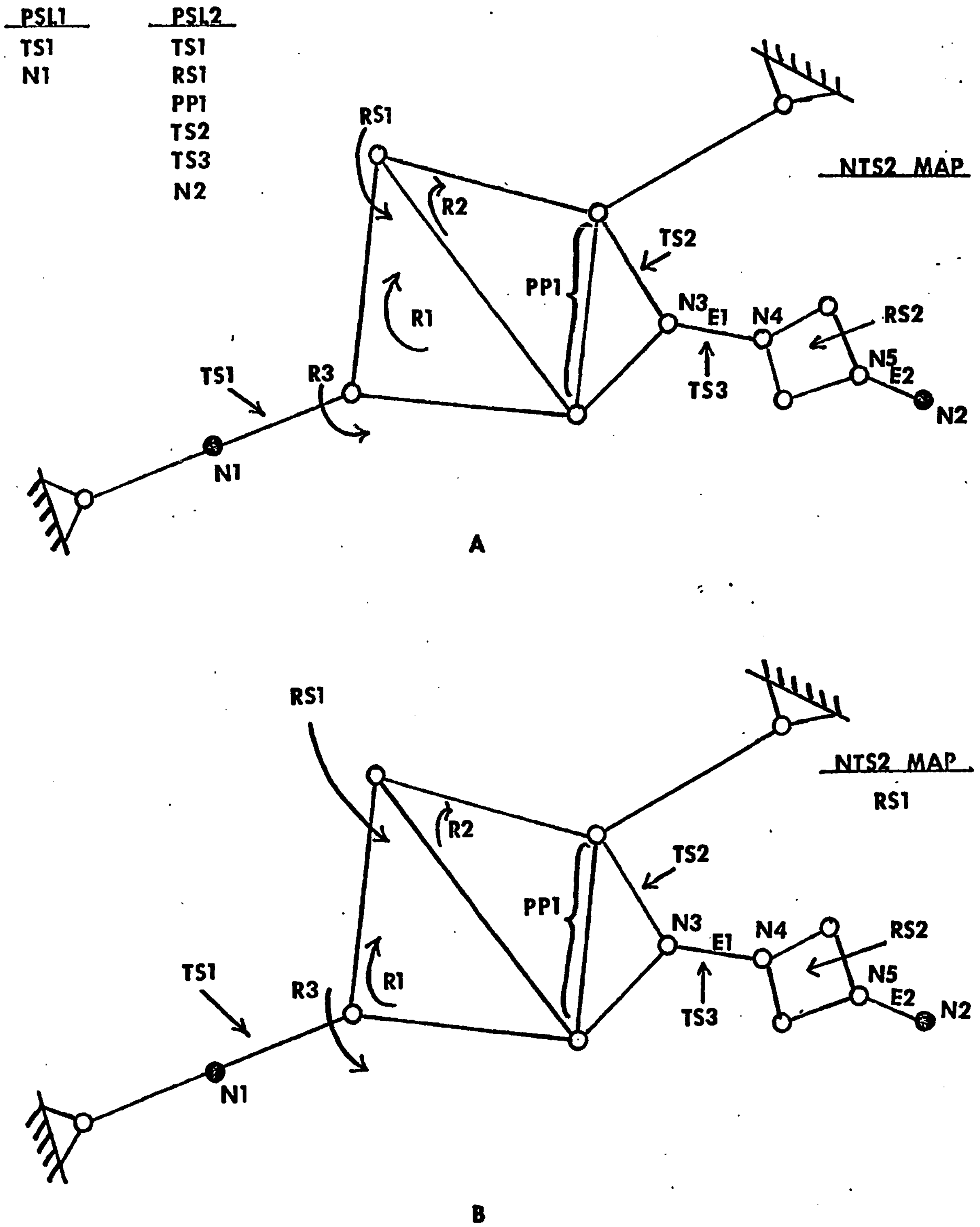


FIGURE 3-3

AN EXAMPLE FOR ROUTINE R68

pertinent graph structures are labeled. The small triangles attached to nodes denote the points of attachment of the structure to some larger graph. Note that as subsequent figures for this example are given, the actual graph might be the same, but the NTS is changed.

Current Structure is a Region Structure. - If the current structure is an RS, the previous structure must be a tree structure, as this is the only possible parent structure for an RS. Several situations can exist:

(1) If the TS that is the previous structure is the base structure of the PSL, the RS is simply stored on the NTS map by routine R32 (see Chart F16). This is the case in the example, and RS1 on the NTS map is shown in Figure 3-3-B.

(2) If case (1) is not true, but if the PS is an anchored TS, then routine R32 is again called. This time however, the RS will have already been on the NTS map since the map of the previous structure TS would have been added to the NTS map. Thus, R32 calls routine R31 (see Chart F18) which "prunes" the NTS map at the current structure RS. If the RS is the last structure on its TS, R31 does nothing. Otherwise, R31 calls upon routine +37 to make a tree structure of the NTS map from the node after the RS to the end, and to assign the RS as the parent structure of this newly created TS. The NTS map now ends in the current structure RS, its end having been pruned.

If neither case (1) nor case (2) is true, then the PS is a braced TS, and by previous manipulations of R68 (described later), the current structure RS is its only map structure. For this situation, the following two cases may hold:

(3) If the pivot pair set that is the parent structure of the braced TS is the base structure of the PSL, then R32 is called upon again. This time the NTS map will be empty, since none of the previous PSL structures will have been added to it. Thus, R32 calls upon routine R29 (see Chart F15), which merely adds the RS to the NTS map. (The required restructuring of the base structure PP in this case would be done by the PP base structure routine, R69, to be described later. R68 is a subroutine of R69).

(4) If none of the above cases is true, then the PS is a braced TS and its parent PP is not the base structure of the PSL. In this case, the current structure RS (call it RSB) is merged with the RS (call it RSP) which is the parent of the PP, by routine R67 (see Chart F20). R67 manipulates RSB so that the next structure on the PSL after RSB will border the same region as the takeoff structure of its PSL (the passed planarity test has guaranteed that this can be done). Then R67 calls upon routine R66 to merge RSB with RSP. R66 does this by splitting the boundary of the chosen outside region of RSB into two halves and placing these in the region boundaries of RSP where RSB's PP used to be. Then the remaining regions of RSB are added to the region list of RSP.

Cuttent Structure is a Pivot Pair Set. - In this case action is deferred until the following structure on the PSL, which will be a braced TS belonging to the PP, becomes the current structure. This is the case with the next PSL2 structure in the example.

Current Structure is a Tree Structure. - Here there are three separate cases, corresponding to the three different previous structures that are possible.

(1) Previous structure is a region structure - If the previous structure is the main region structure of the graph (this is the RS containing the outside space nodes and is named B1) then it must also be the base structure of the PSL, and the current structure TS must be anchored. In this case R32 is called which calls R29 which stores the entire TS map, including its anchor node, on the NTS map. The node following any region structure on the TS map is made its brace node, since these regions are now considered part of the map of a braced tree structure, the NTS.

If the previous structure RS is not the main RS, then routine R33 is called which makes the anchor node of the PS the brace node of its parent structure RS, provided that RS is on the NTS map. Then R32 is called, which calls R29, which adds the map of the current structure TS to the NTS map (including its anchor node if the NTS map is empty) and braces its region structures.

(2) Previous structure is a tree structure - If the previous structure TS is the base structure of the PSL and/or is not braced then the current structure TS's map is added to the NTS map (including its anchor node if the NTS map is empty) and its region structures are braced.

If the previous structure TS was braced, then it is currently part of the boundary of two regions of the parent RS of its former PP (R68 did this) and the anchor node of the current structure TS is made the brace node of that ᴾS by R33. Then, the map of the current structure TS map is added to the NTS map as before.

(3) Previous structure is a pivot pair set - If the previous structure PP is the base structure of the PSL, the base structure routine, R69, handles the restructuring of the PP and nothing is done at this point.

Otherwise, the routine R64 (see Chart F19) is called. R64 is a complicated routine because of the many special cases that it has to handle, but basically it makes the current structure TS part of the boundaries of the regions of the parent structure RS of its PP, and restructures that PP and RS accordingly. The result is that if the parent RS of the previous structure PP is on the NTS map, it winds up looking like a well-formed braced RS. If it is the base structure of the PSL, then it winds up prepared for the RS base structure routine, R74 (described later). In either case, the RS has been manipulated so that the present structure TS bounds the same region as the takeoff structure of its PSL.

In the example, the next structure on PSL2 is TS2 with a PS of PP1. Thus, R64 blends TS2 into RS1 and PP1 disappears since it only had two TS's, as shown in Figure 3-4-A. The next structure on PSL2 is TS3, and its entire map, along with its anchor node N3, is added to the NTS map, as shown in Figure 3-4-B. Also, N3 is made the brace node of RS1 by R33.

Current Structure is a Node. - The node will be the last structure on the PSL. It has two possible previous structures.

(1) The previous structure is a region structure - If the previous structure RS is the main RS (B1), the node is simply added to the NTS map by R29.

Otherwise, the node is made the brace node of the RS by R33, if the RS is on the NTS map, and then the node itself is added to the NTS map.

(2) The previous structure is a tree structure - If the previous structure TS is the base structure of the PSL, the current structure node is merely added to the NTS map.

If not, and if the previous structure TS is anchored, the NTS map is pruned at the CS node by routine R31 of R32 and the node is left at the end of the NTS map. This is the case in the example, see Figure 3-4-B.

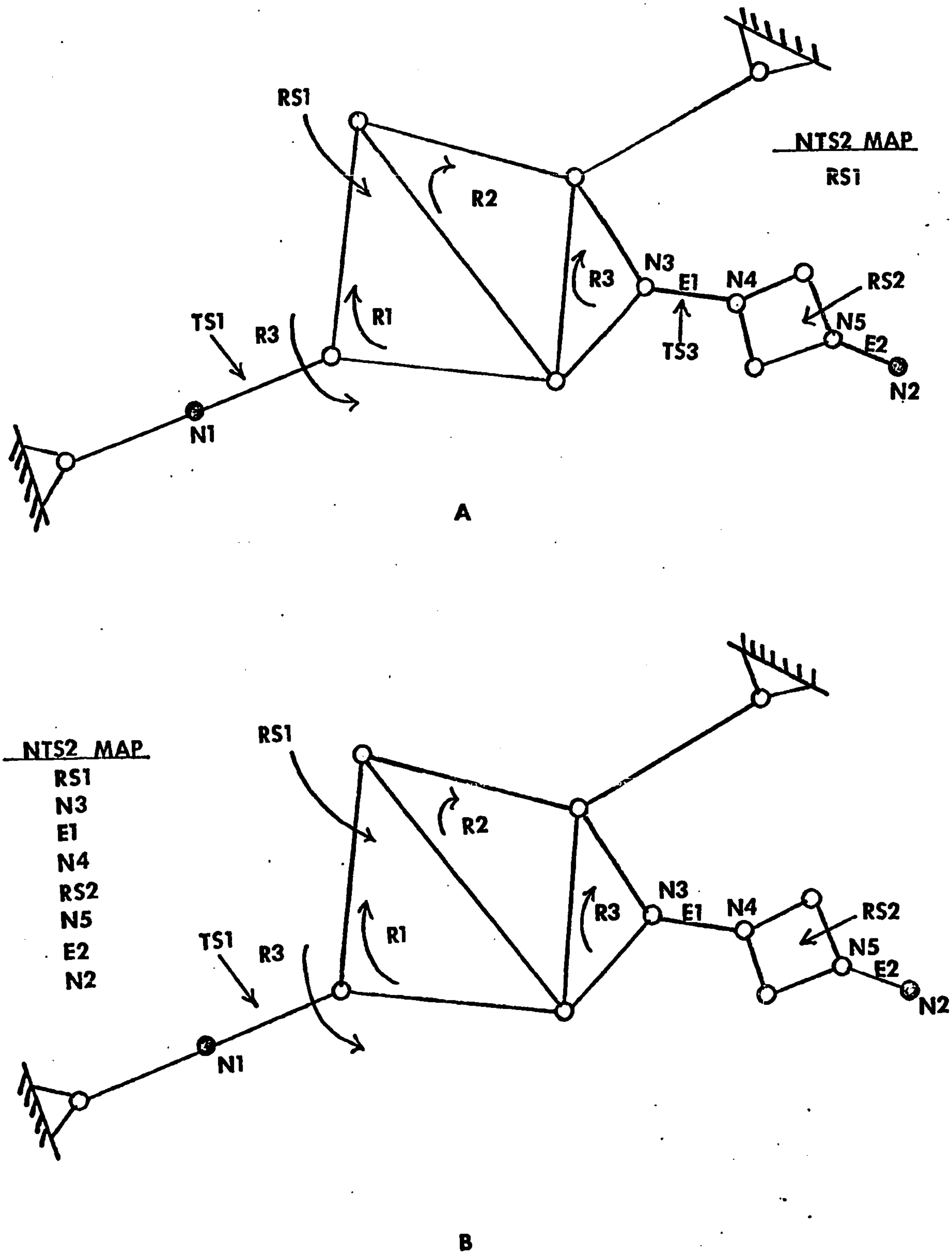


FIGURE 3-4

EXAMPLE FOR ROUTINE R68 (CONTINUED)

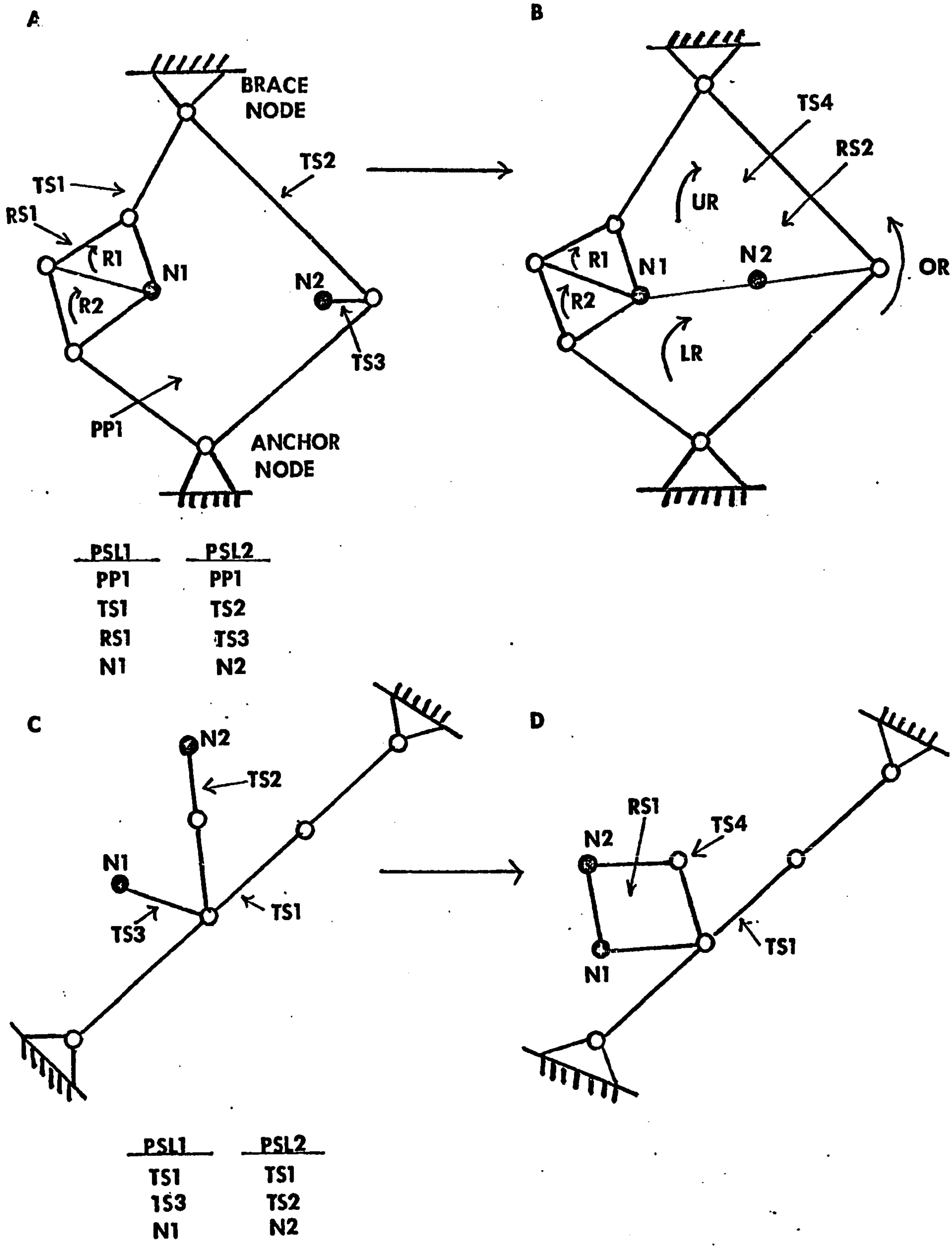
If the previous structure TS is braced, the parent structure RS of its previous PP is given the CS node as its brace node, if that RS is on the NTS map, and the node is added to the NTS map.

In all of the above routines, the description lists of the various structures are updated appropriately. One routine used in R68 and most other graph structuring routines is R49, the garbage collection routine. R49 erases the description lists of obsolete structures, and takes their names off of the main structure lists. The names of the structures are not erased, however, as they may be needed when the search routines refer back to a node in the search tree.

3.2.2.2 Routine for Pivot Pair Structure Common Base

If a pair of parent structure hierarchy lists (PSL1 and PSL2) have a pivot pair as their common base structure, then A2 calls upon routine R69, which will be described in this section. R69 (see Chart F10) has a lengthy IPL-V code because of the many different functions it has to perform, but it has no main branch points in its flow diagram, and it is quite easy to paraphrase. The example in Figure 3-5-A and B, with PP1 as the base structure and N1 to be connected to N2, will be used to clarify the explanation of R69.

If a PP is the PSL's base structure, then the takeoff structures of the two PSL's must be tree structures belonging to the PP. Because of the definition of a PP, these two takeoff structures TS's can be permuted into positions adjacent to one another. The overall goal



PSL1	PSL2
PP1	PP1
TS1	TS2
RS1	TS3
N1	N2

PSL1	PSL2
TS1	TS1
TS3	TS2
N1	N2

FIGURE 3-5

EXAMPLES FOR ROUTINES R69 AND R70

of R69, then, is to combine these two TS's into a single TS by joining the NTS's of their respective PSL's. The resultant TS is composed of a single RS, which roughly speaking can be thought of as being comprised of three main regions, the upper region (UR), the lower region (LR), and the outside region (OR) as shown in Figure 3-5-B.

R69 starts its operations by creating NTS's from PSL1 and PSL2. It then joins these two NTS's end to end by adding the edge connecting N1 and N2, and changes any RS's on the resultant NTS into PP's in preparation for this NTS being part of the boundaries of the UR and LR. Then R69 creates UR, LR and OR in succession, taking into account any RS that was the first structure on NTS1 or NTS2, as RS1 in the example.

Then these regions are all combined into a single region structure (RS2 in the example). This region structure is made the single structure on a newly created tree structure's map (TS4 in the example) and this resultant TS is added to TS list of the base structure PP, while the two merged TS's are removed.

Finally, if the parent RS of the base structure PP has only two regions (counting the outside region), and if the PP now has only one TS, this TS (which contains only the newly formed RS) is merged with the parent RS in order to preserve the pivot pair set conventions described in the planar graph grammar.

3.2.2.3 <u>Routine for Tree Structure Common Base</u>

<u>Overview</u>. - The routine which deals with the tree structure as

a common base structure is R70 (see Chart F11). R70 deals with three main categories of base structure TS's. These are determined on the basis of the location of the PSL takeoff structures on the base structure TS map, and are described in the following three sections. In general, in R70 the two chains of structures designated by PSL1 and PSL2 are joined end to end, thus completing a loop that includes the portion of the base structure TS between their two TOS's. This loop forms the basis for a new RS which is then added to the map of the base structure TS at the appropriate place.

The Takeoff Structures are the Same Structure. - This case holds either if the second structure on one PSL is an anchored TS and the second structure on the other PSL is its anchor node, or if the second structures on both PSL's are anchored TS's with the same anchor node. In either case, the takeoff structures of both PSL's are the same structure, a node. An example of this type of situation is shown in Figure 3-5-C, in which TS2 and TS3 have a common anchor node, and TS1 is the base structure.

When the two takeoff structures are the same structure, routine +44 is called by R70. +44 takes the new tree structures created from the two PSL's by R68 and combines these NTS's into a form suitable to be the boundary of a region. R70 then calls upon routine +45 to create a region structure out of the boundary list left by +44. This region structure is made into a tree structure and its anchor node and the map of the base structure TS are updated accordingly by R70. The result of this action for example is shown in Figure 3-5-D.

The Routine for Region Structure as Common Base is Applicable. - If the two takeoff structures are not the same structure, then R70 locates both takeoff structures on the base TS map and calls the one that is nearer to the anchor node the "lower structure" (LS) and the other the "upper structure"(US). R70 then applies an extensive series of tests (see Chart F11) to determine whether it is appropriate to apply the routine for which an RS is the common base structure (R74) instead of R70.

A brief explanation of these tests is necessary. Ordinarily (see next section), R70 creates a new region structure in the process of satisfying an adjacency requirement. The edge joining N1 and N2 becomes part of the boundaries of two regions of this region structure, and any other region structures that may be involved (LS or US if they are RS's or any RS between them on the base structure TS map) are either merged into this new region structure or converted into a pivot pair set. However, if there is a possibility that the edge joining N1 and N2 should be on the map of a TS belonging to a pivot pair set in G_{N+1}, the R74 is the appropriate routine to use, because it contains the mechanism for creating new pivot pair sets of this sort.

An example in which R74 is applicable is shown in Figure 3-6-A. Here the LS is RS1, the US is N2, and the base structure is the anchored TS1. The connection of N1 to N2 will create a pivot pair set, and R74 is already set up to handle the creation of PP's. Thus, R70 first alters the structure of TS1 so that N2 belongs to a newly created

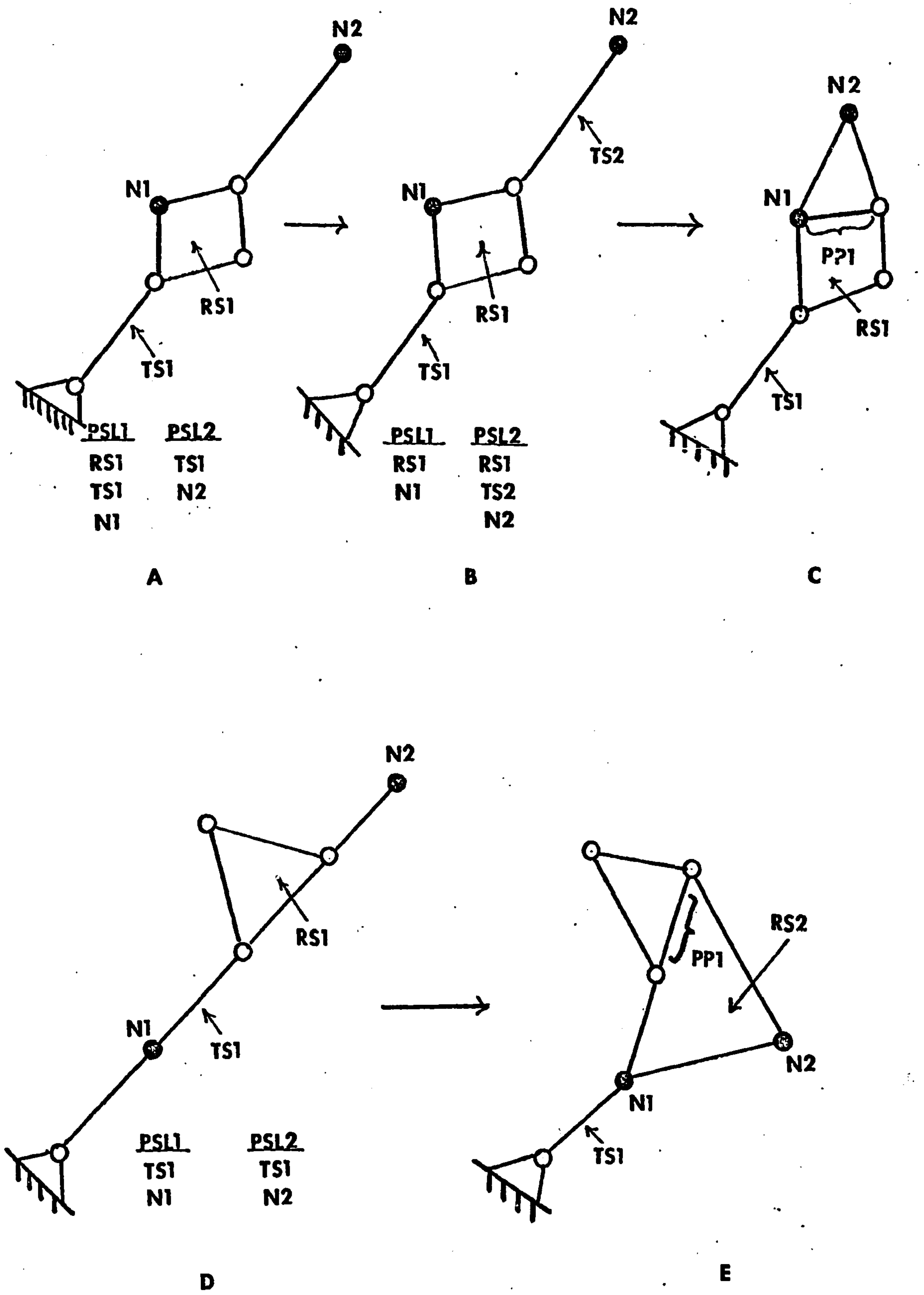


FIGURE 3-6

EXAMPLES FOR ROUTINE R70

tree structure, TS2, which has RS1 as its parent structure, the same as N1. This is shown in Figure 3-6-B. R70 then computes new PSL's for N1 and N2, and the common base structure of these new PSL's is RS1. Thus it is appropriate to apply routine R74, after which R70 updates the map of TS1, the previous base structure, and quits. The result for the example is shown in Figure 3-6-C. Similar action is taken in the other cases in which R74 is applicable, and Chart F11 should be consulted for details.

The Normal R70 Procedure. - If neither of the two above situations holds, then R70 calls upon routine R68 to create NTS's from PSL1 and PSL2. If there exist any region structures on either the NTS's or on the base TS map which should be merged with the new RS that R70 will create, these are collected at this time.

Next R70 calls on routine +43 to create a region boundary list of the two NTS's, much as +44 did in an earlier description. Then, +45 is called to make a new region structure out of this boundary list, taking into account any RS's that were supposed to be merged with the new RS. Finally, the base TS map is updated, and R70 is terminated.

An example for this case is shown in Figures 3-6-D and E. In this example RS2 is the new RS that R70 creates, and it becomes the terminating structure on the map of TS1. Notice that RS1 becomes a pivot pair set, PP1, in RS2. If RS1 had had three or more regions (counting its outside region), it would have been merged into RS2 by +45 instead of having been changed into a pivot pair set.

3.2.2.4 Routine for Region Structure Common Base

Overview. - The routine considered in this section is R74 (see Chart F12). The simplest case to imagine for this routine is the one in which the takeoff structures of the PSL's of the nodes to be connected are both nodes which border exactly one region of the base structure RS in common. Then, in effect, the NTS's of these two PSL's would be stretched across that region, dividing it in half. However, there are several other possibilities for the takeoff structures, and so the description of R74 is broken down into the sections that follow.

TOS1 Same as TOS2. - If the takeoff structures of the two PSL's are the same structure, then a situation exactly like the case of the common TOS's in R70 exists. In this case, R70 already contains most of the mechanism to process the adjacency requirement, so R74 calls upon it. R70 has been coded to handle an RS as the base structure in this special case. An example is shown in Figures 3-7-A and B. Notice that in Figure 3-7-B, RS2 has become part of the map of TS1.

One Intersection Region. - If TOS1 does not equal TOS2, then a test is made to see how many regions of the base RS TOS1 and TOS2 border in common. If they border only one region in common, then there is no possibility of the creation of a new pivot pair set, so R74 calls upon routine R71. R71 creates NTS's from the two PSL's, and stretches them out to divide the single intersection region into two regions. Any RS's on the combined NTS map are changed into single TS pivot pair sets, since the NTS map winds up as part of the boundary

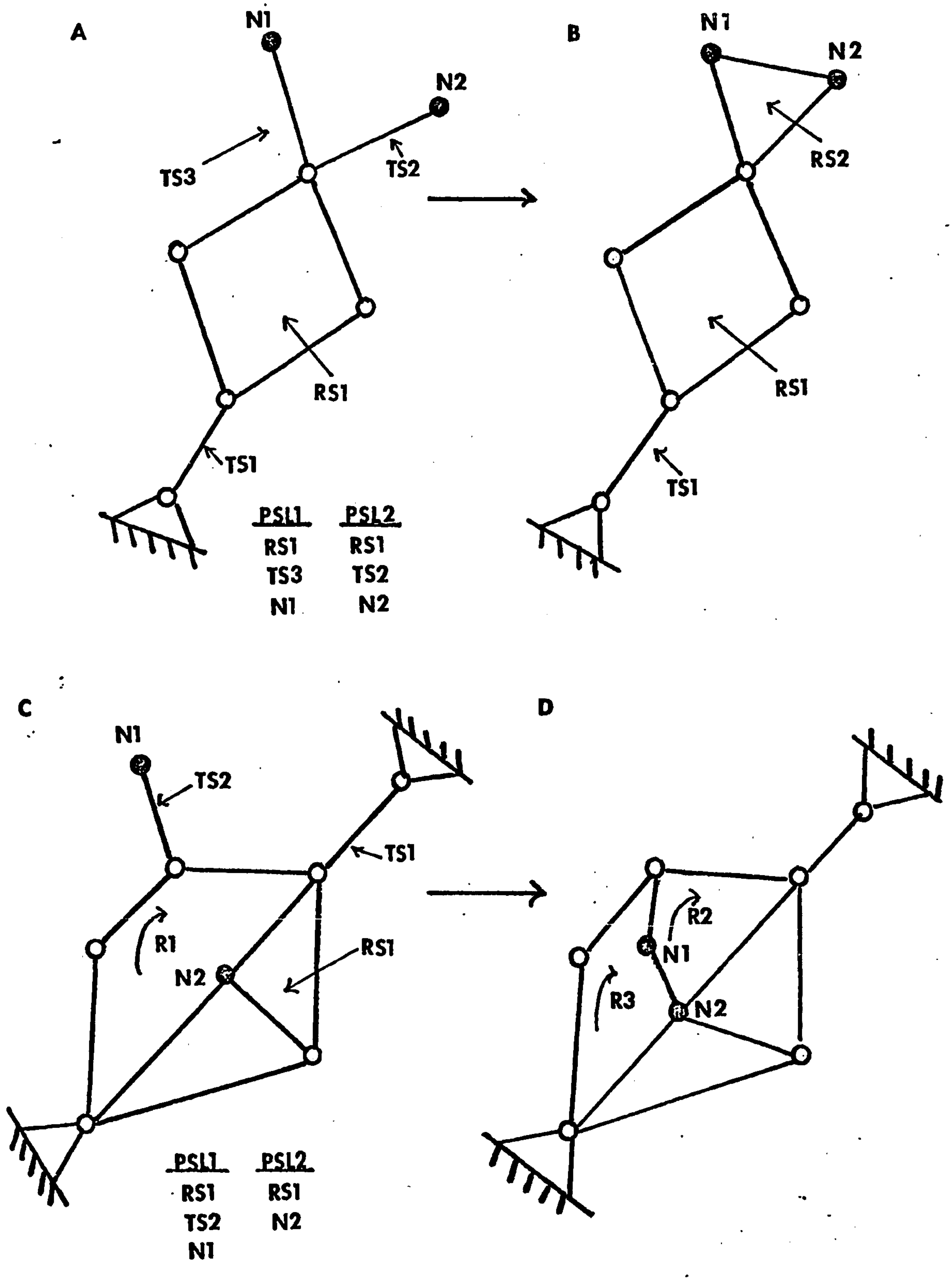


FIGURE 3-7

EXAMPLES FOR ROUTINE R74

of two regions of the base structure RS.

An example is given in Figures 3-7-C and D. Since that region R1 gets divided into regions R2 and R3.

Two Intersection Regions: Neither TOS is a PP. - If TOS1 and TOS2 border two regions of the base RS in common (this is the maximum possible number) and if neither TOS is a PP (meaning both TOS's are nodes), then a new pivot pair set must be constructed, if one does not already exist, with the two TOS nodes as endpoints. This step is necessary to maintain the well-formedness of the base region structure according to the planar graph grammar (see Section 2.4.1). Specifically, the PGG does not allow two nodes of an RS to border more than two g-regions in common. If R71 were used in the current case to simply divide one of the two intersection regions of TOS1 and TOS2 in half, then TOS1 and TOS2 would wind up bounding three regions in common. Thus, this possibility is avoided by instead creating a PP with TOS1 and TOS2 as endpoints, if one does not already exist. A more detailed explanation follows.

Upon determining that the conditions of this case hold, R70 uses R68 to create new tree structures of PSL1 and PSL2. If a pivot pair set already exists with TOS1 and TOS2 as endpoints, R70 calls upon R73 to make a braced TS out of the combination of NTS1 and NTS2, joined by the edge joining N1 and N2. This newly created TS is then added to the already existing PP.

If no PP exists yet, R70 calls upon R72 to create one. To do this, R72 must make a TS out of the region boundary portion between N1 and N2 and make this a part of the new PP. (This case is shown in the example of Figures 3-8-A and B. TS2 is the tree structure that has been formed out of a boundary portion.) If this new TS contains the node by which the base RS is attached at its upper end (the new away from the anchor node) to the map of its parent structure TS, then R70 must prune that parent structure TS at the base structure RS. This is done because the parent structure TS will no longer be the parent of that node, and hence its map must terminate with the base structure RS. Finally, R73 is called, as before, to make a braced TS out of NTS1 and NTS2, and to add that TS to the newly created PP. This TS would be TS1 in the example (Figure 3-8-B). The newly created pivot pair set would be PP1.

Two Intersection Regions; One or Both TOS's is a PP. - In this case, the usual result is that the TOS's that are PP's serve as the foundation for the creation of a new RS, which itself winds up as a single TS pivot pair set. An example is shown in Figures 3-8-C,D and E. A more detailed explanation follows.

R74 first tests to determine whether one TOS is the anchor or brace node for the other TOS, a PP. If this is true, then this node is temporarily added to the map of the TS which appears on the PSL and routine R70 (base structure is a TS) is applied, as it already has the mechanism for dealing with this case.

If one TOS is not the anchor or brace node for the other, then the base structure RS is tested to see if it would be well formed with a one TS pivot pair set. If not, the standard R74 subroutine, R71, can be applied (see above, One Intersection Region). If so, then R72 is called upon to create a single TS pivot pair set between and includ-

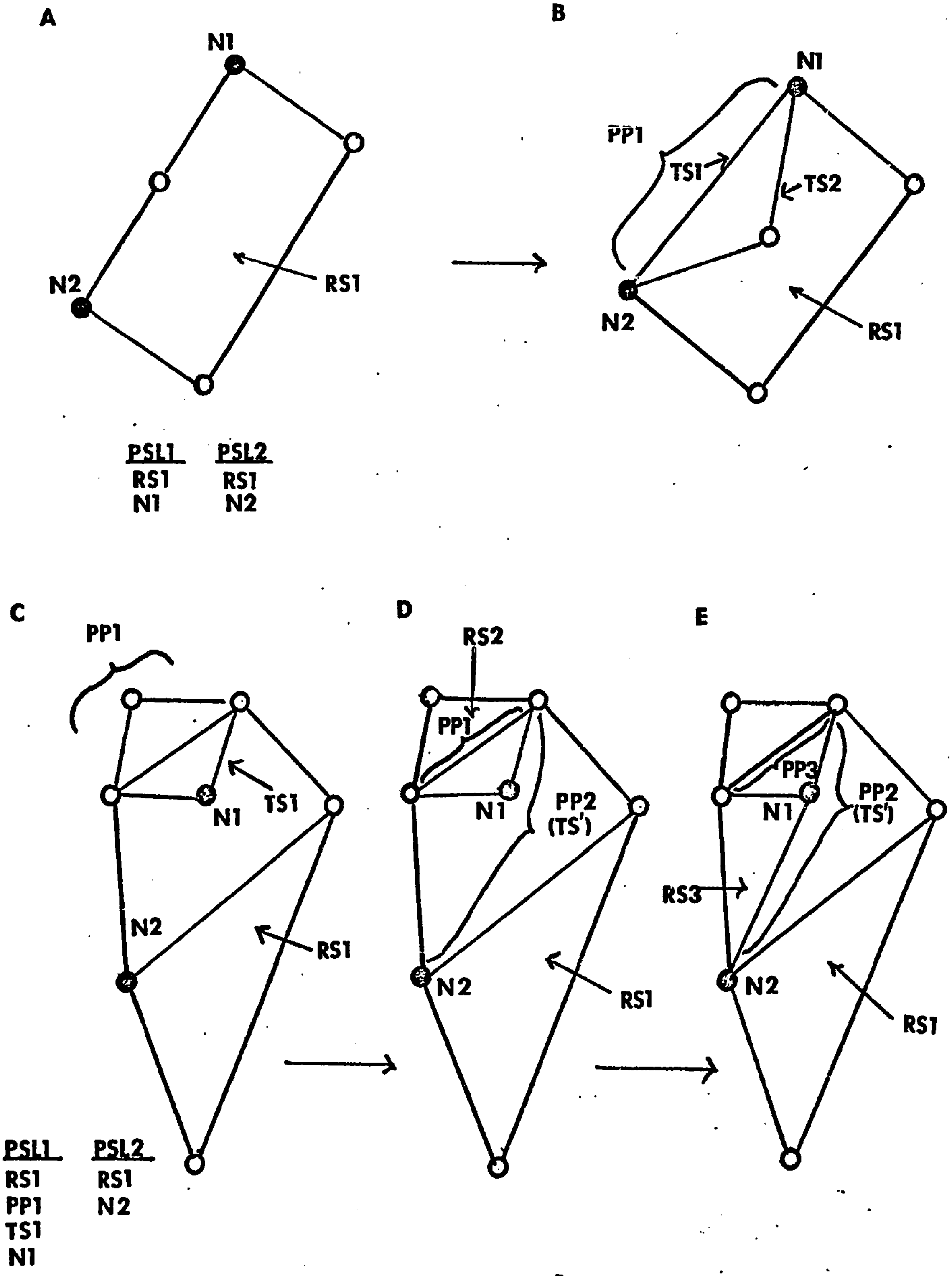


FIGURE 3-8

EXAMPLES FOR ROUTINE R74

ing TOS1 and TOS2 on the region boundary portion which they share. This would be PP2 (TS') in Figure 3-8-D of the example. Notice the region structure RS2 which has been made out of part of PP1. This is because PP1 had to be converted to an RS in order to occur on the TS map. The anchor and brace nodes of this PP are then temporarily added to the map of its only TS and routine R70 is applied. After this, the anchor node and brace node are removed from the map and appropriate structure updating is done. In the example, the result (Figure 3-8-E) is a single TS pivot pair set, PP2, composed of TS' which has region structure RS3 as its only map structure.

3.2.2.5 Routine for No Common Base

Overview. - The routine for this case is R75 (see Chart F13). The parent structure hierarchy lists submitted to R75 will have individual base structures of either B1 (the main region structure), a floating tree structure, or the node that is under consideration. The last case would correspond to a node that has yet to be attached to another node. A "size" hierarchy can be established among these three base types. B1 is the "largest", a floating TS is "smaller" than B1, and a floating node is "smaller" still. In general, R75 will structure the graph by adding a "smaller" structure to a "larger" one, and in the case of a tie an arbitrary choice is made. Obviously there can't be a tie between B1 and B1, because that would imply a common base, which is a contradiction to the assumption of no common base in R75.

Thus, R75 starts by first establishing the "size" hierarchy between the base structures of PSL1 and PSL2. It calls the PSL of the "larger" base structure PSLB and its associated node NB, while the "smaller" PSL and node become PSLA and NA respectively. In case of a tie an arbitrary choice is made, with one exception - the case in which both N1 and N2 are isolated nodes with no parent structures. This and the other cases are explained in the following paragraphs.

N1 and N2 are Floating Nodes. - In this case R75 simply makes a tree structure out of the two nodes and the newly created edge joining them. This is the verification that the program can construct a well-formed hierarchically structured graph of one edge. An example is shown in Figures 3-9-A and B.

NA is an Isolated Node, NB is Not. - Ordinarily this is a simple case of making an anchored TS out of NA and the new edge joining it to NB (NB will be the anchor node of this TS) and then attaching this TS to the parent structure of NB. An example is shown in Figures 3-9-C and D, where B1, the main region structure, is the parent of NB. If the parent of NB had been an anchored TS or RS, then it might have been necessary to do a little tree structure pruning to update the data structure.

However, a special case exists when NB is the anchor node of the base structure of its PSL. In this case a TS is created by adding the edge connecting NA and NB to the anchor node of the map of the base structure TS, preceded by NA itself, which becomes the anchor node of this new TS. An example is shown in Figures 3-10-A and B.

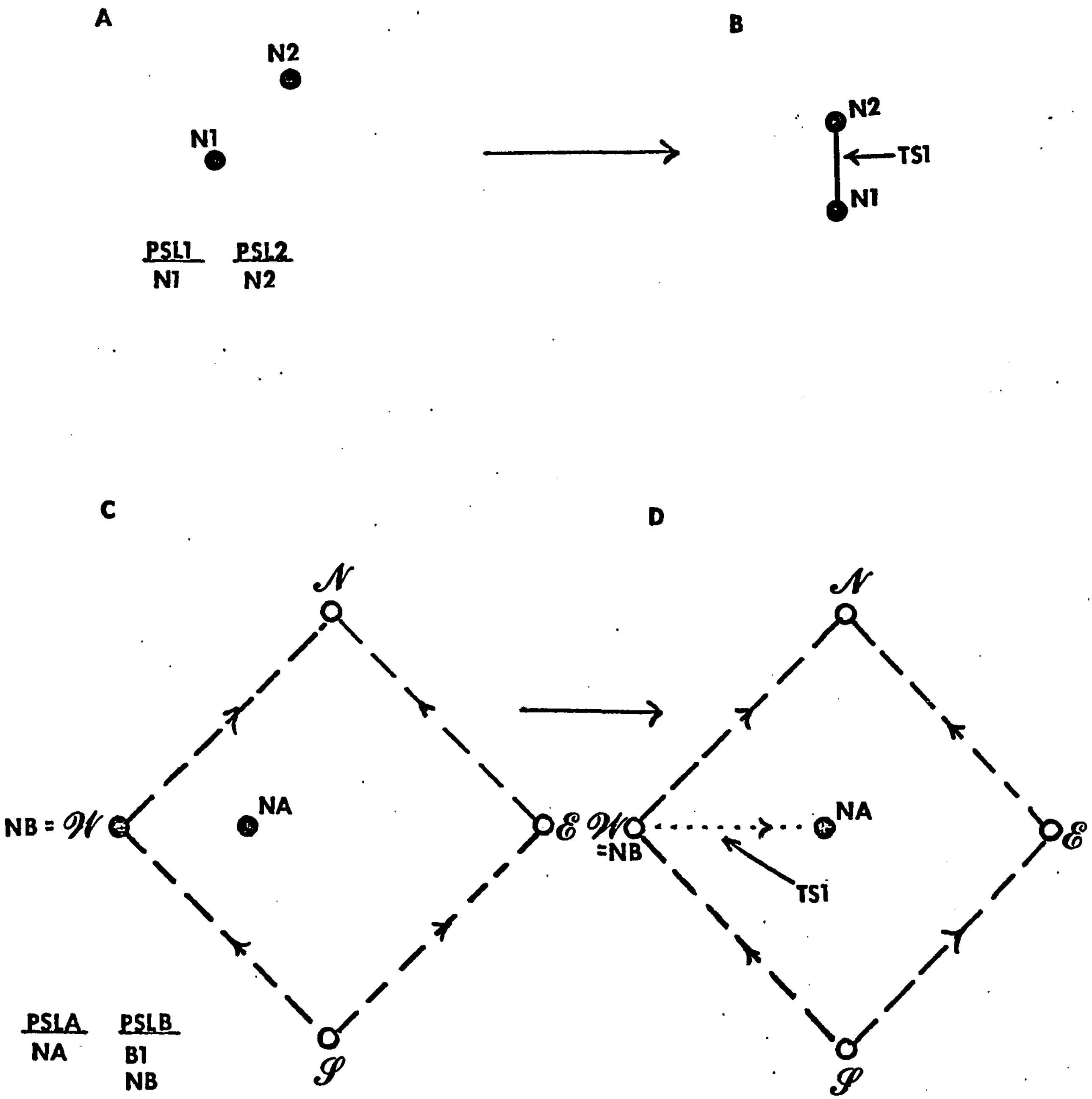


FIGURE 3-9

EXAMPLES FOR ROUTINE R75

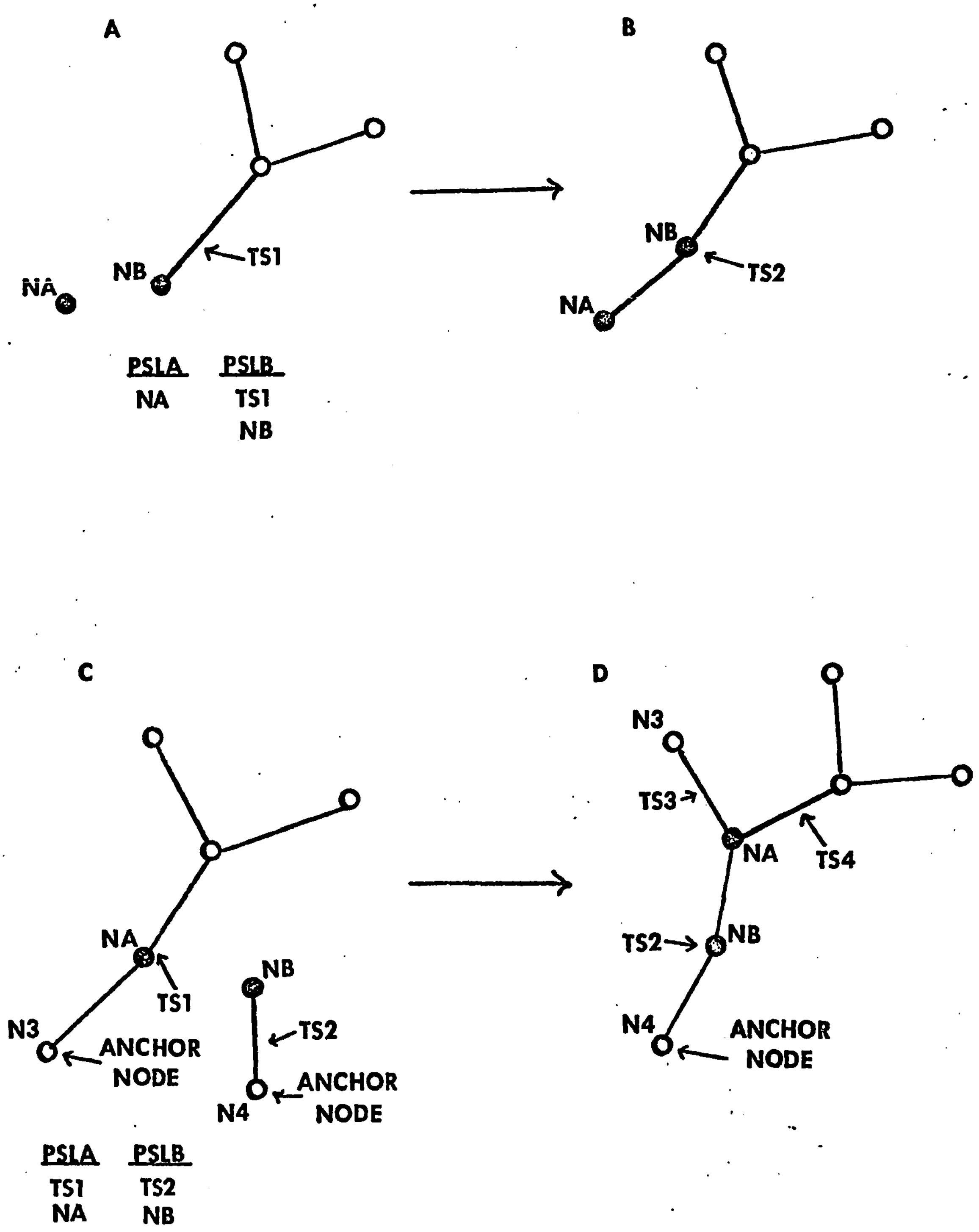


FIGURE 3-10

EXAMPLES FOR ROUTINE R75

Base of PSLA is a Floating TS. - If NB is not the anchor node of the base structure of its PSL, the normal course of action in this case is for R68 to be called to create a new tree structure (NTS) from PSLA. (Subroutines R64 and R67 of R68 treat this case of a "floating" NTS somewhat differently than the case of NTS's in the common base structure routines. This is because the planarity constraints do not require a "floating" NTS to be "stretched out" like a normal NTS, which is treated as a braced tree structure.)

Next NB and the edge connecting it to NA are added to the end of the newly created NTSA. Then routine +41 is called to create an anchored tree structure out of NTSA with NB as its anchor node. Since the branching of NTSA was originally constructed according to braced TS conventions, +41 must search the map of NTSA for nodes or RS's that have anchored TS's as descendants. As each such node or RS is encountered, this point is made the terminus of one anchored TS and the start of another, to preserve the convention that only the end structures of anchored TS maps may have descendant TS's. Finally, this new anchored TS is attached to the parent structure of NB, as before. An example is shown in Figure 3-10-C and D.

In the case that NB is the anchor node of the base structure of PSLB, alternate action must be taken as before. First a new anchored TS, called TS1, is formed by adding NA and the edge connecting it to NB to the end of the map of the base structure TS of PSLB. NA is stored as NB for later use, when TS1 will be attached to the parent structure of NB, (the former NA). If NA is also the anchor node of the base structure of its PSL, then that base structure TS must be restructured so that NA is no longer its anchor node. This is again done by +41, after first finding a suitable new anchor node (a node at the end of one of the descendant anchored TS's which has no descendants). Finally, for both of the immediately preceding cases, TS1 is attached to the parent structure of NB(the former NA) and this can be validly done because NB (the former NA) is not the anchor node of its base structure TS.

3.2.3 Generation of Dual Graph Realizations

If routine A8 can satisfy all of the adjacency requirements, the dual graph which it will have generated will be planar. In general, however, the dual graph will not be completely specified at this stage of the design. It is thus the task of the program to complete both the topological and the metric specification of the dual graph by filling in missing edges, edge colors and directions, and edge weights in accordance with the dimension requirements and the rules of physical realizability.

Most of this activity is the responsibility of Phase II of the program. However, the physical realizability tests (described in detail below) are designed to work only on a single region structure graph, that being the main RS, B1. No tree structures or pivot pair sets are allowed. Since the generation of such a graph from the partially completed dual graph, which possibly contains some TS's or PP's, is purely a topological operation, this activity is left to Phase I of the program.

Briefly, what Phase I of the program does here is, first, to anchor any floating tree structures or floating nodes of the dual graph, and then to brace all anchored tree structures of the resultant graph. The resultant graph of this operation is a single hierarchically structured region structure, B1, which may have many different planar realizations. Phase I then systematically and exhaustively generates these realizations for Phase II of the program to operate upon, provided they first pass the physical realizability tests of routine A11 (see next section). The fact that all realizations are generated, coupled with the fact that all possible anchorings and bracings of the floating and anchored tree structures are considered, satisfies feasibility condition b of Theorem F2 (see Section 3.2).

An outline of this aspect of Phase I was shown in the lower half of Figure 3-1. Notice that this part of the program is implemented by using recursive routines and IPL-V generators (the rough equivalent of FORTRAN DO loops, or ALGOL FOR loops). These routines automatically keep track of the bookkeeping for the generation of the realizations. Charts F21 and F22 show more of the details of this set of routines. The following comments, based on the charts, concern these routines, the generators, and the routines that the generators execute.

Because the current version of GRAMPA is incapable of unbuilding a graph edge by edge, at various places in a tree search such as the one conducted by these routines, it is necessary to make a copy of the current dual graph and save it. Routine R99 does this, as can be noted in the charts. Then when it is necessary to return to a given branch point in the tree search, routine R100 is used to reconstruct the dual graph according to the previously copied version. The copied graphs play an added role in the realization generation process because information describing the desired RS orderings, PP permutations, and outside region selections is stored in the form of attributes of the copied structures by the generators Z6 and Z10. Then, just before the realization implementation routines R142 and R143 are fired, the generators restructure the dual graph according to thc copied structures and their attributes. R142 and R143 then implement the realizations on the dual graph according to these attributes. This method of operation keeps separated the processes of generating alternatives and implementing alternatives.

In the charts it is noted that all possible node connection pairs are generated to anchor floating tree structures or nodes[5] and to brace anchored tree structures. This guarantees an exhaustive tree search. In addition, generator Z11 stores the connections that have already been tried for a given node on a master list, L16. Then if these same connections are attempted later by Phase II of the program (S2), routine A1 will note this and reject this attempt. This guarantees that no realizations are inadvertently repeated in the exhaustive solution search done by the program.

Note on Charts F21 and F22 that immediately after a pair of nodes are successfully connected, routine R126 is applied to check

[5] A floating node is a node which has no edges attached to it. This corresponds to a room with no required adjacencies.

the corners of the building. The purpose of this test is to see whether two rooms are trying to occupy the same corner of the building, which would be physically unrealizable in a floor plan. This test is directly derived from the triangular region property of Section 2.4.2, which states that no graph structures may be placed inside of a triangular region. This would be the case if two rooms were in the same corner of the building, as shown in Figure 3-11.

This particular test is done earlier in the solution search than the other physical realizability tests simply because it is feasible to do so. The other tests require a single region structure graph to work on, while this one does not. In general it is good strategy to apply a tree pruning test as early in the search as it becomes feasible.

After the floating and anchored structures have all been braced, the routines of R150 generate dual graph realizations according to the generation algorithm of Section 2.4.1.3. First generator routine Z10 generates all possible combinations of region structure orderings and pivot pair set permutations by marking them on the copied graph structures. Routine R143 implements PP permutations by first creating a braced RS, with the TS's of the PP converted into region boundary segments in the spatial order predetermined by Z10. This braced RS is then merged with the parent RS of the original PP by routine R120. R120 calls upon routine R66, which was discussed previously in the section on the satisfaction of adjacency requirements.

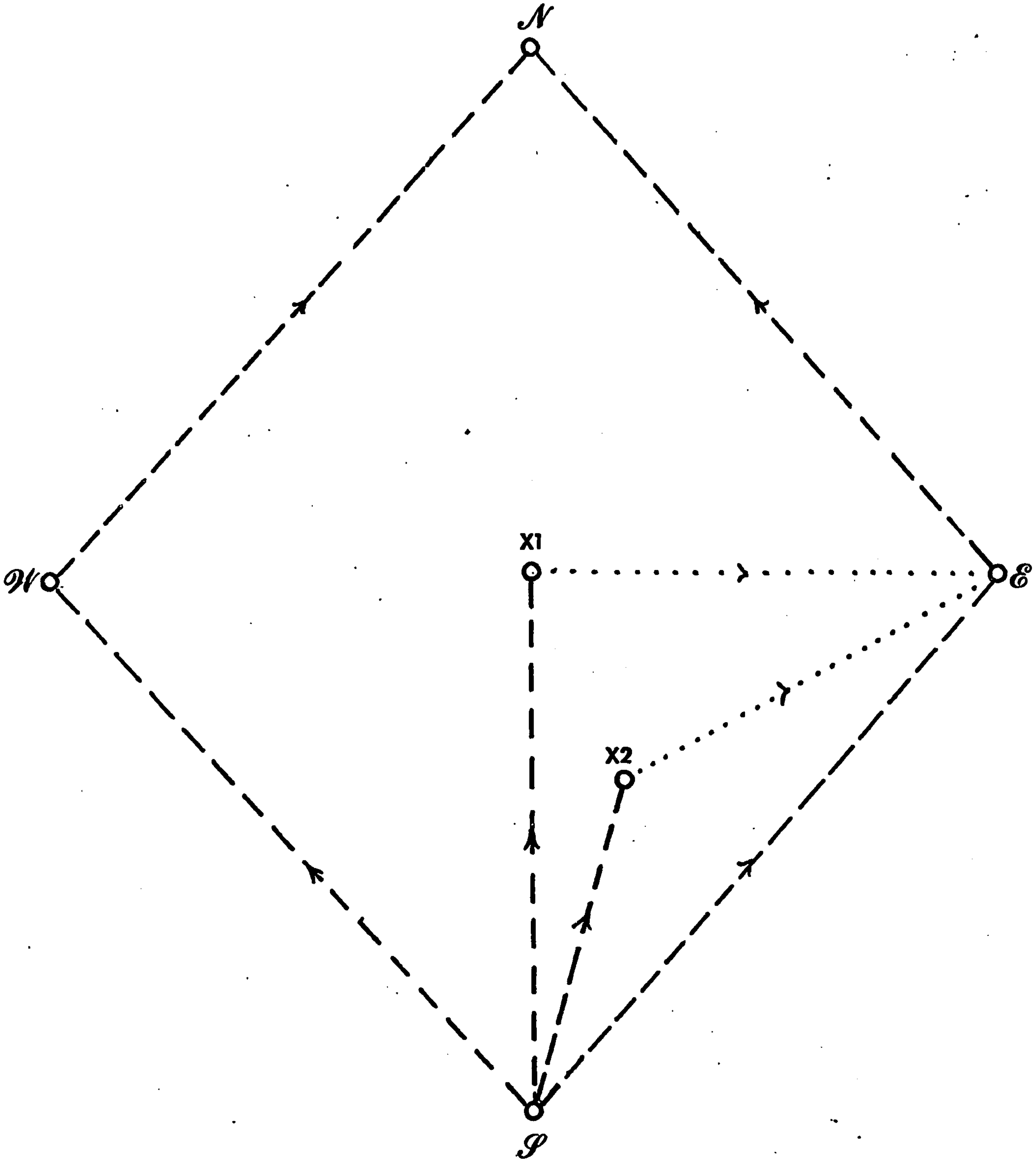


FIGURE 3-11

EXAMPLE OF TWO ROOMS TRYING TO OCCUPY THE SOUTHEAST CORNER OF A BUILDING

After all PP permutations are implemented, Z6 generates outside regions for any remaining braced region structures. These of course result from single TS pivot pair sets. R143 purposely did not merge these with their parent TS's because their outside regions would not have been chosen yet when R143 was fired[6]. Finally, R142 merges those remaining RS's with the parent RS's of their PP's and thus ultimately with the main RS, B1. Again, R120 is the routine that does this.

3.2.4 Topological Physical Realizability Tests

3.2.4.0. Introduction

Immediately after a dual graph realization has been generated in R150, the topological physical realizability tests are applied by routine A11. These must be done in such a way that feasibility requirements F1-C and F2-C are satisfied. The general flow of routine A11 is shown in Chart F23. It performs three different physical realizability tests: the triangular region test, the node well-formedness test, and the region-well-formedness test. These are discussed in detail in the following three sections, but basically they check for the well-formedness rules established in Chapter 2.

The two data structures global to these routines are the master node check list L9, and the master region check list L10. In routine A11 these lists are filled with all nodes and regions of the dual

[6] No outside region or RS ordering need be generated for the RS resulting from the implementation of a multi-TS PP permutation. This is already inherent in the planar ordering of the TS's of the PP.

graph. Later on in Phase II of the program (S2) the topological physical realizability tests will also be applied periodically using routine A9. These tests will be applied to those nodes and regions which are put on the master check lists by the other routines of Phase II. In general, only a few nodes and regions are tested each time A9 is fired in Phase II.

3.2.4.1. The Triangular Region Test

This test is applied to insure that no triangular region has been subdivided into regions by the realization generation routine. This is equivalent to testing whether there exist four possible adjacencies (corresponding to four rectangular walls) for each node of the graph, as was pointed out in Section 2.4.2. Thus, what routine A14 actually does is to check a node of the dual graph (excluding the outside space nodes) to insure that there exist enough possible connections to create four walls for the room corresponding to that node. See Chart F25 for details.

3.2.4.2. The Node Well-Formedness Test

Overview. - A3, the node well-formedness test and A4, the region well-formedness test use a common test format, provided by routines R79 and R101. These will be described in detail in this section as they apply to A3, and in the next section the differences in these routines as applied to A4 will be described. A given node to be tested may have some non- (colored, directed) edges incident to it, and a given region to be tested may have some non- (colored, directed) edges in its boundary. The purpose of R79 and R101 is to try different combinations of colors and directions for these edges to see if any combination exists under which the well-formedness rules are satisfied.

The Format Routine R101. - A3 as a routine simply loads the node check list, L9, into format routine R101 and fires it (see Chart F26 for R101). In the current version of the program the node region checks are applied only when the dual graph consists of a single region structure, B1. However, with some modifications it is also possible to apply these tests to graphs containing other region structures, pivot pair sets, and tree structures. Providing for this possibility accounts for much of the structure of R101. For simplicity, only the part of R101 that is used in the current version of the program will be treated in this section.

The first thing that R101 does to a node is to update its ordered edge list, using routine R80. The ordered edge list of a node is the list of edges attached to it arranged in clockwise order. Only those edges belonging to the parent structure RS of the node are considered (in this case, these are all of the edges attached to the node).

Next R101 applies the well-formedness test, R79, to the node. R79 uses property W1* of Section 2.4.2 to determine whether a node is well formed. It also has the capability of making some deductions about the colors and directions of edges which are uncolored and directed or colored and undirected. If the node passes the well-formedness test, R101 applies routine R78 which is capable of deducing the color and direction of edges which are undirected and uncolored. After this, R101 goes on to the next node of the check list. Some details on R79 and R78 follow.

R79 as the Node Well-Formedness Test. - The first thing that R79 does to a node is to glean all colored, non-directed or non-colored, directed edges on the ordered edge list and place these edges on a special auxiliary check list. Then all combinations of colors and directions of the edges on the auxiliary list are tried in an attempt to see which combinations lead to the satisfaction of the fundamental node well-formedness test, R77 (see details below). The successful combinations are noted, and after all combinations have been tried, if at least one combination has been successful, routine +50 is applied to each edge. If only one color-direction combination has been successful for that edge, +50 deduces that to be the actual color and direction of the edge and assigns it as such. If no combination of edges is successful, the test fails.

Routine R77 utilizes a "turn" property of colored directed edges which is modified from that discussed in Section 2.4.2. These turns measure the clockwise rotational "angle" between adjacent colored, directed edges about a node. Examples of +90° (+1), +180° (+2), +270° (+3), and 0° (0)[7] turns are shown in Figures 3-12-A, B, C, and D. R77 counts the number of walls needed by the room node to satisfy all its adjacencies by totalling the "clockwise turns about the axis of the node" between the edges on the ordered edge

[7] By an idiosyncracy, in the node test, angles are measured as positive in the clockwise direction.

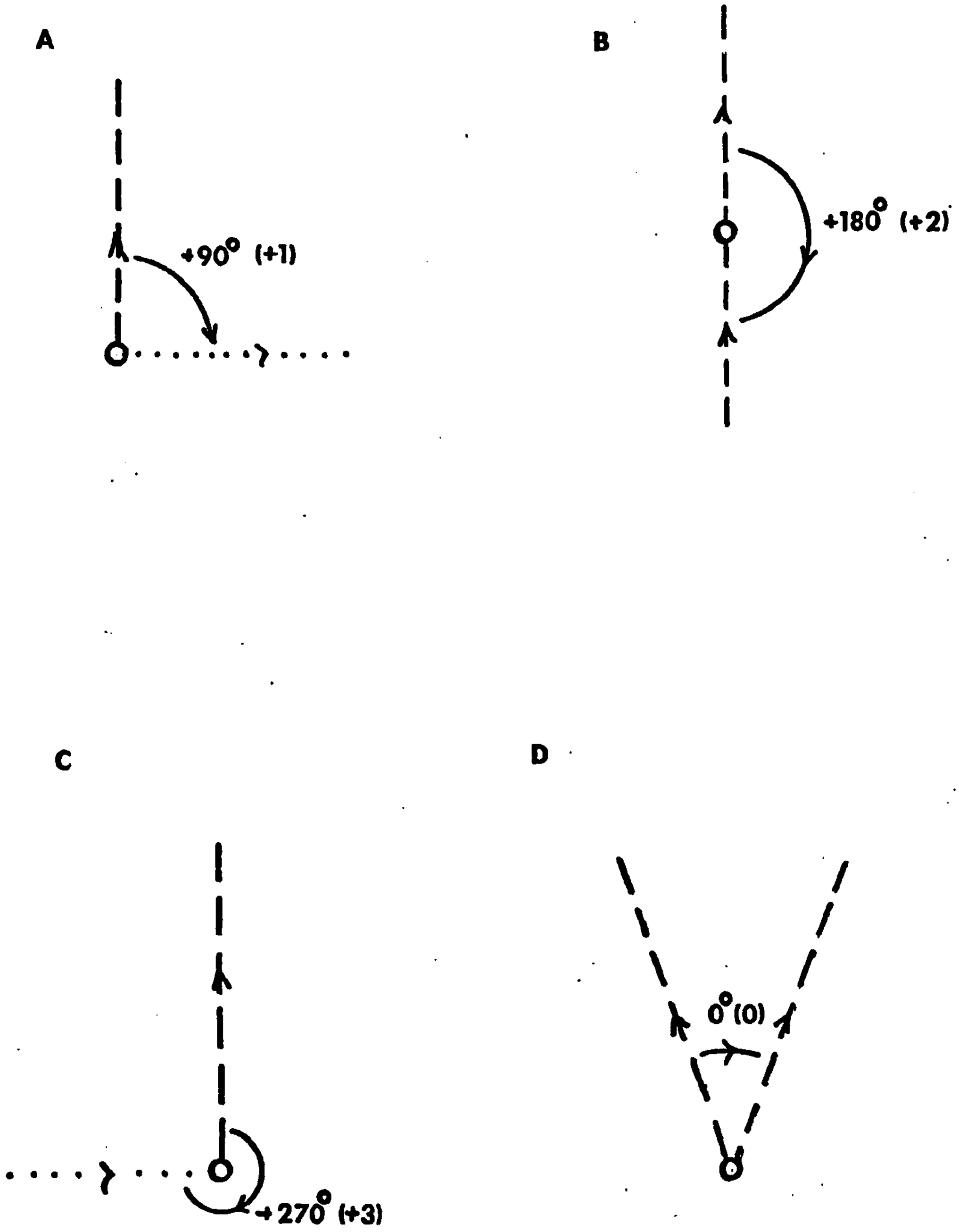


FIGURE 3-12

"TURNS" BETWEEN EDGES ABOUT A NODE

list. If the turns total more than 360° (+4), the room needs more than four walls and the test fails. For a partially completed room node, if the total is less than +4, the test will still pass.

R79, the Deduction Routine for Non-colored, Non-directed Edges.- In the case of nodes, routine R79 is only capable of deducing properties of edges with at least the color or direction already specified. This is done to avoid having to try a very large number of possible color-direction combinations for the non-colored, non-directed edges connected to a node. Routine R78 partially picks up this slack however, and essentially deduces the color and direction of a non-colored, non-directed edge if it is sandwiched between two edges of the same color and direction of the ordered edge list of a node. It will also deduce the color and direction of an edge if it is recognizable as being the only edge which can have that color, direction combination for the node under consideration.

3.2.4.3 The Region Well Formedness Test

Overview. - This test, A4, also uses routines R101 and R79 as formats. A4 first inputs the region check list, L10, and then fires R101. R101 first tests if the region well-formedness test is applicable by making sure that there are not so many non-colored, non-directed edges that trying all combinations of colors and directions would be infeasible. It then follows exactly the same set of operations as for the node test, with the exception that routine R78 is not used.

Routine R79 and the Region Well-Formedness Test. - In the case of regions, R79 again prepares an auxiliary check list, but in this case all edges that are not both colored and directed are included. R79 can afford to do this because it does not test regions with more than one non-colored, directed edge. Aside from this, the only difference from the node check is that the well-formedness test R84 instead of R77 is applied.

R84, the Basic Region Well-Formedness Test. - What R84 does is to simply apply, verbatim, the physical realizability criteria for Theorem 3 (see Section 2.4.2.).

One final note on the feasibility conditions F1-c and F2-c. Since the well-formedness tests rely on necessary and sufficient conditions for physical realizability, F1-c and the first part of F2-c, are certainly satisfied. As for the second part of F2-c, the requirement that no well-formed node or region can be created by adding structures to a non-well-formed node or region, two cases exist:

(1) No well-formed node can be created by adding extra edges around a non-well-formed node because the turn sum will always exceed 360°.

(2) No well formed region can be created by adding extra edges inside a non-well-formed region because Theorem 2 says that a non-well-formed region is not "fillable".

3.3 Phase II of the Program

3.3.0 Introduction

The task of Phase II is to take the realizations generated by Phase I of the program and from them produce all final solutions which exist to the design problem. This involves filling in the unspecified adjacencies of the dual graph in a well-formed way, and filling in the edge weights of the dual graph in a well-formed way in response to the problem's dimension requirements.

Actually, the task of completing the design solution search began in Phase I, immediately after all adjacency requirements had been satisfied. At that point any one of several solution search techniques could have been initiated. The technique implemented in this program is exhaustive search, first generating region structure realizations of the partially specified dual graph and then attempting to produce final solutions from these realizations by assigning the remaining adjacencies and edge weights in parallel in Phase II of the program. Other possible techniques for implementing the search are discussed in Chapter 5. These include heuristic search techniques based on the present technique, a mathematical programming approach, a GPS approach[8], and a circuit simulation approach. All of these methods allow the possibility of specifying size requirements by room area rather than by specific room dimensions.

[8] GPS (General Problem Solver). See Reference 22.

The present search technique was used in preference to these other techniques because the representation lends itself well to the parallel treatment of topological and dimensional considerations in completing dual graph realizations. All but the first of the other four methods treat the topological completion and the dimentional completion of the realization separately, in series. This discards the possibility of early search tree pruning due to the interaction of topological and dimensional constraints.

Regarding the heuristic search technique mentioned first, it was felt that experience should be gained using the exhaustive search technique before this method was attempted. However, the use of the exhaustive search technique has limited severely the size of the design problems that can be attempted with the current version of the program, as described in Chapter 5.

To return to the present search technique, in describing Phase II the issue of feasibility, introduced in Section 3.1, must again be considered. To review, the issues relevant to feasibility in Phase II of the program are:

(a) the decision scheme for the topological and dimensional completion of dual graph realizations

(b) satisfaction of dimension requirements

(c) local tests of dimensional well-formedness

(d) global tests of dimensional well-formedness

(e) satisfaction of adjacency requirements (also in Phase I)

(f) test for well-formedness of nodes and regions

(also in Phase I)

The following two theorems provide conditions for the feasibility of the dual graphs generated by Phase II of the program.

Theorem F3:

Three necessary conditions for Phase II of the program to generate no infeasible partially completed design solutions are given below. If the solution under consideration is a final solution (a completely specified dual graph), then the conjunction of the conditions is both necessary and sufficient for feasibility.

(a) All regions and nodes of the dual graph must be well formed.

(b) All weights assigned to edges of the dual graph must satisfy the rules of dimensional well-formedness.

(c) All dimensional and adjacency requirements which have been considered must be satisfied.

This theorem is proved in Appendix A1.7. As in Phase I, effort is devoted in Phase II to letting as few infeasible partially completed graphs as possible slip by the feasibility tests.

Theorem F4:

Five necessary and sufficient conditions to guarantee that Phase II of the program generates all feasible final design solutions are:

(a) No well-formed nodes or regions may be rejected by the node and region well-formedness tests.

(b) No well-formed assignments of weights to edges may be rejected by the dimension well-formedness tests.

(c) The global dimension well-formedness tests may reject only infeasible solutions.

(d) No feasible solutions may be generated by adding detail to solutions rejected by either the (1) node, (2) region, (3) dimension, or (4) global dimension well-formedness tests.

(e) All possible combinations of room size assignments and topological completions of the dual graph realization must be considered by the program.

This theorem is proved in Appendix A1.8. Except for F3-c, F4-d, and F4-e, all of the conditions of the above two theorems are satisfied provided the various well-formedness tests of Phase II perform their jobs properly. This is true because the well-formedness tests are the only mechanisms of the program involved in these conditions. Parts 1 and 2 of condition F4-d were already proved in describing Phase I of the program. Part 3 and 4 are covered in Sections 3.3.3 and 3.3.4 respectively. Conditions F3-c and F4-e are covered in Section 3.3.2.

3.3.1 Outline of Phase II of the Program

In Phase I of the program, the bookkeeping for the generation of the dual graph realizations was taken care of automatically by the use of recursive routines and the IPL-V generators. This format was appropriate for the generation of realizations because, with the exception of R126, tests were only applied after a complete realization had been generated. Thus there was little reason to prefer any other search strategy than the depth first strategy forced by the use of recursive routines and nested generators. Also, the fact that IPL-V generators and recursive routines are rather covert in how they do their bookkeeping was of little concern. There was little need to monitor the realization generation process as long as it was exhaustive and non-redundant.

In Phase II however, tests are applied each time an individual piece of detail is added to the partially completed dual graph. This points to the possibility of some other sort of solution search than depth-first. There is also a strong motivation for wanting to monitor both the generate and the test activities of Phase II. This rules out the bookkeeping techniques used in Phase I.

Thus, in Phase II, although depth first search techniques are used in the current version of the program, the possibility of doing a more selective type of solution tree search with minor modifications is kept alive by using a different type of bookkeeping. This method uses R99 to create a copy of the dual graph for storage at each search node, just as in Phase I. However, instead of using generators or recursive routines to guide the search, this method uses single level iteration routines, and keeps track of which alternatives have been tried at a search node by storing lists of them as attributes of the search nodes. This method also makes monitoring of the search easier than in Phase I. More details on this method are given in the following sections.

What follows is a brief sketch of Phase II of the program: S2 is the highest level routine of Phase II (see Chart F28). S2 selects non-terminal regions of the dual graph one at a time and tries to complete them, using S3, the region completion routine. S3 tries to complete a region by assigning dimensions to undimensioned nodes and filling in edges until all region nodes are dimensioned and the region is filled. The basic working routine of S3 is R113, which tries to implement each node dimension assignment or edge addition. R113 makes use of a battery of physical realizability tests to determine the feasibility of each graph addition. The main tests are the node and region well-formedness tests, described in Section 3.2.4, and the dimension and global well-formedness tests, described in the following sections.

3.3.2. The Search Strategy

3.3.2.0. Introduction

It has been pointed out that the search strategy used in Phase II is one of several possible alternatives. It was selected because it was felt to be the strategy that would provide the most immediate opportunity for the development of the dual graph manipulation routines. A depth-first search is used, with a certain amount of guidance supplied by test routines and alternative selection routines. The main guidance routines are listed below:

(1) Subroutine R119 of S2 selects the next region to work on at each region selection node of the search tree.

(2) Subroutine R106 of S3 selects the next node to dimension or edge to add to the dual graph at each move selection node of the search tree.

(3) The physical realizability tests of R113 can terminate any given branch of the search tree with a failure.

The following sections provide details on the various routines associated with the solution search strategy.

3.3.2.1. S2, The Phase II Executive Routine

Overview. - A paraphrased flowchart of S2 is given in Chart F28. In general, S2 selects non-terminal regions of the dual graph, one at a time in a depth first fashion, and tries to complete them, using routine S3. Corresponding to each node in the search tree at which the next region is selected, there is a search node information list called the search node. The search node is passed on to routine S3 as input data. The attributes of a search node are as follows:

(1) The name of the region being worked on.

(2) L8', the copied list of the graph structures which existed just before S3 started to try to complete the current region.

(3) The list of region completion moves tried by S3.

(4) The list of the remaining non-terminal regions of the graph.

Referring to Chart F28, S2 simply selects and tries to complete regions of the dual graph in a depth first fashion. Whenever the region completion routine S3 reports a failure, S2 returns to the previous search node and inputs that to S3. S3 then resumes generating completions for the region of that search node where it left off the last time it worked on it. S3 can do this because the move list attribute of the search node tells it what moves have already been tried for that region.

R119, The Region Selection Routine. - The routine that selects each region to work on is R119. In the current version of the program, R119 weights the following factors in selecting a region:

(1) If some of the remaining non-terminal regions of attribute (4) of the search node border outside space nodes,

then routine +60 creates a sublist of regions from that list for consideration by R119. In +60 the outside space node (or nodes in case of a tie) is selected which requires the lowest non-zero number of edge additions to fill in the non-terminal regions bordering it. These non-terminal regions are the ones put on that sublist for R119. The motivation for this selection operation is to guide the solution search so that an outside space node is surrounded by terminal regions as early as possible. This allows the very powerful global well-formedness tests to be applied, thus leading to early pruning of the search tree if no solutions exist along the current branch.

(2) From the sublist of non-terminal regions under consideration, R119 considers only those which border the most recently completed region in the search, provided such regions exist. This is done because of the assumption that if search effort is concentrated on the areas of the dual graph with high information density, physically unrealizable configurations, if they exist, will be detected early in the process, thus pruning the search tree.

(3) The selected eligible regions are weighted according to the three criteria listed below. The region with the highest weight is selected for completion.

(a) A weight of -1 is added for each edge over the minimum number of three in a region. This is to insure that small, rapidly completable regions are worked on. Rapid completion of regions generally leads to feasibility contradictions early in the search.

(b) A weight of -1 is added for each non- (colored, directed) edge in a region. This again contributes to the rapid completion of regions.

(c) A weight of +1 is added for each dimensional node or weighted edge in a region. This guides the search to regions of high information density.

R157, The Search Mop-up Routine. - When no non-terminal regions of the dual graph remain to be filled in, it is still possible, because of the way the search was conducted[9], that some local detail of the dual graph has been unspecified or untested. R157 (see Chart F29) sees that these final details are attended to.

The first thing R157 does is to generate colors and directions for any non- (colored, directed) edges remaining on the dual graph. It does this in a recursive fashion, but since there are usually only a few edges of this type, this is not considered a serious departure from the bookkeeping methods of Phase II.

If all edges can be colored and directed, the next thing R157 does is to apply routine R155 to check the well-formedness of any regions that might have been marked as potentially four-sided terminal

[9] The critical item in the solution search is to drive toward the termination of unproductive search tree branches as quickly as possible. Thus, local details are sometimes overlooked and must be mopped up later.

by R113 (explained below). Finally, if this test is passed, R157 applies routine R160 to assign dimensions to any remaining undimensional nodes. If this can be done, a design solution has been reached and R160 prints it. It can be assured that a valid design solution has indeed been reached for the following reasons:

(1) Whenever an edge is given a weight during the solution search, its two endpoint nodes are checked for dimensional well-formedness.

(2) Whenever an edge is added to the graph, or a color or direction is added to an edge during the solution search, the two regions bounding that edge are checked for well-formedness, and the two endpoint nodes of that edge are checked for dimensional and topological well-formedness.

(3) The solution search routine guarantees that all regions of the graph as input to R157 are either triangular terminal or marked potentially four-sided terminal.

(4) R157 insures that all edges are colored, directed, and weighted. The edge weight deduction capabilities of routine A5, called by routine R160, are instrumental in seeing that all edges get assigned weights. See the last paragraph of Section 3.3.3 for clarification on this matter.

(5) Facts (1) through (4) guarantee that all nodes of a completed dual graph will be checked for dimensional well-formedness, and that all nodes and regions are checked for topological well-formedness. This in turn means

that on the floor plan corresponding to the completed dual graph, all rooms are rectangular with the correct dimension,all desired adjacencies are satisfied, and all adjacencies between rooms are topologically well formed (recall that a well-formed terminal region corresponds to a physically realizable corner at which rooms meet). This will be taken as adequate proof that the entire floor plan in well-formed.

3.3.2.2 S3, the Region Completion Routine

Overview. - The flowchart for routine S3 is given in Chart F30. A search node from routine S2 is the input to S3. As mentioned in Section 3.3.2.1, routine S3 adds a "move list" to the search node. The concept of a region completion "move" is associated with the solution search algorithm that S3 uses. Basically speaking, however, S3 attempts all possible dimensional and topological completions for the region under consideration.

The Search Algorithm of S3. - S3 considers a region to be completed when all of the nodes in its original boundary have been dimensioned (i.e. a given pair of x and y dimensions has been chosen for the room corresponding to each node) and when its interior has been filled in with colored, directed edges in such a way as to divide it into well formed terminal regions. This does not necessarily imply that all of the edges of the original boundary have been colored and directed, as this task is left up to the color and direction deduction capabilities of the physical realizability tests. This is one reason why routine R157 is necessary in S2 (see above).

Thus, there are two kinds of steps that S3 can take in completing a region. It can dimension a node, or it can fill in one or two edges. These two types of steps are called node moves and edge moves respectively. If S3 chooses to dimension a node as its next move, it can choose from among the list of allowable room sizes assigned to that node by the dimension design requirements which were input to the program. These are called the alternatives for the node move. Similarly, if S3 chooses an edge move, it can choose from four different color-direction combinations for each edge that it wants to add to the graph. These are the alternatives for the edge move. If a single edge move involves the addition of two edges, there would be sixteen alternatives for that move. The alternatives for an edge move are actually more complicated than this. (See below.) S3 attempts one move at a time, either a node move or an edge move, and routine R106 (explained below) is used to select which move to attempt next.

The edge moves are chosen so as to search the region completion space in an exhaustive, but non-redundant fashion. This is done by using the concept of "building on an edge". For a given region to be filled in, R106 chooses an edge to build on. This means that edge moves will be generated that create all possible terminal regions having the chosen edge as one side and extending into the region to be filled. An example of all such regions for some edge, E, and some region, R, is shown in Figure 3-13. Instead of calling

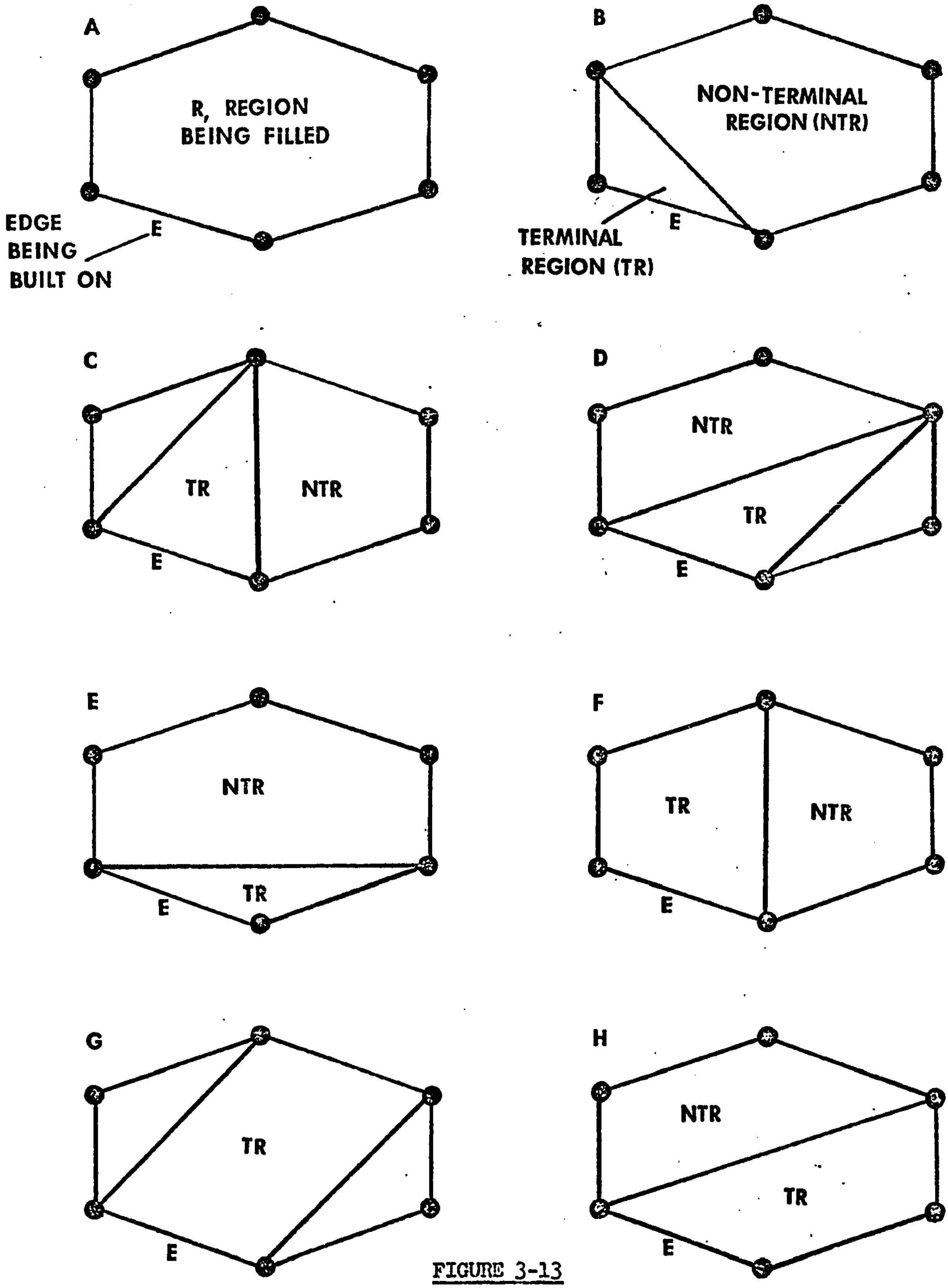


FIGURE 3-13

TYPES OF TERMINAL REGIONS BUILT ON AN EDGE each possible terminal region construction a different edge move, S3 simply reverses an edge move for each edge being built on. Then the various terminal region constructions, along with the various combinations of edge color and direction that go along with them are all considered as alternatives of that edge move. Thus, considering only the edge moves the process of filling a region proceeds with the following steps:

(1) An edge is chosen to build on, and a terminal[10] region containing that edge and extending into the region to be filled is constructed. This action leaves one or two new non-terminal regions within the boundary of the original region. (See Figure 3-13 for examples.)

(2) An edge is chosen to build on in one of the new non-terminal regions and the process is repeated.

(3) The process of filling the original region is completed when no new non-terminal regions remain to be filled.

This search procedure is exhaustive because it is known that each edge chosen to build on must be part of the boundary of a terminal region when the initial region is filled, and each possible type of terminal region is tried for that edge. The search procedure is non-redundant because each possible terminal region chosen for the

[10] In the version of GRAMPA which was used to run the examples for this thesis (see Chapter 4), only triangular terminal regions were tried at this point. The only time a four-sided terminal region was considered as a possibility was when a non-terminal region was itself four-sided. This meant that the solution search was not actually exhaustive. The degree to which this affected the example problems is discussed in Chapter 4.

edge that is being built on has at least one edge which intersects in a non-planar fashion at least one edge of every other possible terminel region chosen for the edge being built on. Since the region completion routine does not allow non-planar graphs this precludes the possibility of traversing the same branch of the search tree more than once.

The Attributes of a Move. - Each move on the move list has the following attributes:

(1) A list of the temporary design requirements associated with the move. -The details which are to be added to the dual graph in a given move are expressed in the same format as design requirements. There are three possible types of requirement lists for a move:

(a) A list containing a single node dimension requirement, in which a given pair of x and y dimensions is proposed for a room node.

(b) A list containing a single adjacency requirement.

(c) A list containing two adjacency requirements.

(2) A list called the alternative list which has for its attributes, among other things, a list of the remaining alternative requirements for the current move.

R106, the Move Selection Routine. - R106 considers the factors listed below in selecting which edge or node to work on. As with subroutine R119 of S2, variation of R106 can significantly influence the solution search strategy of Phase II of the program.

(1) If there still exists some non-terminal subregions of the region being filled in, R106 tries to find an undimensioned node or a weighted edge next to an outside space node to dimension or build on respectively. This is with the dual purpose of working in the area of highest information density and completing outside walls so the global well-formedness routines can be applied.

(2) Failing the above, R106 searches the non-terminal subregions for any weighted edge to build on.

(3) Failing the above, R106 searches for an edge next to a dimensioned node to build on.

(4) Failing the above, or if there were no non-terminal sub-regions left, R106 searches for non-dimensioned nodes left from the boundary of the original region. If there are any, it choses the one which has the maximum initial size (by area) on its allowable dimension list. This is an attempt to force feasibility contradictions as early as possible in the search.

3.3.2.3 R113, the Move-Trying Routine

Overview. - The paraphrased flowchart for routine R113 is given in Chart F31. Simply speaking, given a node move or a region move as an input, R113 generates the next alternative for that move and attempts to implement it.

Node and Edge Move Alternatives. - The alternatives for a node move are taken in order from the allowable dimension list of the node. This dimension list was formed from the input design requirements, and its order can influence the design search. If an input dimension requirement specified a size with unequal x and y dimensions, both orientations of that requirement (larger dimension as y dimension, and larger dimension as x dimension) appear on the alternative list.

The alternatives for an edge move are produced by generating the edges or edge pairs necessary to produce all possible terminal regions containing the edge being built on, and then generating all possible color-direction combinations for those edges or edge pairs. Note that one valid alternative for filling a nonterminal region that happens to have four sides is to consider it as a possible terminal four-sided region.

Trying a Node Move. - If the move under consideration is a node move and there exists an untried alternative, R113 uses routine A6 to assign that alternative dimension pair to the node under consideration. Next the dimension tests (routine A5, see Section 3.3.3) and the global realizability tests (routine A13 see Section 3.3.4) are applied, and if they are both passed the move is considered successful. There is no need to apply the topological realizability tests (routine A9) because A6 and A5 cannot do anything more than assign dimensions to nodes and weights to edges.

Trying an Edge Move. - If the move under consideration is an edge move, a plausibility test (routine R110) is first applied to see if the move should even be attempted. R110 looks ahead at the consequences of adding the proposed edge(s) to the graph and determines whether the introduced terminal region will be well-formed, and whether or not the move is attempting to connect the wrong colored or directed edges to an outside space node. These tests are rather inexpensive and can be applied before the edges are actually added to the dual graph. It takes in the order of one-tenth as much computer time to reject a move alternative using R110 as it would were the edges actually added to the dual graph.

If R110 is passed, routine R112 (see Chart F32) is used to attempt to add the edges to the dual graph. Since Phase II works on dual graph realizations containing only conventional regions (no pivot pair sets or tree structures are allowed, hence no hierarchical structuring of the graph is needed), R112 satisfies adjacency requirements by using routine R71 from Phase I of the program. R71 assumes all regions are conventional, and produces no pivot pair sets, even though one might be appropriate if the graph under consideration were being hierarchically structured. The fact that all non-terminal regions are filled on the inside removes the ambiguity of edge location that first necessitated the use of pivot pair sets. R112 can fail a proposed edge move on the basis that a proposed edge has already been generated previously by Z11 (see Section 3.2.3) and to repeat it would make the tree search redundant.

If R112 is passed, the local realizability tests (routine A12, see Chart F32) and the global realizability tests are applied. A12 fires routine A9, explained previously in Section 3.2.4, and A5.

Some Facts About the Well-Formedness Tests. - The various well-formedness tests, A5, A9, and A13, applied by R113 are used selectively. In the case of A5 and A9, the tests are applied only to those nodes or regions appearing on the various check lists L9, L10, and L15. Nodes and regions are added to these lists at various places in the program, whenever the tests might possibly be applicable. Then when the tests are fired by R113, they know which nodes and regions to work on. In general, the following rules are followed:

(1) Whenever a new region is created, or whenever the color or direction of an edge bounding a region is changed, that region is put on the region check list, L10.

(2) Whenever a new edge is attached to a node, or whenever the color or direction of an edge attached to a node is changed, that node is put on the node check list, L9.

(3) Whenever a node is dimensioned, or whenever a weight is assigned to an edge attached to a node, or whenever an edge is put on the node check list, that node is put on the dimension check list, L15.

In the case of A13, the entire battery of global realizability tests is applied when it is encountered in the program, if a special internal flag has been set. This flag is set by the program in the following circumstances:

(1) Every time a node is dimensioned.

(2) Every time an edge connected to an outside space node is assigned a weight.

(3) Every time a region bounded by one of the outside space nodes is created.

These circumstances are used to set the flag because the global tests are primarily concerned with the outside dimensions of the building and their compatibility with the overall area available for all the rooms. Section 3.3.4 provides more detail.

An important extra function of the various well-formedness tests is their ability to deduce certain information about the dual graph. The ability of A9 to deduce edge colors and directions was discussed previously. The ability of A5 to deduce edge weights will be discussed below.

Printing Protocol Information. - To keep the user informed as to what is going on during a given solution search, Phase II of GRAMPA prints a running account of each graph completion move that is tried and whether it was successful or not. The print routines which do this are scattered throughout R113 and its subroutines. Phase II also prints each new dual graph realization which is generated in R150 and accepted for attempted completion. Lastly, Phase II prints out final design solutions as they are generated. These solutions are printed in the form of a list description of the dual graph. This description can readily be transformed by hand into a floor plan drawing. However, a student, Robert McFarland, is currently writing an IPL-V program to translate the dual graph descriptions into line printer drawings of floor plans as a senior project in the Electrical Engineering Department at Carnegie-Mellon University.

3.3.3 The Dimension Tests

Overview. - The paraphrased flowchart for A5, the dimension deduction and checking routine, is given in Chart F33. Each node on the dimension check list, L15, is submitted to the tests. First each edge attached to the node is submitted to the outside space deduction routine, R85, and the adjacency deduction routine R86. These two routines can deduce the weight of an edge under certain conditions. They, in fact, along with R87, are the only mechanism that the program has for assigning weights to edges, since the dimension requirement satisfaction routine, A6, only assigns a given pair of dimensions to a node.

Next, R87, the dimension test routine, is applied to the node in question. R87 checks the dimension well-formedness of a node, and is also capable of some deduction of weights. Details on these routines are in the following paragraphs.

R85, the Outside Space Deduction Routine. - A special case of the rules for well-formed room nodes (see Section 2.4.2) applies to rooms adjacent to outside spaces. Since the outside space wall is the external wall of the building, a room adjacent to an outside space can be adjacent to no other spaces on that side. Thus all of the dimensional weight allowed to the node for that type of adjacency must be assigned to the edge representing the adjacency.

Routine R85 tests for this condition and assigns the weight to the edge if appropriate. If the node has not yet been dimensioned (which is possible), no edge weight can be assigned.

R86, the Adjacency Deduction Routine. - R86 relies on the physical property of the adjacency of rectangular rooms illustrated in Figure 3-14. Suppose a triangular region is encountered in the dual graph such as region A, B, C in Figure 3-14-A. Suppose, further, that edge AB already has a weight assigned to it. This means that the wall segment separating rooms A and B has already had its length specified. This in turn means that the length of the wall segment separating rooms A and C can be deduced, provided rooms A and C have already been assigned dimensions.

This can be done by calculating the amount of unassigned west wall length still available to Room A, call it D', and comparing this with the total east wall dimension assigned to room B, call it D. D' = 0 is a physically unrealizable condition (edge AC has to be assigned some weight), and no weight can be assigned to edge AC. (The rejection on the basis of physical unrealizability is left to routine R87, however.) If D' < D, the situation in Figure 3-14-A exists and the weight D - D' is assigned to edge AC. If D' = D, the situation in 3-14-B exists and the weight D is assigned to edge AC. If D' > D, the situation in 3-14-C exists, and D' - D is assigned as the weight of edge AC.

Routine R86 attempts this deduction procedure on every edge attached to the node under consideration.

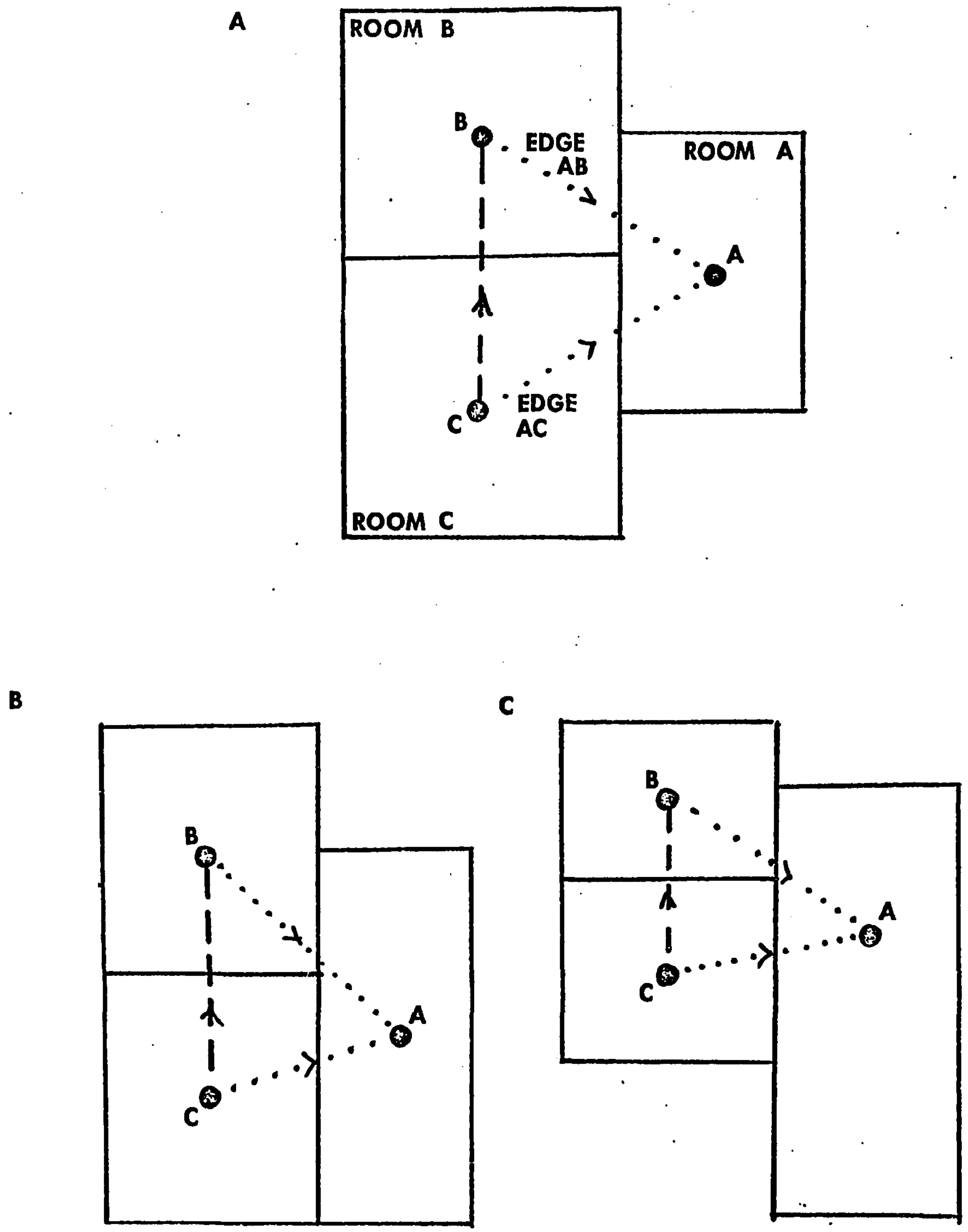


FIGURE 3-14

ILLUSTRATIONS FOR ROUTINE R86

R87, the Dimension Test Routine. - The basic purpose of R87 is to test whether the weights assigned to the edge of a given color and direction attached to a node (i.e., the edge corresponding to segments of one of the four walls of the room) fall within the bounds allowable by the current dimensions which have been assigned to that room node. R87 can also deduce an edge weight under certain circumstances.

For each of the types of colored and directed edges that can be attached to a node (north wall edges, east wall edges, south wall edges, and west wall edges) R87 performs the following steps:

(1) A list of the edges of the current color and direction combination is created.

(2) The sum of the weights assigned to these edges is calculated; call it S. (Not all edges will necessarily have weights.)

(3) This sum is compared with the assigned value for the room dimension (east-west or north-south) corresponding to the edge list under consideration. Call this dimension D.

(4) If $S > D$ the allowable dimension is exceeded and a failure is reported.

(5) If $S \leq D$ and there were no unweighted edges, then if it is impossible to add any more edges of the current color and direction to the node[11], S must equalD or a

[11] This condition exists if all edges on the list are bounded by terminal regions and if the two end bounding regions contain edges of a different color, direction combination attached to the node under consideration.

failure is reported.

(6) If $S \leq D$ and there were one or more unweighted edges on the list, then:

(a) If $D = S$ a failure is reported

(b) If $D > S$ and it is still possible to add more edges of the same color and direction to the node, success is reported.

(c) If $D > S$ and it is not possible to add more edges of the same color and direction to the node, then if only one unweighted edge exists on the list, it is assigned the weight $D - S$ and success is reported. Otherwise, no weight is assigned and success is reported.

Feasibility Condition F4-d. - Part 3 of feasibility condition F4-d requires that it must be impossible to construct a dimensionally well-formed node by adding detail to a node that is not dimensionally well formed. The failures in R87 are caused either by the existence of an edge for which there is no weight available, by the existence of weight for which there is no edge available, or by exceeding the allowable weight sum for an edge type. Since the program cannot assign negative weights to edges, none of the above conditions can be eliminated by adding new detail to the dual graph without changing its existing detail.

All Edges Get Assigned a Weight. - In the discussion of R157 the mop-up routine, it was stated that A5 is instrumental in guaranteeing that all edges of a graph that is accepted as completed have been assigned a weight (except for the four edges joining the outside space nodes to one another). This is justified by the following reasoning:

(1) As each node on the graph receives an information change pertinent to dimensioning, it is brought to the attention of A5.

(2) Before the final application of A5 by R160, all regions of the dual graph are made terminal, and all edges are assigned both a color and a direction. Thus, the graph is topologically complete.

(3) R160 insures that all room nodes of the graph are dimensioned.

(4) If a room node of a topologically complete dual graph has only one edge of a given color and direction connected to it and, if that node is dimensioned, then A5 assigns a weight to that edge equal to the corresponding dimension of the room node. This is done by the deduction capabilities of routines R85 and R87.

(5) It is up to routine R87 to deduce the edge weights of all those edges which do not satisfy the conditions of (4). A special characteristic of R87 is that it depends on at least one other edge having been weighted before it can deduce an edge weight. It can be shown that the edge weights deduced by

R85 and R87 are sufficient to allow R86 to deduce all remaining edge weights in a topologically completed dual graph. See Appendix A1.9 for details.

(6) Facts (1) through (5) taken together prove that A5 will have assigned all edge weights in a dual graph considered as complete.

3.3.4 The Global Realizability Tests

Overview. - The dimension tests and the topological realizability tests described in earlier sections generally operate on local structures of the dual graph. However, several of the most powerful search tree pruning tests of the program are concerned with the deminsions and overall area of the floor plan being designed. These are called the global realizability tests (or the global well-formedness tests) and the flowchart for their master routine, A13, is given in Chart F34. A13 merely generates the three types of global test for execution. One thing the three routines have in common is that if few room nodes are dimensioned, they can be quite time consuming. Thus, as was mentioned above, A13 will only fire the tests if the global test flag has been set elsewhere in the program. A brief description of each of the global test routines follows:

R153, the Outside Node Compatibility Test. - This test is designed to determine whether two opposite outside walls of the floor plan are being planned for the same length. If an outside space node is completely enclosed by terminal regions, it is considered to be completed. That is, no more room nodes can be connected to it in the present configuration of the dual graph. This accordingly means that the outside building wall corresponding to that node must have a total dimension which is the sum of the dimensions of the various rooms adjacent to that wall. In a partially completed dual graph, some of the room adjacency edges connected to a completed outside space node may be unweighted, since their corresponding nodes may not have been assigned dimensions yet. In this case a list of possible dimensions for the outside building wall may be generated by summing all possible combinations of weights which may be assigned to the edges incident to its corresponding outside space node. In the case of the unweighted edges, the possible weights are generated from the allowable dimension list of the corresponding room node.

R153 considers the outside space node pairs $\mathcal{N}$, $\mathcal{S}$ and $\mathcal{E}$, $\mathcal{W}$ respectively. For each pair, if both nodes in the par are completed nodes, R153 generates all possible wall dimensions for each node in the pair. If at least one dimension for the one wall is equal to at least one dimension for the opposite wall, the test passes. Otherwise the test fails because the building as projected could not possibly be rectangular.

R159, Test for Uncompleted Outside Nodes. - This test is quite similar to R153 in that it considers the building walls corresponding to opposite outside space nodes. However, it adds the capability of generating partial outside wall lengths for incomplete outside space nodes. Thus, if an incomplete outside space node exists opposite a complete one, if there exists no single alternative wall length for the incomplete node which is less than or equal to at least one alternative wall length of the opposite completed node, the test fails.

R158, the Area Test. - In general, this test determines whether or not there exists at least one combination of the currently assigned and unassigned room areas which is compatible with the overall dimensions of the building and any overall building area limits that may have been specified. The test is divided into three main parts.

The first part, subroutine R121, creates a list of all possible total areas that the building can have, based on its current dimensioned and undimensioned room nodes. For those nodes which are undimensioned, possible areas have to be generated from the associated list of alternative dimensions. Hence, several possible total areas for the building can exist. R121 checks each total area that is generated against upper and lower area limits for the building which were specified as design requirements. It is possible that no current possible area exists satisfying these limits, and in this case the test fails.

Next R158 checks each pair of adjacent outside space nodes and determines if both members of the pair are completed. If they are, R158 then generates all possible combinations of outside wall lengths for the pair of nodes under consideration and multiplies them together to create possible total areas for the building. These areas are compared with the areas previously calculated by R121 from the node dimensions, and if there is not at least one pair of areas that are equal, the test fails.

Finally, if there was not at least one adjacent pair of completed outside nodes to test, R158 searches for single completed outside space nodes. For each of these it then generates all possible outside wall lengths and divides these wall lengths into the areas from the list created by R121. If none of these divisions can be done without a remainder, the test fails, because this means there is not an exact combination of room dimensions available to exactly fill in the floor plan.

Feasibility Condition F4-d. - According to the last part of this condition, it must not be possible to generate a globally well-formed graph by only adding detail to a globally ill-formed graph. For the current version of GRAMPA this is true, because the program is not set up to receive additions to the dimension alternative list of a node while in the course of solving a problem. Hence the lists of possible lengths and areas cannot be enriched by adding such detail to the graph.

However, if it were possible to add and subtract dynamically room dimension alternatives while doing Phase II of the solution search, interesting refinements of the current search techniques would present themselves. These possibilities are discussed in Chapter 5.

4. DISCUSSION OF ILLUSTRATIVE DESIGN PROBLEMS

4.0. Introduction

To illustrate the capabilities, and limitations, of the design method, GRAMPA was used to solve several floor plan design problems. These will be discussed in this chapter.

A few general comments on how the examples were run and on the format of the computer output are necessary. The 200 or so basic IPL-V processes which are used in the writing of an IPL-V program are fairly low level. That is, they perform relatively rudimentary information processing tasks, such as adding an element to a list, locating an element on a list, adding two numbers, and so on. Nevertheless, because of the large amount of bookkeeping that the IPL-V interpretive system must do, the IBM 360 at Carnegie-Mellon University executes IPL-V basic processes at an average rate of about 5,000 per second, or about 200 microseconds per process. Coupling this with the fact that GRAMPA searches a problem solution space exhaustively leads to fairly long run times. The shortest example took 23 minutes to run, and the longest four hours.

IPL-V has a provision for interrupting and saving a partially run program on a disc data set and then restarting and continuing it at a later date. Thus all of the examples were run in 10 to 15 minute segments. Each segment produced output that could be used to check the progress of the solution before the program was restarted.

The output produced by the program is of two types. First, whenever a new realization of the dual graph is generated or whenever a design solution is achieved, a list structure description of the dual graph is printed out. The second type of output is printed whenever the program is attempting to complete a given realization. It consists of a statement for each move that is attempted, telling:

(1) the number of the move (counted from the very first move for the current problem), (T1),

(2) the number of IPL-V basic processes that have been executed since the problem began, (H3),

(3) the type of move (which node is being dimensioned or which edge is being built upon),

(4) the move itself, and

(5) whether the move was successful (if so, *SUCCESS* is printed).

In the examples, the average number of IPL-V basic processes executed per move varies between 5,000 and 10,000, i.e., it takes one to two seconds per move on the average. Since the examples take thousands of moves to reach a solution, it can easily be seen why they take so long to run. The program has been sampled while running and it has been found that the routines that consume most of the running time are the various well-formedness tests. The global tests in particular are expensive, because of the large number of length and area combinations they have to consider. Another routine that runs a good deal of the time is R100, which rebuilds the dual graph from a stored copy every time a search node is repeated. A possibly faster alternative would be to write a set of routines to unbuild the dual graph, edge by edge, to get back to a previous search node (see Chapter 5 for a discussion).

The routines that test the dual graph for planarity and construct it (Phase I of the program up to but not including the generation of realizations) work quite fast. Each of the dual graphs in the examples, resulting from the adjacency requirements alone, was generated in tens of seconds of computer time. Since the computation time increases approximately linearly with the size of the graph, Phase I might have applications in testing for the planarity of large graphs. Most algorithms familiar to the author for testing the planarity of a graph involve an effort that increases more than linearly with the size of the graph.

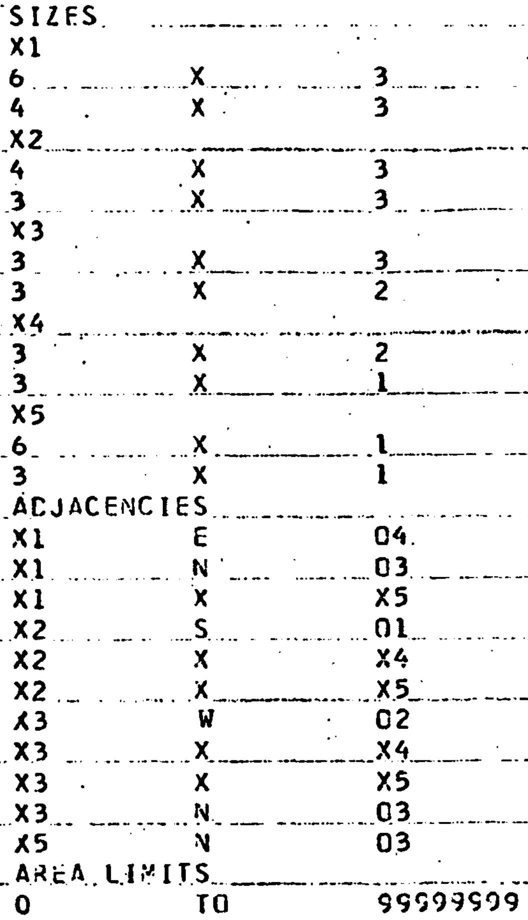
The decision whether the examples described here are either under-, well-, or over-specified (in the sense of Chapter 1) becomes clouded somewhat. This is because the requirement that the program be able to find all solutions, if they exist, in a reasonable amount of time is not satisfied in all the examples. In fact it has not been established just what constitutes a reasonable amount of time to spend on problems such as these. Since these examples were run for the purposes of research, it was feasible to set a time limit much higher than what would be allowed in a practical application. A discussion of what might be done to make GRAMPA applicable to practical design problems is given in Chapter 5.

The following sections describe the examples. Example 3 is the one in which an effort is made to determine if the omission of some possible four-sided regions by the program (as mentioned in Chapter 3) could have caused it to miss some solutions.

4.1. Example 1: Short, One Realization

The set of input requirements for Example 1 is given in Figure 4-1. The rooms are designated by X1, X2, X3, etc. Following each room under "SIZES" is a list of sizes that the room can have. Only one orientation of each is listed (X dimension first), but the program considers both orientation of each. The identifiers 01, 02, 03, 04 correspond to $\mathcal{N}$, $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{W}$ respectively. Under "ADJACENCIES" the letters, $\mathcal{N}$, $\mathcal{E}$, $\mathcal{S}$, $\mathcal{W}$, and X stand for "north of," "east of," "south of," "west of," and "next to" respectively.

From an architectural point of view, this set of design requirements could have been motivated in the following way: A client desires a house to be designed for him. This house should have a living room, X1, a bedroom, X2, a kitchen, X3, and a bathroom, X4. To satisfy the communication patterns within the house a corridor, X5, is also needed. The living room should either be 24' by 12' or 16' by 12' (these are scaled into 6 units by 3 units and 4 units by 3 units in the computer specifications); the bedroom should be 16' by 12' or 12' by 12', and so on. It is desired that the living room be on the southwest



```
SIZES
X1
6          X          3
4          X          3
X2
4          X          3
3          X          3
X3
3          X          3
3          X          2
X4
3          X          2
3          X          1
X5
6          X          1
3          X          1
ADJACENCIES
X1         E          04
X1         N          03
X1         X          X5
X2         S          01
X2         X          X4
X2         X          X5
X3         W          02
X3         X          X4
X3         X          X5
X3         N          03
X5         N          03
AREA LIMITS
0          TO         99999999
```

FIGURE 4-1

REQUIREMENTS FOR EXAMPLE 1

corner of the house and that it be connected with the corridor. This translates into the requirements X1 "east of" 04 (the west wall of the house), X1 "north of" 03 (the south wall), and X1 "next to" X5. These are the first three adjacency requirements input to the computer program. It is also desired that the bedroom be on the north side of the house and that it communicate with the corridor and the bathroom. This translates into the next three adjacency requirements of Figure 4-1, and so on. There is no limit on the area of the house, so the area limits are set from 0 to a very large number.

The dual graph that exists after Phase I of the program is finished is shown in Figure 4-2. The program takes 103,147 IPL-V interpretive cycles, or about 20 seconds, to complete Phase I for this example. Notice that only one dual graph realization exists for this example.

Figure 4-3 gives 18 pages of computer output describing the moves made by Phase II of the program as it generates the first solution to the example. A very good feel for how the program operates can be obtained by making an ink copy of Figure 4-2 and then tracing on it in pencil (so that they can be erased when the program backtracks) the various moves made by the program as it seeks the first solution. For the less curious reader, several "snapshots" of the trace are provided in Figures 4-4 through 4-6.

Figure 4-4 shows the progress of the search as of move 31. Here the program has generated an infeasible dual graph, but has not discovered this yet. It is in the process of filling region

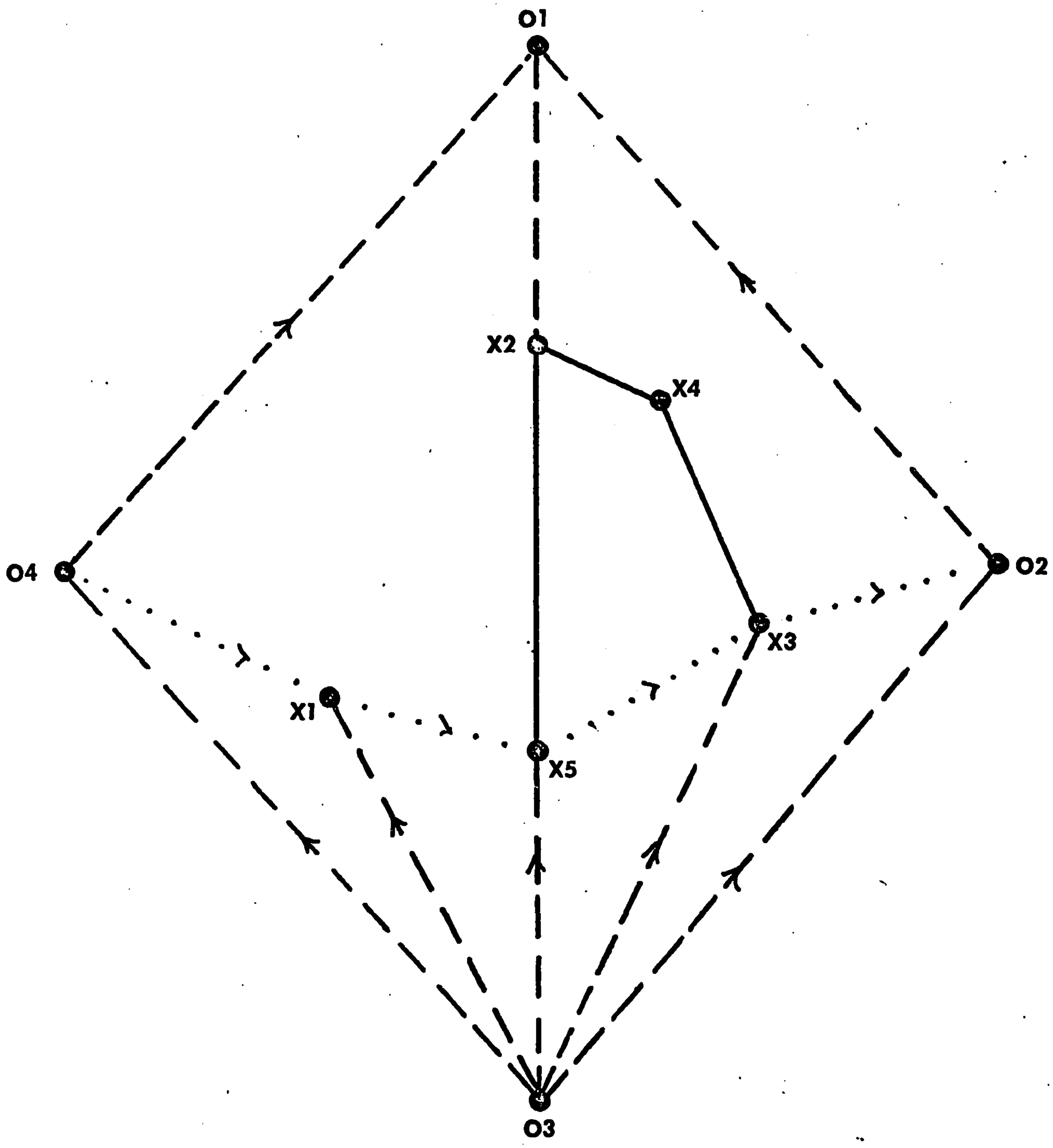
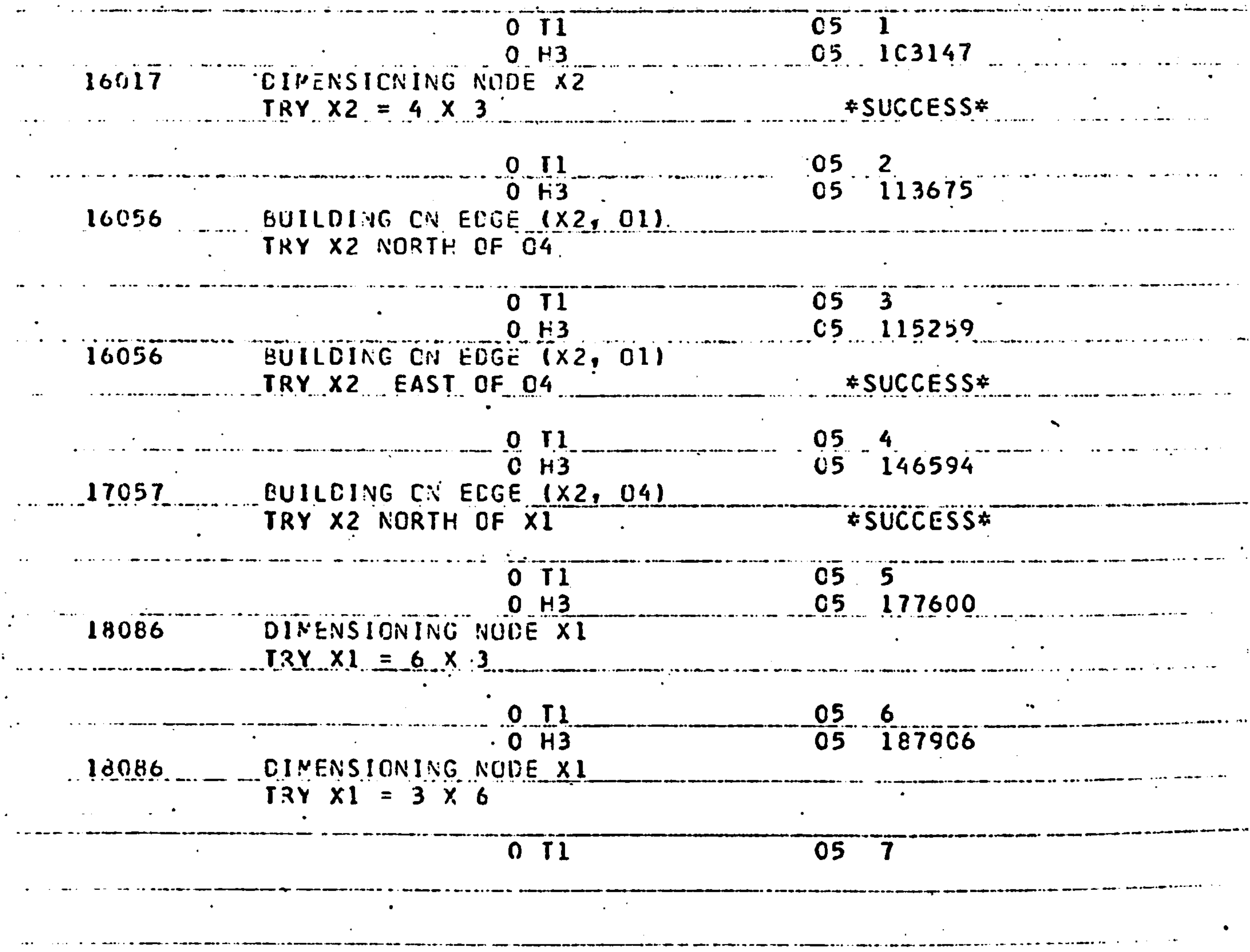


FIGURE 4-2

AT MOVE 1 IN EXAMPLE 1

FIGURE 4-3

TRACE FOR SOLUTION 1 OF EXAMPLE 1



```
                                     0 T1                  05  1
                                     0 H3                  05  103147
16017         DIMENSIONING NODE X2
              TRY X2 = 4 X 3                                 *SUCCESS*

                                     0 T1                  05  2
                                     0 H3                  05  113675
16056         BUILDING ON EDGE (X2, 01)
              TRY X2 NORTH OF 04

                                     0 T1                  05  3
                                     0 H3                  05  115259
16056         BUILDING ON EDGE (X2, 01)
              TRY X2  EAST OF 04                             *SUCCESS*

                                     0 T1                  05  4
                                     0 H3                  05  146594
17057         BUILDING ON EDGE (X2, 04)
              TRY X2 NORTH OF X1                             *SUCCESS*

                                     0 T1                  05  5
                                     0 H3                  05  177600
18086         DIMENSIONING NODE X1
              TRY X1 = 6 X 3

                                     0 T1                  05  6
                                     0 H3                  05  187906
18086         DIMENSIONING NODE X1
              TRY X1 = 3 X 6

                                     0 T1                  05  7
```

```
                               0 H3                     05  205174
          18086      DIMENSIONING NODE X1
                     TRY X1 = 4 X 3                       *SUCCESS*

                               0 T1                     C5  8
                               0 H3                     C5  218607
          13225      DIMENSIONING NODE X5
                     TRY X5 = 6 X 1

                               0 T1                     C5  9
                               C H3                     C5  225705
          13225      DIMENSIONING NODE X5
                     TRY X5 = 1 X 6                       *SUCCESS*

                               0 T1                     C5  10
                               0 H3                     C5  234929
          279892     DIMENSIONING NODE X3
                     TRY X3 = 3 X 3

                               0 T1                     C5  11
                               0 H3                     C5  244280
          279892     DIMENSIONING NODE X3
                     TRY X3 = 3 X 2

                               0 T1                     C5  12
                               0 H3                     C5  253641
          279892     DIMENSIONING NODE X3
                     TRY X3 = 2 X 3                       *SUCCESS*

                               0 T1                     C5  13
                               0 H3                     05  259650
          13448      BUILDING ON EDGE (X3, O2)
                     TRY X3 NORTH OF X2
                     TRY O2 NORTH OF X2

                               0 T1                     C5  14
                               0 H3                     05  261884
          13448      BUILDING ON EDGE (X3, O2)
                     TRY X3 NORTH OF X2
                     TRY O2  EAST OF X2

                               0 T1                     C5  15
                               0 H3                     C5  264130
          13448      BUILDING ON EDGE (X3, O2)
                     TRY X3 NORTH OF X2
                     TRY O2 SOUTH OF X2

                               0 T1                     05  16
                               0 H3                     05  265395
          13448      BUILDING ON EDGE (X3, O2)
                     TRY X3 NORTH OF X2
                     TRY O2  WEST OF X2

                               0 T1                     C5  17
                               C H3                     05  266762
          13448      BUILDING ON EDGE (X3, O2)
                     TRY X3  EAST OF X2
                     TRY O2 NORTH OF X2
```

```
                                   0 T1                         05  18
                                   0 H3                         05  263144
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3  EAST OF X2
                    TRY 02  EAST OF X2

                                   0 T1                         05  19
                                   0 H3                         05  271197
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3  EAST OF X2
                    TRY 02 SOUTH OF X2

                                   0 T1                         05  20
                                   0 H3                         05  272462
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3  EAST OF X2
                    TRY 02  WEST OF X2

                                   0 T1                         05  21
                                   0 H3                         05  273829
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3 SOUTH OF X2
                    TRY 02 NORTH OF X2

                                   0 T1                         05  22
                                   0 H3                         05  275211
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3 SOUTH OF X2
                    TRY 02  EAST OF X2

                                   0 T1                         05  23
                                   0 H3                         05  315330
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3 SOUTH OF X2
                    TRY 02 SOUTH OF X2

                                   0 T1                         05  24
                                   0 H3                         05  316645
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3 SOUTH OF X2
                    TRY 02  WEST OF X2

                                   0 T1                         05  25
                                   0 H3                         05  318012
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3  WEST OF X2
                    TRY 02 NORTH OF X2

                                   0 T1                         05  26
                                   0 H3                         05  319394
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3  WEST OF X2
                    TRY 02  EAST OF X2

                                   0 T1                         05  27
                                   0 H3                         05  322729
        13448       BUILDING ON EDGE (X3, 02)
                    TRY X3  WEST OF X2
```

```
              TRY 02 SOUTH OF X2

                                  0 T1                   05  28
                                  0 H3                   05  323994
  13448       BUILDING ON EDGE (X3, 02)
              TRY X3  WEST OF X2
              TRY 02  WEST OF X2

                                  0 T1                   05  29
                                  0 H3                   05  325361
  13448       BUILDING ON EDGE (X3, 02)
              TRY 02 NORTH OF X4

                                  0 T1                   05  30
                                  0 H3                   05  327213
  13448       BUILDING ON EDGE (X3, 02)
              TRY 02  EAST OF X4                            *SUCCESS*

                                  0 T1                   05  31
                                  0 H3                   05  373106
  12131       DIMENSIONING NODE X4
              TRY X4 = 3 X 2                                *SUCCESS*

                                  0 T1                   05  32
                                  0 H3                   05  377913
  281539      BUILDING ON EDGE (02, X4)
              TRY X4 NORTH OF 01

                                  0 T1                   05  33
                                  0 H3                   05  379019
  281539      BUILDING ON EDGE (02, X4)
              TRY X4  EAST OF 01

                                  0 T1                   05  34
                                  0 H3                   05  379625
  281539      BUILDING ON EDGE (02, X4)
              TRY X4 SOUTH OF 01

                                  0 T1                   05  35
                                  0 H3                   05  427914
  281539      BUILDING ON EDGE (02, X4)
              TRY X4  WEST OF 01

                                  0 T1                   05  36
                                  0 H3                   05  428520
  281539      BUILDING ON EDGE (02, X4)
              TRY 02 NORTH OF X2

                                  0 T1                   05  37
                                  0 H3                   05  429530
  281539      BUILDING ON EDGE (02, X4)
              TRY 02  EAST OF X2

                                  0 T1                   05  38
                                  0 H3                   05  488436
  281539      BUILDING ON EDGE (02, X4)
              TRY 02 SOUTH OF X2
```

```
                                  0 T1                05  39
                                  0 H3                05  489051
   281539     BUILDING ON EDGE (02, X4)
              TRY 02  WEST OF X2

                                  0 T1                05  40
                                  0 H3                05  489666
   281539     BUILDING ON EDGE (02, X4)
              ALTERNATIVES EXHAUSTED

                                  0 T1                05  41
                                  0 H3                05  495253
   12131      DIMENSIONING NODE X4
              TRY X4 = 2 X 3                             *SUCCESS*

                                  0 T1                05  42
                                  0 H3                05  500070
   282792     BUILDING ON EDGE (02, X4)
              TRY X4 NORTH OF 01

                                  0 T1                05  43
                                  0 H3                05  501176
   282792     BUILDING ON EDGE (02, X4)
              TRY X4  EAST OF 01

                                  0 T1                05  44
                                  0 H3                05  501782
   282792     BUILDING ON EDGE (02, X4)
              TRY X4 SOUTH OF 01

                                  0 T1                05  45
                                  0 H3                05  549022
   282792     BUILDING ON EDGE (02, X4)
              TRY X4  WEST OF 01

                                  0 T1                05  46
                                  0 H3                05  549628
   282792     BUILDING ON EDGE (02, X4)
              TRY 02 NORTH OF X2

                                  0 T1                05  47
                                  0 H3                05  550638
   282792     BUILDING ON EDGE (02, X4)
              TRY 02  EAST OF X2

                                  0 T1                05  48
                                  0 H3                05  609544
   282792     BUILDING ON EDGE (02, X4)
              TRY 02 SOUTH OF X2

                                  0 T1                05  49
                                  0 H3                05  610159
   282792     BUILDING ON EDGE (02, X4)
              TRY 02  WEST OF X2

                                  0 T1                05  50
                                  0 H3                05  610774
   282792     BUILDING ON EDGE (02, X4)
```

```
                 ALTERNATIVES EXHAUSTED

                                     0 T1                   05  51
                                     0 H3                   05  616361
        12131    DIMENSIONING NODE X4
                 TRY X4 = 3 X 1

                                     0 T1                   05  52
                                     0 H3                   05  625117
        12131    DIMENSIONING NODE X4
                 TRY X4 = 1 X 3

                                     0 T1                   05  53
                                     0 H3                   05  633873
        12131    DIMENSIONING NODE X4
                 ALTERNATIVES EXHAUSTED

                                     0 T1                   05  54
                                     0 H3                   05  638765
        13448    BUILDING ON EDGE (X3, 02)
                 TRY 02 SOUTH OF X4

                                     0 T1                   05  55
                                     0 H3                   05  639706
        13448    BUILDING ON EDGE (X3, 02)
                 TRY 02  WEST OF X4

                                     0 T1                   05  56
                                     0 H3                   05  640647
        13448    BUILDING ON EDGE (X3, 02)
                 TRY X3 NORTH OF 01

                                     0 T1                   05  57
                                     0 H3                   05  642373
        13448    BUILDING ON EDGE (X3, 02)
                 TRY X3  EAST OF 01

                                     0 T1                   05  58
                                     0 H3                   05  643305
        13448    BUILDING ON EDGE (X3, 02)
                 TRY X3 SOUTH OF 01

                                     0 T1                   05  59
                                     0 H3                   05  666913
        13448    BUILDING ON EDGE (X3, 02)
                 TRY X3  WEST OF 01

                                     0 T1                   05  60
                                     0 H3                   05  667845
        13448    BUILDING ON EDGE (X3, 02)
                 ALTERNATIVES EXHAUSTED

                                     0 T1                   05  61
                                     0 H3                   05  673158
        279892   DIMENSIONING NODE X3
                 ALTERNATIVES EXHAUSTED

                                     0 T1                   05  62
```

```
                                    0 H3                    05  678023
      13225        DIMENSIONING NODE X5
                   TRY X5 = 3 X 1

                                    0 T1                    05  63
                                    0 H3                    05  685131
      13225        DIMENSIONING NODE X5
                   TRY X5 = 1 X 3                             *SUCCESS*

                                    0 T1                    05  64
                                    0 H3                    05  694357
      20668        DIMENSIONING NODE X3
                   TRY X3 = 3 X 3

                                    0 T1                    05  65
                                    0 H3                    05  703708
      20668        DIMENSIONING NODE X3
                   TRY X3 = 3 X 2

                                    0 T1                    05  66
                                    0 H3                    05  713069
      20668        DIMENSIONING NODE X3
                   TRY X3 = 2 X 3

                                    0 T1                    05  67
                                    0 H3                    05  722430
      20668        DIMENSIONING NODE X3
                   ALTERNATIVES EXHAUSTED

                                    0 T1                    05  68
                                    0 H3                    05  727295
      13225        DIMENSIONING NODE X5
                   ALTERNATIVES EXHAUSTED

                                    0 T1                    05  69
                                    0 H3                    05  732039
      18086        DIMENSIONING NODE X1
                   TRY X1 = 3 X 4                             *SUCCESS*

                                    0 T1                    05  70
                                    0 H3                    05  745472
      13428        DIMENSIONING NODE X5
                   TRY X5 = 6 X 1

                                    0 T1                    05  71
                                    0 H3                    05  752570
      13428        DIMENSIONING NODE X5
                   TRY X5 = 1 X 6                             *SUCCESS*

                                    0 T1                    05  72
                                    0 H3                    05  761794
      13105        DIMENSIONING NODE X3
                   TRY X3 = 3 X 3

                                    0 T1                    05  73
                                    0 H3                    05  771145
      13105        DIMENSIONING NODE X3
                   TRY X3 = 3 X 2
```

```
                                   0 T1                     C5  74
                                   0 H3                     C5  780506
        13105          DIMENSIONING NODE X3
                       TRY X3 = 2 X 3                          *SUCCESS*

                                   0 T1                     C5  75
                                   0 H3                     C5  786515
        23009          BUILDING CN EDGE (X3, 02)
                       TRY X3 NORTH CF X2
                       TRY 02 NORTH CF X2

                                   0 T1                     C5  76
                                   0 H3                     C5  788749
        23009          BUILDING CN EDGE (X3, 02)
                       TRY X3 NORTH CF X2
                       TRY 02  EAST CF X2

                                   0 T1                     C5  77
                                   0 H3                     05  790995
        23009          BUILDING CN EDGE (X3, 02)
                       TRY X3 NORTH OF X2
                       TRY 02 SOUTH OF X2

                                   0 T1                     C5  78
                                   0 H3                     C5  792260
        23009          BUILCING CN EDGE (X3, 02)
                       TRY X3 NORTH OF X2
                       TRY 02  WEST OF X2

                                   0 T1                     C5  79
                                   0 H3                     C5  793627
        23009          BUILCING ON ECGE (X3, 02)
                       TRY X3  EAST OF X2
                       TRY 02 NORTH CF X2

                                   0 T1                     C5  80
                                   0 H3                     05  795C09
        23009          BUILDING CN ECGE (X3, 02)
                       TRY X3  EAST OF X2
                       TRY 02  EAST CF X2

                                   0 T1                     C5  81
                                   0 H3                     05  798062
        23009          BUILDING CN EDGE (X3, 02)
                       TRY X3  EAST CF X2
                       TRY 02 SOUTH CF X2

                                   0 T1                     05  82
                                   0 H3                     C5  799327
        23009          BUILDING CN EDGE (X3, 02)
                       TRY X3  EAST CF X2
                       TRY 02  WEST OF X2

                                   0 T1                     C5  83
                                   0 H3                     C5  800694
        23009          BUILDING CN EDGE (X3, 02)
                       TRY X3 SOUTH CF X2
                       TRY 02 NORTH CF X2
```

```
                                 0 T1                   05  84
                                 0 H3                   05  802076
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3 SOUTH OF X2
                     TRY 02  EAST OF X2

                                 0 T1                   05  85
                                 0 H3                   05  842195
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3 SOUTH OF X2
                     TRY 02 SOUTH OF X2

                                 0 T1                   05  86
                                 0 H3                   05  843510
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3 SOUTH OF X2
                     TRY 02  WEST OF X2

                                 0 T1                   05  87
                                 0 H3                   05  844877
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3  WEST OF X2
                     TRY 02 NORTH OF X2

                                 0 T1                   05  88
                                 0 H3                   05  846259
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3  WEST OF X2
                     TRY 02  EAST OF X2

                                 0 T1                   05  89
                                 0 H3                   05  849594
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3  WEST OF X2
                     TRY 02 SOUTH OF X2

                                 0 T1                   05  90
                                 0 H3                   05  850859
       23009         BUILDING ON EDGE (X3, 02)
                     TRY X3  WEST OF X2
                     TRY 02  WEST OF X2

                                 0 T1                   05  91
                                 0 H3                   05  852226
       23009         BUILDING ON EDGE (X3, 02)
                     TRY 02 NORTH OF X4

                                 0 T1                   05  92
                                 0 H3                   05  854078
       23009         BUILDING ON EDGE (X3, 02)
                     TRY 02  EAST OF X4                    *SUCCESS*

                                 0 T1                   05  93
                                 0 H3                   05  899971
       282080        DIMENSIONING NODE X4
                     TRY X4 = 3 X 2                        *SUCCESS*

                                 0 T1                   05  94
```

```
                                  0 H3                   05  904778
       283668       BUILDING ON EDGE (02, X4)
                    TRY X4 NORTH OF 01

                                  0 T1                   05  95
                                  0 H3                   05  905884
       283668       BUILDING ON EDGE (02, X4)
                    TRY X4  EAST OF 01

                                  0 T1                   05  96
                                  0 H3                   05  906490
       283668       BUILDING ON EDGE (02, X4)
                    TRY X4 SOUTH OF 01

                                  0 T1                   05  97
                                  0 H3                   05  953730
       283668       BUILDING ON EDGE (02, X4)
                    TRY X4  WEST OF 01

                                  0 T1                   05  98
                                  0 H3                   05  954336
       283668       BUILDING ON EDGE (02, X4)
                    TRY 02 NORTH OF X2

                                  0 T1                   05  99
                                  0 H3                   05  955346
       283668       BUILDING ON EDGE (02, X4)
                    TRY 02  EAST OF X2

                                  0 T1                   05  100
                                  0 H3                   05  1014617
       283668       BUILDING ON EDGE (02, X4)
                    TRY 02 SOUTH OF X2

                                  0 T1                   05  101
                                  0 H3                   05  1015232
       283668       BUILDING ON EDGE (02, X4)
                    TRY 02  WEST OF X2

                                  0 T1                   05  102
                                  0 H3                   05  1015847
       283668       BUILDING ON EDGE (02, X4)
                    ALTERNATIVES EXHAUSTED

                                  0 T1                   05  103
                                  0 H3                   05  1021434
       282080       DIMENSIONING NODE X4
                    TRY X4 = 2 X 3                          *SUCCESS*

                                  0 T1                   05  104
                                  0 H3                   05  1026251
       282496       BUILDING ON EDGE (02, X4)
                    TRY X4 NORTH OF 01

                                  0 T1                   05  105
                                  0 H3                   05  1027357
       282496       BUILDING ON EDGE (02, X4)
                    TRY X4  EAST OF 01
```

```
                                  0 T1                 05   106
                                  0 H3                 05   1027963
     282496      BUILDING ON EDGE (O2, X4)
                 TRY X4 SOUTH OF O1

                                  0 T1                 05   107
                                  0 H3                 05   1076252
     282496      BUILDING ON EDGE (O2, X4)
                 TRY X4  WEST OF O1

                                  0 T1                 05   108
                                  0 H3                 05   1076858
     282496      BUILDING ON EDGE (O2, X4)
                 TRY O2 NORTH OF X2

                                  0 T1                 05   109
                                  0 H3                 05   1077868
     282496      BUILDING ON EDGE (O2, X4)
                 TRY O2  EAST OF X2

                                  0 T1                 05   110
                                  0 H3                 05   1137139
     282496      BUILDING ON EDGE (O2, X4)
                 TRY O2 SOUTH OF X2

                                  0 T1                 05   111
                                  0 H3                 05   1137754
     282496      BUILDING ON EDGE (O2, X4)
                 TRY O2  WEST OF X2

                                  0 T1                 05   112
                                  0 H3                 05   1138369
     282496      BUILDING ON EDGE (O2, X4)
                 ALTERNATIVES EXHAUSTED

                                  0 T1                 05   113
                                  0 H3                 05   1143956
     282080      DIMENSIONING NODE X4
                 TRY X4 = 3 X 1

                                  0 T1                 05   114
                                  0 H3                 05   1152712
     282080      DIMENSIONING NODE X4
                 TRY X4 = 1 X 3

                                  0 T1                 05   115
                                  0 H3                 05   1161468
     282080      DIMENSIONING NODE X4
                 ALTERNATIVES EXHAUSTED

                                  0 T1                 05   116
                                  0 H3                 05   1166360
     23009       BUILDING ON EDGE (X3, O2)
                 TRY O2 SOUTH OF X4

                                  0 T1                 05   117
                                  0 H3                 05   1167301
     23009       BUILDING ON EDGE (X3, O2)
```

```
                TRY 02  WEST OF X4

                               0 T1                 05  118
                               0 H3                 05  1168242
     23009      BUILDING ON EDGE (X3, 02)
                TRY X3 NORTH OF 01

                               0 T1                 05  119
                               0 H3                 05  1169968
     23009      BUILDING ON EDGE (X3, 02)
                TRY X3  EAST OF 01

                               0 T1                 05  120
                               0 H3                 05  1170900
     23009      BUILDING ON EDGE (X3, 02)
                TRY X3 SOUTH OF 01

                               0 T1                 05  121
                               0 H3                 05  1194508
     23009      BUILDING ON EDGE (X3, 02)
                TRY X3  WEST OF 01

                               0 T1                 05  122
                               0 H3                 05  1195440
     23009      BUILDING ON EDGE (X3, 02)
                ALTERNATIVES EXHAUSTED

                               0 T1                 05  123
                               0 H3                 05  1200753
     13105      DIMENSIONING NODE X3
                ALTERNATIVES EXHAUSTED

                               0 T1                 05  124
                               0 H3                 05  1205618
     13428      DIMENSIONING NODE X5
                TRY X5 = 3 X 1

                               0 T1                 05  125
                               0 H3                 05  1212726
     13428      DIMENSIONING NODE X5
                TRY X5 = 1 X 3

                               0 T1                 05  126
                               0 H3                 05  1219834
     13428      DIMENSIONING NODE X5
                ALTERNATIVES EXHAUSTED

                               0 T1                 05  127
                               0 H3                 05  1224578
     18086      DIMENSIONING NODE X1
                ALTERNATIVES EXHAUSTED

                               0 T1                 05  128
                               0 H3                 05  1229060
     17057      BUILDING ON EDGE (X2, 04)
                TRY X2  EAST OF X1

                               0 T1                 05  129
```

```
                                 0 H3                    05  1231631
17057      BUILDING ON EDGE (X2, 04)
           TRY X2 SOUTH OF X1

                                 0 T1                    05  130
                                 0 H3                    05  1233136
17057      BUILDING ON EDGE (X2, 04)
           TRY X2  WEST OF X1

                                 0 T1                    05  131
                                 0 H3                    05  1235432
17057      BUILDING ON EDGE (X2, 04)
           TRY 04 NORTH OF X5

                                 0 T1                    05  132
                                 0 H3                    05  1236412
17057      BUILDING ON EDGE (X2, 04)
           TRY 04  EAST OF X5

                                 0 T1                    05  133
                                 0 H3                    05  1237024
17057      BUILDING ON EDGE (X2, 04)
           TRY 04 SOUTH OF X5

                                 0 T1                    05  134
                                 0 H3                    05  1237636
17057      BUILDING ON EDGE (X2, 04)
           TRY 04  WEST OF X5

                                 0 T1                    05  135
                                 0 H3                    05  1251564
17057      BUILDING ON EDGE (X2, 04)
           ALTERNATIVES EXHAUSTED

                                 0 T1                    05  136
                                 0 H3                    05  1256250
16056      BUILDING ON EDGE (X2, 01)
           TRY X2 SOUTH OF 04

                                 0 T1                    05  137
                                 0 H3                    05  1257085
16056      BUILDING ON EDGE (X2, 01)
           TRY X2  WEST OF 04

                                 0 T1                    05  138
                                 0 H3                    05  1257920
16056      BUILDING ON EDGE (X2, 01)
           TRY X2 NORTH OF X1
           TRY 01 NORTH OF X1

                                 0 T1                    05  139
                                 0 H3                    05  1261599
16056      BUILDING ON EDGE (X2, 01)
           TRY X2 NORTH OF X1
           TRY 01  EAST OF X1

                                 0 T1                    05  140
                                 0 H3                    05  1262795
```

```
   16056        BUILDING ON EDGE (X2, 01)
                TRY X2 NORTH OF X1
                TRY 01 SOUTH OF X1

                                 0 T1                05  141
                                 0 H3                05  1264093
   16056        BUILDING ON EDGE (X2, 01)
                TRY X2 NORTH OF X1
                TRY 01  WEST OF X1

                                 0 T1                05  142
                                 0 H3                05  1265391
   16056        BUILDING ON EDGE (X2, 01)
                TRY X2  EAST OF X1
                TRY 01 NORTH OF X1                      *SUCCESS*

                                 0 T1                05  143
                                 0 H3                05  1316041
   17122        DIMENSIONING NODE X1
                TRY X1 = 6 X 3

                                 0 T1                05  144
                                 0 H3                05  1323603
   17122        DIMENSIONING NODE X1
                TRY X1 = 3 X 6                          *SUCCESS*

                                 0 T1                05  145
                                 0 H3                05  1334655
   279549       DIMENSIONING NODE X5
                TRY X5 = 6 X 1

                                 0 T1                05  146
                                 0 H3                05  1341945
   279549       DIMENSIONING NODE X5
                TRY X5 = 1 X 6

                                 0 T1                05  147
                                 0 H3                05  1349695
   279549       DIMENSIONING NODE X5
                TRY X5 = 3 X 1

                                 0 T1                05  148
                                 0 H3                05  1356995
   279549       DIMENSIONING NODE X5
                TRY X5 = 1 X 3                          *SUCCESS*

                                 0 T1                05  149
                                 0 H3                05  1366106
   20932        DIMENSIONING NODE X3
                TRY X3 = 3 X 3

                                 0 T1                05  150
                                 0 H3                05  1375365
   20932        DIMENSIONING NODE X3
                TRY X3 = 3 X 2                          *SUCCESS*

                                 0 T1                05  151
                                 0 H3                05  1381223
```

```
21075          BUILDING ON EDGE (X3, O2)
               TRY X3 NORTH OF C1

                                  0 T1                05  152
                                  0 H3                05  1383032
21075          BUILDING ON EDGE (X3, O2)
               TRY X3  EAST OF C1

                                  0 T1                05  153
                                  0 H3                05  1383979
21075          BUILDING ON EDGE (X3, O2)
               TRY X3 SOUTH OF C1

                                  0 T1                05  154
                                  0 H3                05  1408375
21075          BUILDING ON EDGE (X3, O2)
               TRY X3  WEST OF C1

                                  0 T1                05  155
                                  0 H3                05  1409322
21075          BUILDING ON EDGE (X3, O2)
               TRY X3 NORTH OF X2
               TRY O2 NORTH OF X2

                                  0 T1                05  156
                                  0 H3                05  1411560
21075          BUILDING ON EDGE (X3, O2)
               TRY X3 NORTH OF X2
               TRY O2  EAST OF X2

                                  0 T1                05  157
                                  0 H3                05  1413346
21075          BUILDING ON EDGE (X3, O2)
               TRY X3 NORTH OF X2
               TRY O2 SOUTH OF X2

                                  0 T1                05  158
                                  0 H3                05  1415151
21075          BUILDING ON EDGE (X3, O2)
               TRY X3 NORTH OF X2
               TRY O2  WEST OF X2

                                  0 T1                05  159
                                  0 H3                05  1416558
21075          BUILDING ON EDGE (X3, O2)
               TRY X3  EAST OF X2
               TRY O2 NORTH OF X2

                                  0 T1                05  160
                                  0 H3                05  1417980
21075          BUILDING ON EDGE (X3, O2)
               TRY X3  EAST OF X2
               TRY O2  EAST OF X2

                                  0 T1                05  161
                                  0 H3                05  1421073
21075          BUILDING ON EDGE (X3, O2)
               TRY X3  EAST OF X2
```

```
                  TRY 02 SOUTH OF X2

                                       0 T1                 05  162
                                       0 H3                 05  1422378
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3  EAST OF X2
                  TRY 02  WEST OF X2

                                       0 T1                 05  163
                                       0 H3                 05  1423785
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3 SOUTH OF X2
                  TRY 02 NORTH OF X2

                                       0 T1                 05  164
                                       0 H3                 05  1425207
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3 SOUTH OF X2
                  TRY 02  EAST OF X2

                                       0 T1                 05  165
                                       0 H3                 05  1463615
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3 SOUTH OF X2
                  TRY 02 SOUTH OF X2

                                       0 T1                 05  166
                                       0 H3                 05  1464945
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3 SOUTH OF X2
                  TRY 02  WEST OF X2

                                       0 T1                 05  167
                                       0 H3                 05  1466352
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3  WEST OF X2
                  TRY 02 NORTH OF X2

                                       0 T1                 05  168
                                       0 H3                 05  1467774
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3  WEST OF X2
                  TRY 02  EAST OF X2

                                       0 T1                 05  169
                                       0 H3                 05  1471149
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3  WEST OF X2
                  TRY 02 SOUTH OF X2

                                       0 T1                 05  170
                                       0 H3                 05  1472454
        21075     BUILDING ON EDGE (X3, 02)
                  TRY X3  WEST OF X2
                  TRY 02  WEST OF X2

                                       0 T1                 05  171
                                       0 H3                 05  1473861
```

```
21075          BUILDING ON EDGE (X3, O2)
               TRY O2 NORTH OF X4

                                  0 T1                  05  172
                                  0 H3                  05  1475793
21075          BUILDING ON EDGE (X3, O2)
               TRY O2  EAST OF X4                        *SUCCESS*

                                  0 T1                  05  173
                                  0 H3                  05  1527191
21487          DIMENSIONING NODE X4
               TRY X4 = 3 X 2

                                  0 T1                  05  174
                                  0 H3                  05  1535845
21487          DIMENSIONING NODE X4
               TRY X4 = 2 X 3

                                  0 T1                  05  175
                                  0 H3                  05  1544509
21487          DIMENSIONING NODE X4
               TRY X4 = 3 X 1                            *SUCCESS*

                                  0 T1                  05  176
                                  0 H3                  05  1549176
21874          BUILDING ON EDGE (O2, X4)
               TRY X4 NORTH OF O1

                                  0 T1                  05  177
                                  0 H3                  05  1550282
21874          BUILDING ON EDGE (O2, X4)
               TRY X4  EAST OF O1

                                  0 T1                  05  178
                                  0 H3                  05  1550888
21874          BUILDING ON EDGE (O2, X4)
               TRY X4 SOUTH OF O1

                                  0 T1                  05  179
                                  0 H3                  05  1591786
21874          BUILDING ON EDGE (O2, X4)
               TRY X4  WEST OF O1

                                  0 T1                  05  180
                                  0 H3                  05  1592392
21874          BUILDING ON EDGE (O2, X4)
               TRY O2 NORTH OF X2

                                  0 T1                  05  181
                                  0 H3                  05  1593402
21874          BUILDING ON EDGE (O2, X4)
               TRY O2  EAST OF X2                        *SUCCESS*

                                  0 T1                  05  182
                                  0 H3                  05  1637183
286476         BUILDING ON EDGE (X3, X5)
               TRY X5 NORTH OF X4
```

```
                         0 T1            05  183
                         0 H3            05  1639778
   286476     BUILDING ON EDGE (X3, X5)
              TRY X5  EAST OF X4

                         0 T1            05  184
                         0 H3            05  1641579
   286476     BUILDING ON EDGE (X3, X5)
              TRY X5 SOUTH OF X4

                         0 T1            05  185
                         0 H3            05  1662300
   286476     BUILDING ON EDGE (X3, X5)
              TRY X5  WEST OF X4               *SUCCESS*
```

(01, 02, X4, X2) and is building on edge (02, X4). However, it will not be able to do this, as can be seen from the trace, because any edge that can be constructed satisfying the topological rules of well-formedness will cause different dimensions for at least one pair of opposing outside walls of the building.

By move 148 the program is on the right track again, as can be shown in Figure 4-5. The region (01, X2, X5, X1, 04) has been completed and regions (01, X2, X3, X4, X2) and (X2, X4, X3, X5) await completion. Finally, at move 185 (1,662,300 interpretation cycles, or about 5 1/2 minutes into the run) the first design solution is reached, producing the completely specified dual graph shown in Figure 4-6. The floor plan corresponding to this solution is shown in Figure 4-7-A.

The other four solutions to this problem are shown in Figure 4-7. As noted on the Figure, the search for solutions was exhausted after 848 moves, or 6,838,109 interpretive cycles, or almost exactly 23 minutes of computer time. The average number of cycles per move was 8,100 for this problem, or about 1.6 seconds per move.

4.2. Example 2: Longer, Several Realizations

The design requirements are given in Figure 4-8. The dual graph that exists after all adjacency requirements are satisfied, but before the realizations are generated, is shown in Figure 4-9. The main thing which distinguishes this example from Example 1 is the fact that there exist several realizations for this graph instead of just one. It was necessary to draw one

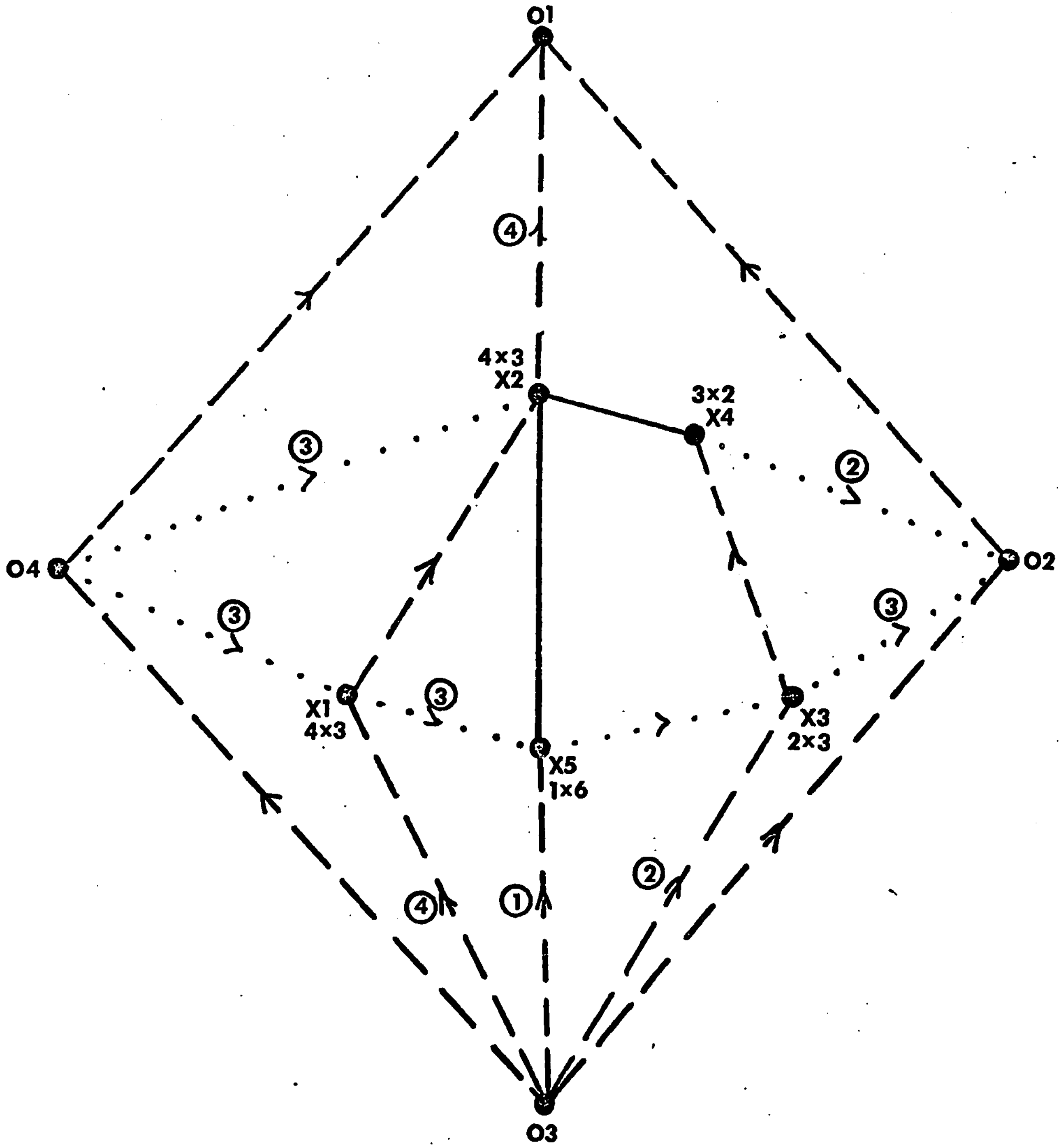


FIGURE 4-4

AT MOVE 31 IN EXAMPLE 1

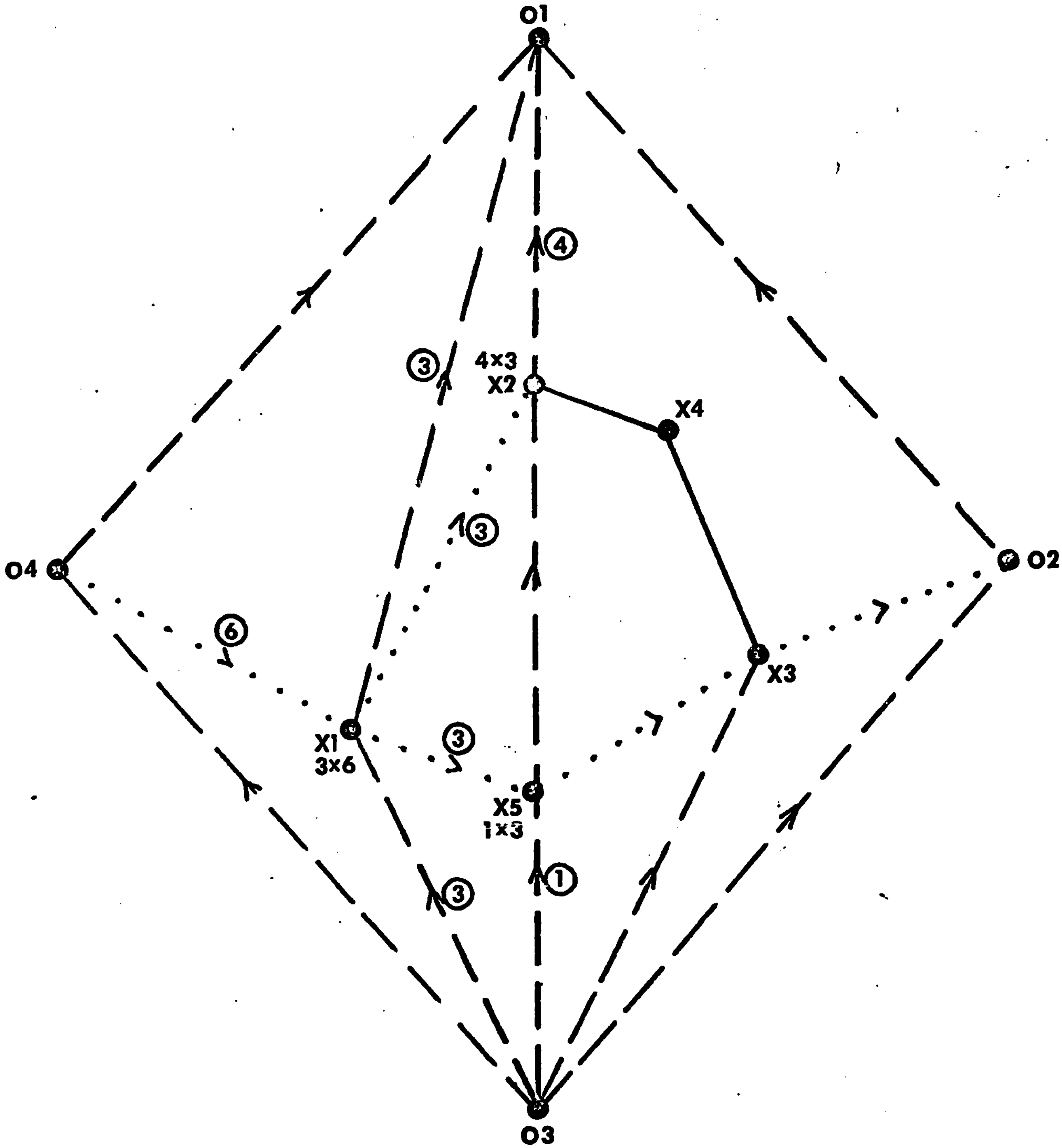


FIGURE 4-5

AT MOVE 148 IN EXAMPLE 1

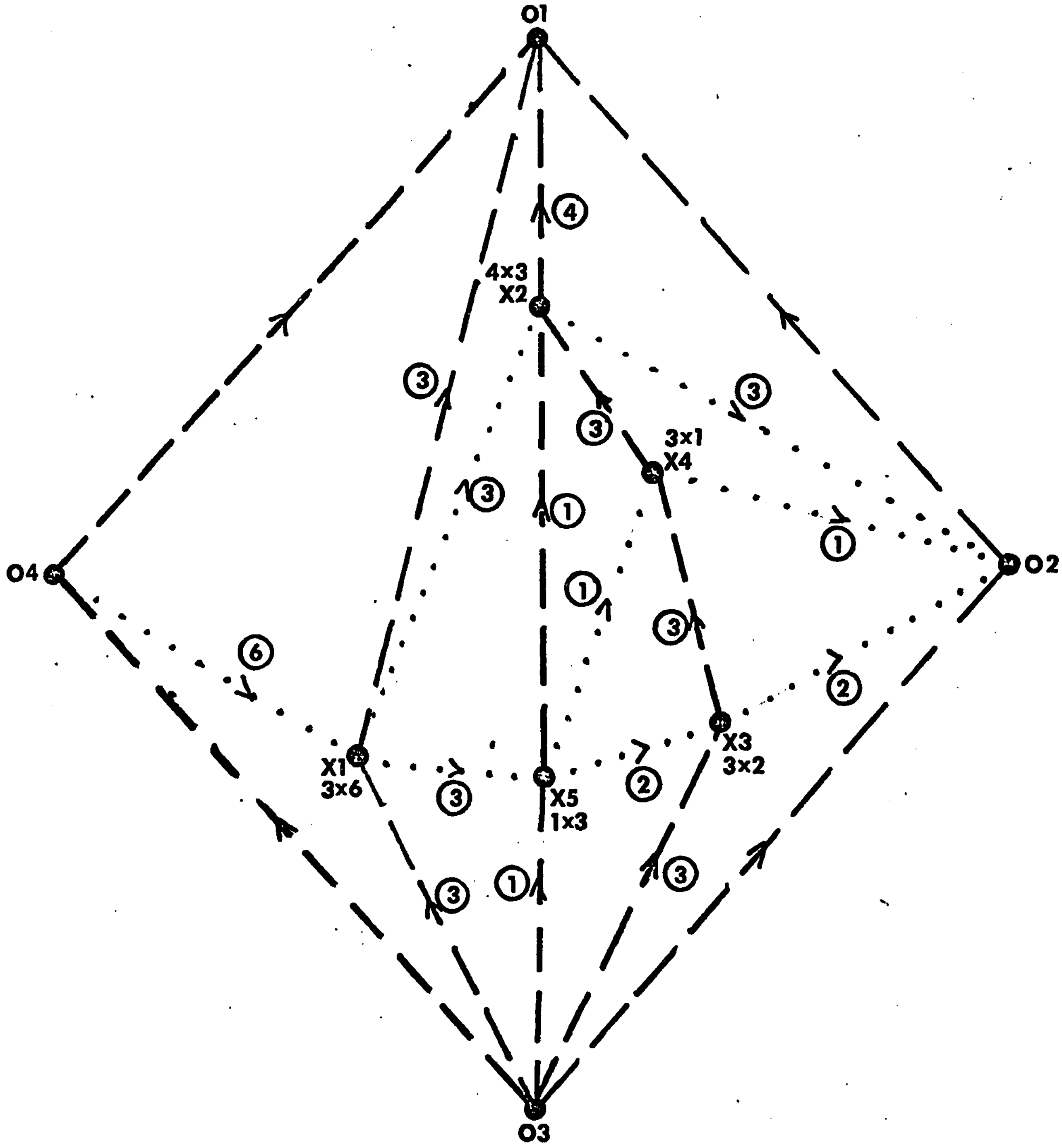
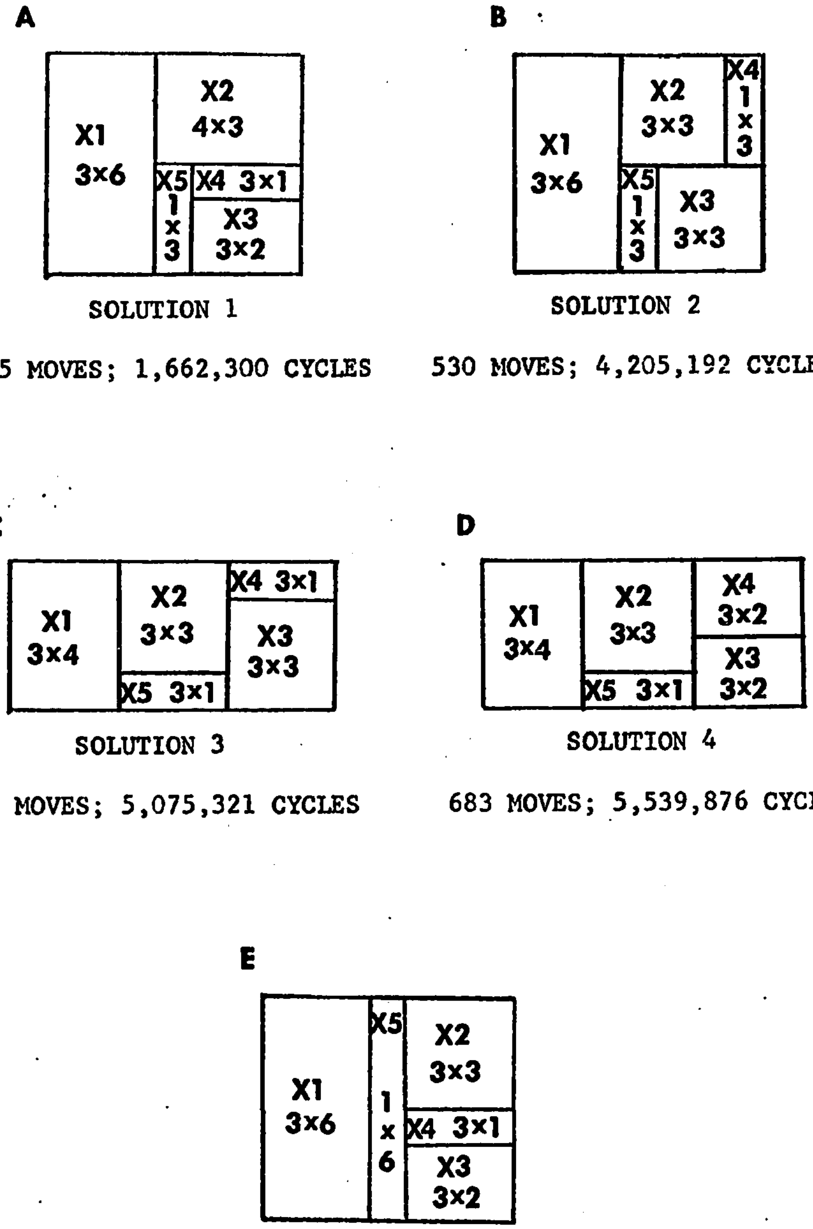


FIGURE 4-6

AT MOVE 185 IN EXAMPLE 1 - FIRST SOLUTION



SOLUTION 1

185 MOVES; 1,662,300 CYCLES

SOLUTION 2

530 MOVES; 4,205,192 CYCLES

SOLUTION 3

635 MOVES; 5,075,321 CYCLES

SOLUTION 4

683 MOVES; 5,539,876 CYCLES

SOLUTION 5

781 MOVES; 6,362,453 CYCLES

848 MOVES; 6,838,109 CYCLES TO COMPLETION

FIGURE 4-7

THE 5 SOLUTIONS FOR EXAMPLE 1

```
SIZES
X1
8          X          4
X2
3          X          3
X3
4          X          2
X4
5          X          4
X5
4          X          4
X6
1          X          3
ADJACENCIES
X1         N          03
X5         W          02
X4         S          01
X4         X          X5
X2         X          X5
X5         X          X6
X2         X          X6
X4         X          X6
X3         X          X6
X1         X          X6
AREA LIMITS
0          TO         99999999
```

FIGURE 4-8

REQUIREMENTS FOR EXAMPLE 2

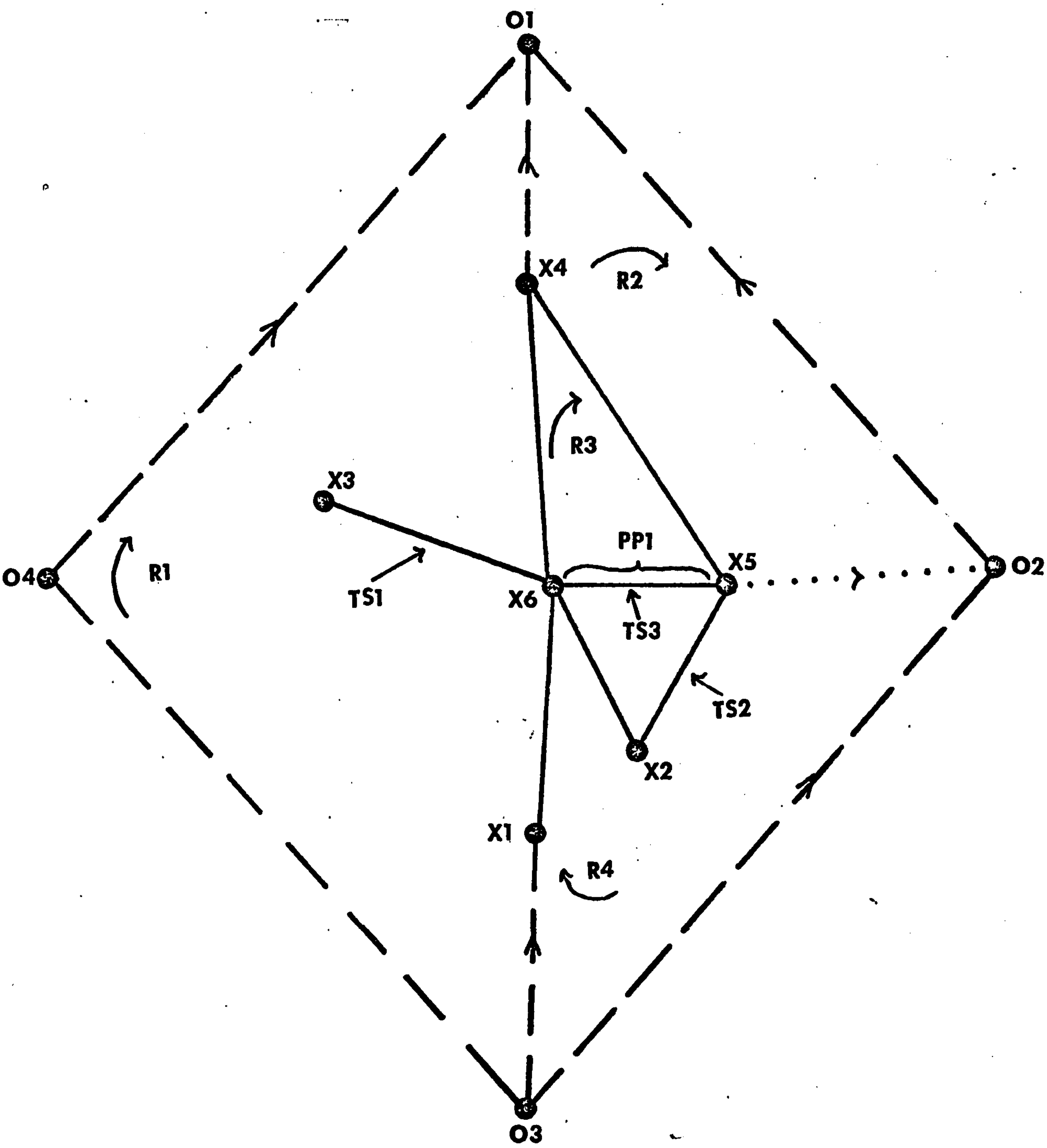


FIGURE 4-9

THE ADJACENCY REQUIREMENT DUAL GRAPH FOR EXAMPLE 2

realization of the graph in Figure 4-9, so that it could be viewed. The PGG non-terminal structures are labelled on the graph for identification.

The first non-terminal structure to note is the pivot pair structure PP1 composed of braced tree structures TS2 and TS3. It has 2! or 2 permutations. The next non-terminal structure to note is the anchored tree structure TS1. According to the realization generation processes of R150, this TS must be braced before realizations of the dual graph can be generated. R150 provides bracings for TS1 by generating different nodes of the graph to which X3 can be connected. The nodes that are generated are 01, 02, 03, 04, X1, X2, X4, and X5, in that order. X3 and X6 have to be omitted because the dual graph is simple. Taking into account the permutation of PP1, this means that R150 will generate 28 realizations for this dual graph. It took the program 100 minutes to generate and test the first 12 of these, and it was terminated at this point since two solutions had already been generated.

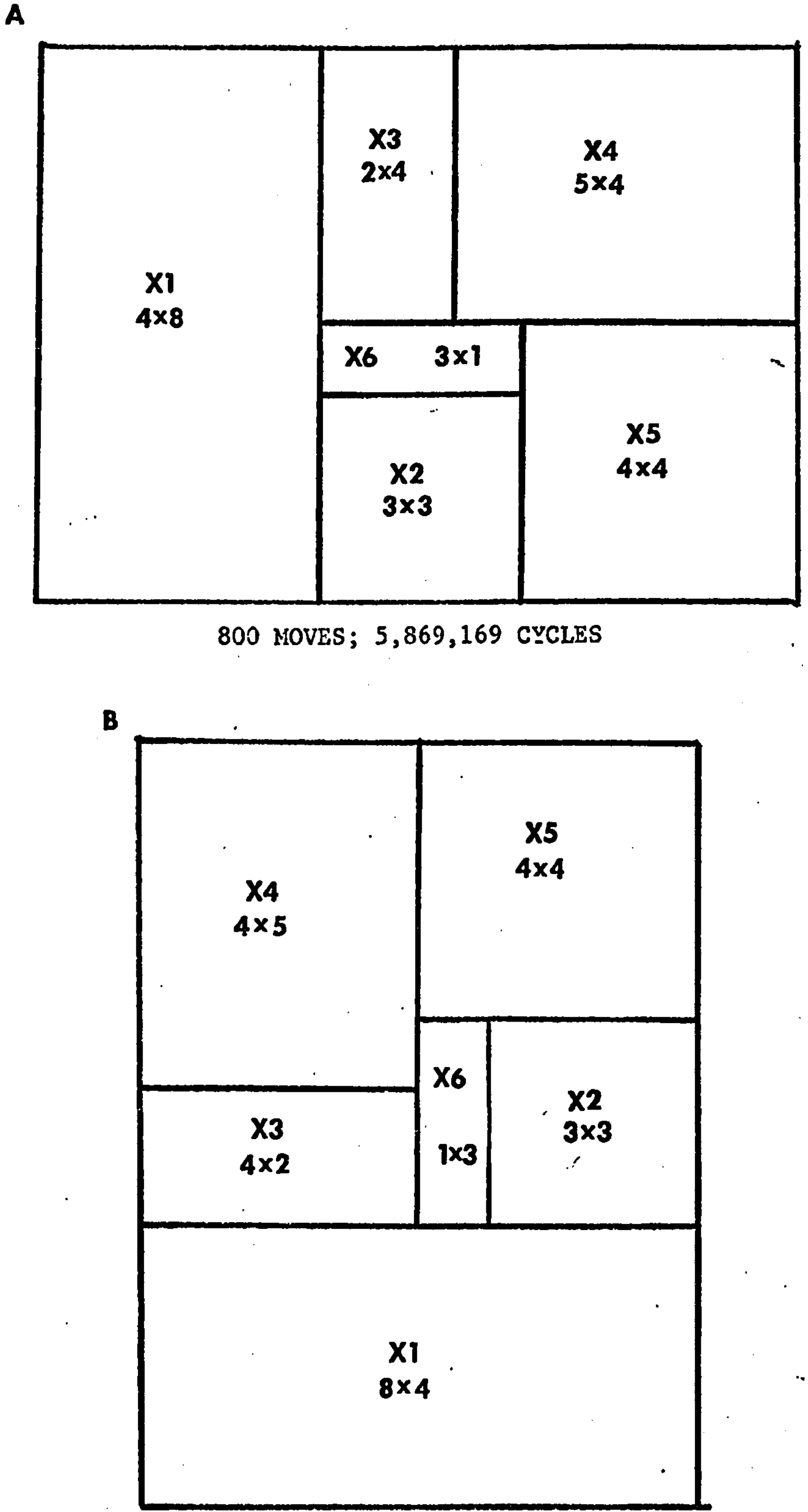
Of the 12 realizations that were generated, exactly half were rejected immediately by routine A14. This was because they attempted to place tree structure TS2 inside of region R3, which results in a physically unrealizable configuration--node X2 would have only three possible adjacencies, while it needs at least four to correspond to a rectangular room. In the remaining six realizations, the relative position of TS2 and TS3 is as shown in Figure 4-9.

The first realization was generated at the end of 63,599 interpretation cycles. Part of the reason that this example takes less time than Example 1 for Phase I is that the program tests a larger number of regions for well-formedness in Example 1. The first solution to this problem,produced after 800 moves and 5,869,167 interpretation cycles, is shown in Figure 4-10-A.

Before the second solution was produced, eight more realizations were generated, four of which the program tried to complete (the other four were rejected by A14). These were the realizations in which X3 was connected to nodes 02, 03 (first with X3 in region R1 and then with X3 in region R4), and 04. With X3 connected to 04, the second solution was obtained after a total of 2,951 moves and 21,984,156 execution cycles, measured from the start of the program. This solution is shown in Figure 4-10-B. Notice that this is just the first solution rotated 90° clockwise. This is a coincidence, but it does point out that if this were the floor plan of a house, the owner could have his choice of orientations and still be guaranteed that his design requirements were satisfied.

4.3. Example 3: Still Longer, Several Realizations

The Example. -- The requirements for this example are given in Figure 4-11. The dual graph for the adjacencies is shown in Figure 4-12. Notice again that some processing had to be done before this graph was ready for Phase II of the program. In particular, node X6 had to be connected to some other node to brace



800 MOVES; 5,869,169 CYCLES

3350 MOVES; 24,684,260 CYCLES

FIGURE 4-10
SOLUTIONS FOR EXAMPLE 2

```
SIZES
X1
16          X          18
X2
16          X          18
X3
10          X          14
X4


16          X          18
X5
16          X          18
X6
4           X          8
4           X          10
X7
6           X          8
6           X          10
X8
8           X          8
8           X          10
ADJACENCIES
X1          X          X7
X1          X          X8
X1          N          03
X2          X          X3
X2          X          X8
X2          N          03
X3          X          X4
X3          X          X8
X3          W          02
X4          X          X8
X4          W          02
X5          X          X8
X5          S          01
X6          X          X8
X7          X          X8
X7          E          04
AREA LIMITS
1200        TO         1500
```

FIGURE 4-11

REQUIREMENTS FOR EXAMPLE 3

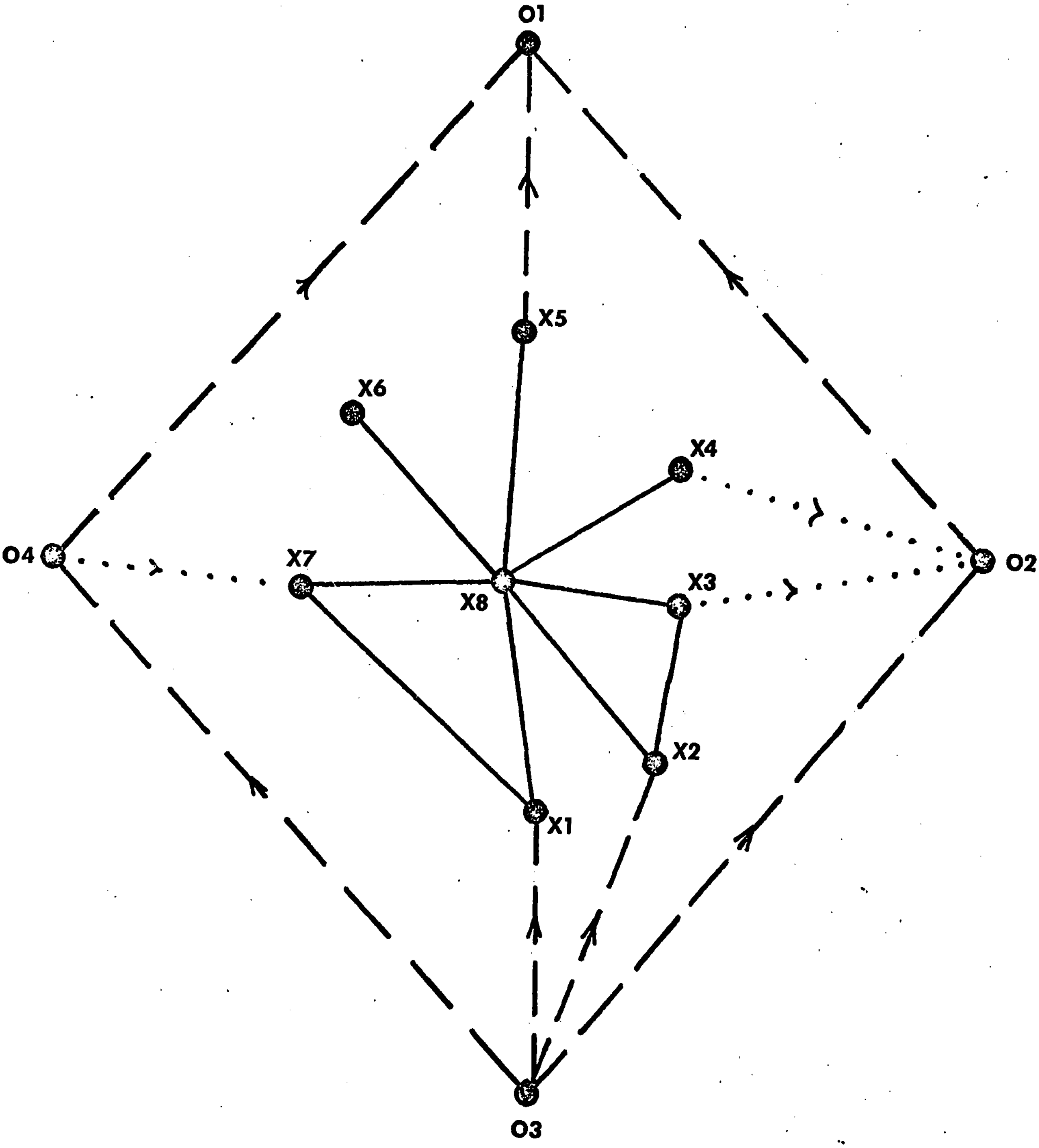


FIGURE 4-12

THE ADJACENCY REQUIREMENT DUAL GRAPH FOR EXAMPLE 3

its parent tree structure. The program was allowed to try nodes 01 (X6 on both the left and the right side of X5), 02, 03, and 04 before it was turned off, having achieved a design solution.

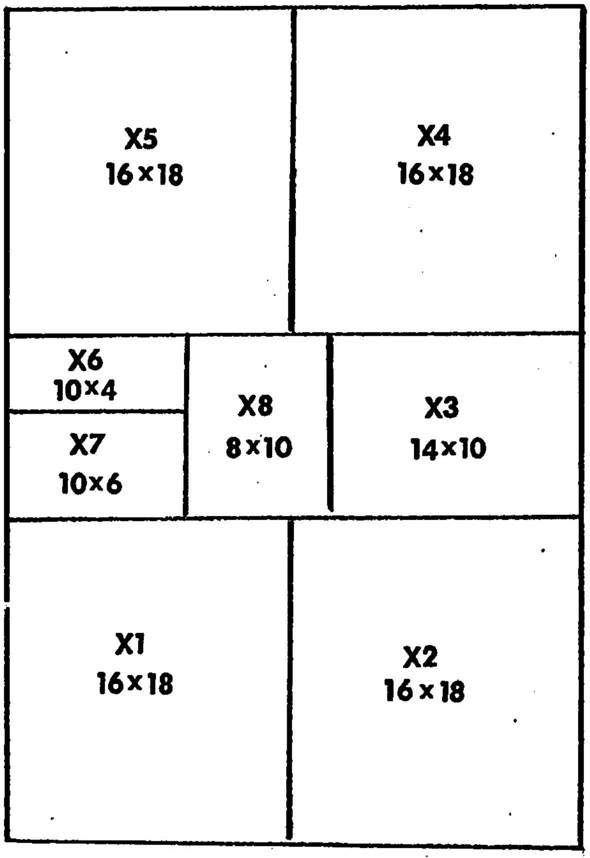
The design solution was achieved after 6,774 moves and 62,636,965 interpretation cycles, or about 3 1/2 hours of computer time.[1] The solution is shown in Figure 4-13.

One observation is quite interesting at this point. Once the realization in which X6 is adjacent to the west wall was selected, it took the program only 260 moves to find the solution. This was less than seven minutes of computer time and is well in line with the computation times of Examples 1 and 2, once their proper realizations were selected. Thus, it appears that if a method existed for selecting realizations in which solutions were most likely to occur, the mean time to the first solution could be reduced greatly.

The Special Nature of the Search -- The above observation points out the special nature of the solution space that must be searched by GRAMPA. If more than one realization exists for the adjacency requirement dual graph, then the search is actually divided into two segments. First a realization must be chosen, and then that realization must be searched to see if it contains a design solution.

If one does not intend to search the solution space exhaustively (as in Examples 2 and 3 when the search was terminated as soon as a solution or two was found), then it is best to work first on

[1] The reader who is disturbed by this is again invited to read Chapter 5 for a discussion of how this program can be converted into a practical one.



6774 MOVES; 62,636,965 CYCLES

FIGURE 4-13

A SOLUTION FOR EXAMPLE 3

those realizations that are most likely to yield solutions. Most of the effort in this thesis was devoted to improving the program's performance in searching a given realization for a solution. The program is quite good at this, although it could be better. However, little effort was devoted to selecting a realization intelligently, and the results reflect this. The only criterion that was used in generating realizations was that, when floating or anchored structures had to be anchored or braced respectively, this would be done by trying first to connect them to outside space nodes. That is, it was assumed that rooms that could have windows were preferable to rooms that could not have windows.

Another aspect of the search should also be mentioned. Assuming that each realization is to be searched exhaustively, there still exists a choice of method. An exhaustive search progresses more quickly if non-productive branches of the search tree terminate early. Hence, it is prudent to conduct a meta-search of the search space to determine first which branches are likely to terminate early, if no solution exists on them, before committing resources to any branch. For instance, suppose that while attempting to complete a given realization, the program decided to start its search by attempting to complete a given region, call it region R1. Suppose further that time and time again each completion of that region would lead to an extended depth first search of other region completions that always resulted in failure. After a few such failures were noted, it might be wise to attempt the completion of the realization by starting with the completion of some other region, because that search strategy, although still exhaustive, might be faster.

The notions of the meta-search of the solution search space and the intelligent selection of realizations were deemed to be out of the scope of the current work. However, these topics are commented upon in Chapter 5.

The Four-Sided Regions. -- It was mentioned in Chapter 3 that, through an omission in programming, the only time the possibility of a four-sided terminal region was considered was when the region to be completed was itself four-sided. A brief discussion is provided here of how that omission affects this example.

Observe that no region that contains one of the outside space nodes in its boundary can be a well-formed four-sided terminal region. This is because the conventions for the outside spaces were selected so that only "T" type corners can exist along the outside walls of the building, and these have to come from three-sided terminal regions (see Figure 2-1).

In Example 3 all non-terminal regions of all realizations considered were configured so that any four-sided region that could be produced would include an outside space node (see Figure 4-12). Thus there were no possible well-formed four-sided terminal regions for this example anyway.

The same thing is true of Example 2, and in Example 1 the only possible four-sided terminal region is one that the search algorithm does not ignore. Thus the programming omission did no harm in these three examples. It could have done harm in Examples 4 and 5 if they had been carried to a solution, but they were not.

4.4. Example 4: Too Long, Many Realizations

The requirements for Example 4 are given in Figure 4-14. The adjacency requirement dual graph is shown in Figure 4-15. Again, names are given to the non-terminal structures for the purposes of illustration. When one observes the large number of ways that the anchored tree structures TS3, TS4, and TS5 can be braced, it should be clear that there is a huge number of possible realizations for this graph. It is difficult to predict how many realizations will be generated, but the number is certainly in the thousands.

It therefore is out of the question to search the solution space of this problem exhaustively. However, the program was run for an hour and fifteen minutes to illustrate its capability for treating a problem with many realizations.

The first realization accepted for completion by the program is shown in Figure 4-16. It occurred after 228,966 interpretation cycles. This was actually the third realization generated, but the other two were rejected by routine A11. The reason node X4 is connected to node 01 in this realization is that R150 attempted to brace TS3 by connecting X4 first rather than X10. This meant that X10 still had to be connected to some node to brace its tree structure, so it was connected to 01. The same sort of thing holds for nodes X8 and X9. In retrospect it appears that a

```
SIZES
X1
15          X          4
X2
7           X          7
X3
9           X          6
X4
4           X          6
X5
5           X          4
X6
5           X          6
X7
7           X          10
X8
10          X          10
X9
2           X          6
X10
3           X          6
ADJACENCIES
X3          N          03
X3          X          X1
X1          N          03
X7          N          03
X8          S          01
X1          X          X2
X5          X          X6
X4          X          X3
X4          X          X10
X9          X          X8
X7          X          X2
X5          E          04
X5          S          01
AREA LIMITS
0           TO         99999999
```

FIGURE 4-14

REQUIREMENTS FOR EXAMPLE 4

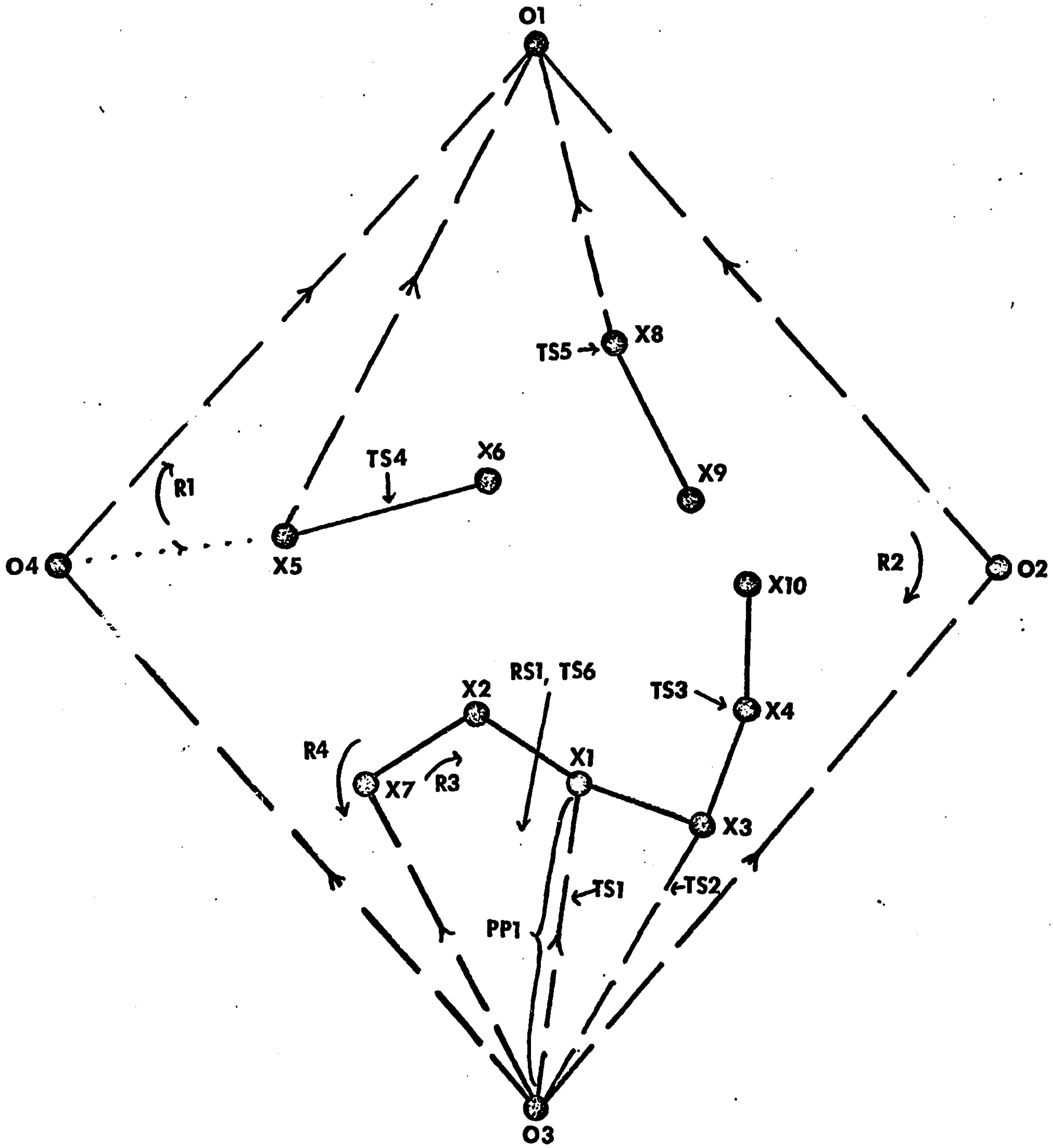


FIGURE 4-15

THE ADJACENCY REQUIREMENT DUAL GRAPH FOR EXAMPLE 4

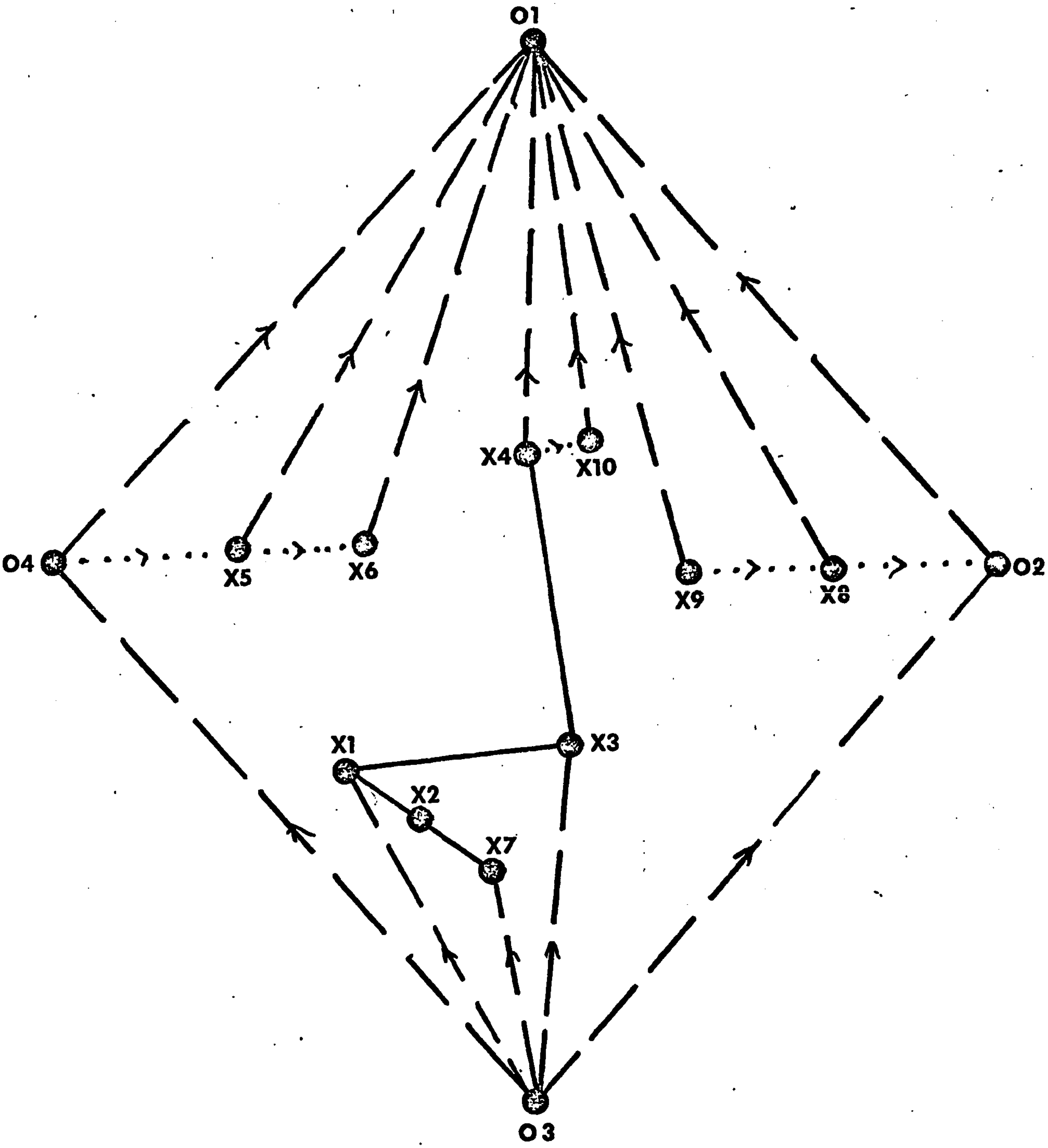


FIGURE 4-16

FIRST REALIZATION FOR EXAMPLE 4

more refined tree bracing algorithm would be desirable--one which would brace all anchored tree structures with a minimum number of edge additions to the graph. This would reduce the total number of realizations generated.

Phase II of the program started on this realization by trying to fill region (03, X3, X4, X10, 01, X9, X8, 02). However, after 1,138 moves and 5,064,272 total cycles, it failed. The large number of moves was required because of the size of the region attempted; but the average number of cycles per move was low, only about 4,400. This is a bit more than half as many as in Example 1, and the reason lies in the large number of rejections by R110, the move plausibility routine. Rejection of a move by R110 does not take nearly as many cycles as rejection by the well-formedness tests.

After the failure of the first realization, a second realization was generated. This is shown in Figure 4-17. It too failed, and subsequently two more realizations were generated, which shifted nodes X1, X2, and X7 to the other side of edge (03, X3) in mirror image of the first two realizations. By the time these four realizations had failed, an hour and fifteen minutes of computer time had been used and the program was terminated. Clearly exhaustive search is not practical for problems of this size.

4.5. Example 5: Rectangular Jigsaw Puzzle

This example was an attempt at a slightly different formulation of the floor plan problem. It essentially took the form of a jigsaw puzzle with rectangular pieces. Seven rooms were specified

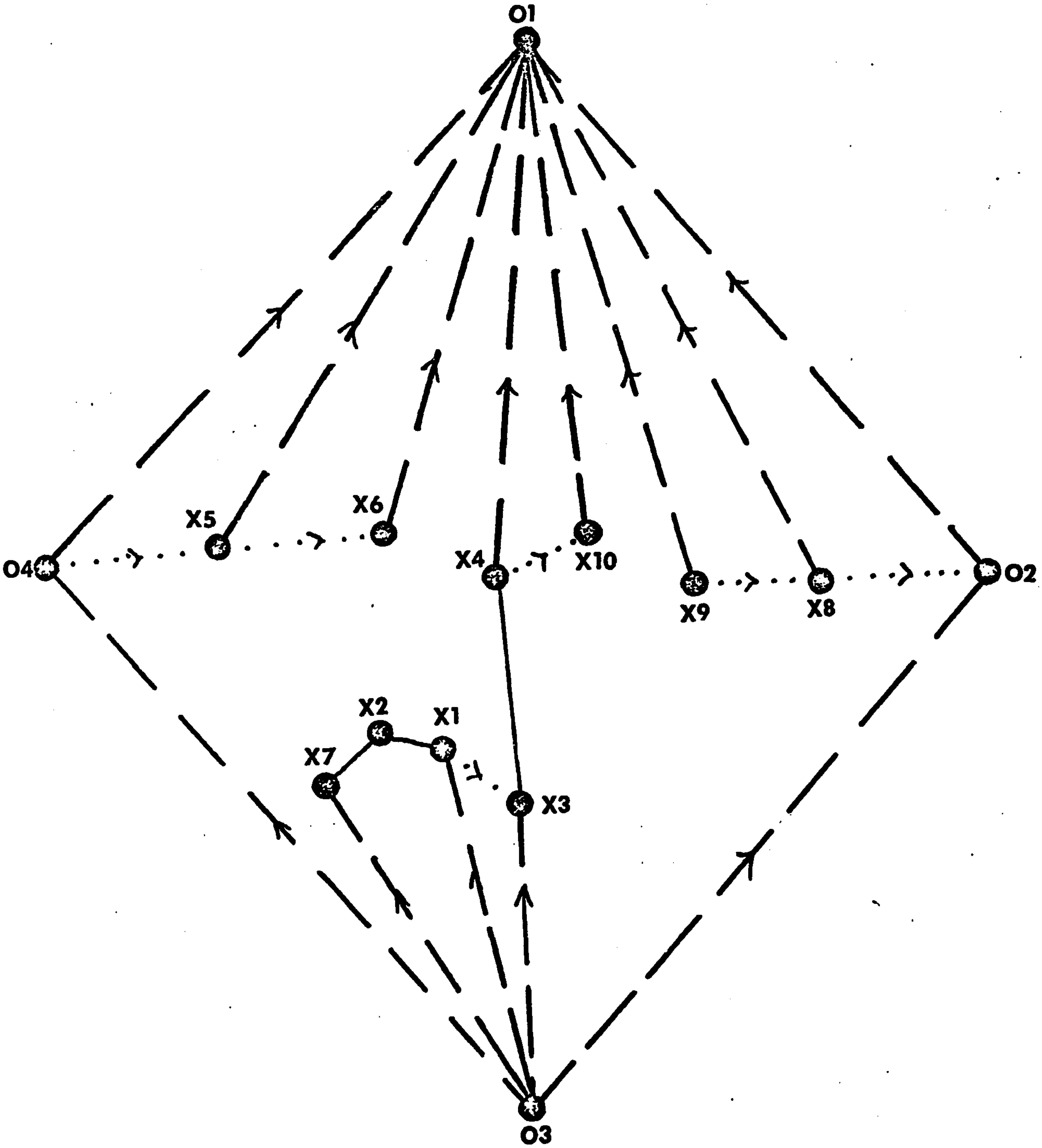


FIGURE 4-17

SECOND REALIZATION FOR EXAMPLE 4

with sizes X1 = 1 x 3, X2 = 2 x 2, X3 = 2 x 2, X4 = 5 x 2, X5 = 3 x 3, X6 = 3 x 2, and X7 = 1 x 6. The task was to pack these rooms into a 6 x 7 building, just like a jigsaw puzzle. No adjacencies were specified for any of the rooms.

The program was not originally designed to treat buildings of definite outside dimensions. This was easily overcome by writing a single routine, R125, to check the outside dimensions. R125 was then applied in routines S1 and R150 to keep track of the possible outside dimensions of the floor plan while realizations were being generated. It was also inserted in the global test routine R158. In the former case R125 had the power to reject an entire realization if it was not well-formed, and in the latter case it could reject only a realization completion move. Thus R125, like R126 (see Section 3.2.3), had the ability to reject realizations while they were in the process of being generated. This type of capability for early search tree pruning is highly desirable.

Since no adjacency requirements were specified for this problem,the creation of the realizations which were to be input to Phase II of the program relied solely on routines S1 and R150, which anchored and then braced the various floating nodes corresponding to the rooms. The first realization that was generated is shown in Figure 4-18. Note that all possible wall lengths of 6 exist at 01 and 03, and possible wall lengths of 7 exist at 02 and 04. These were the required outside wall dimensions.

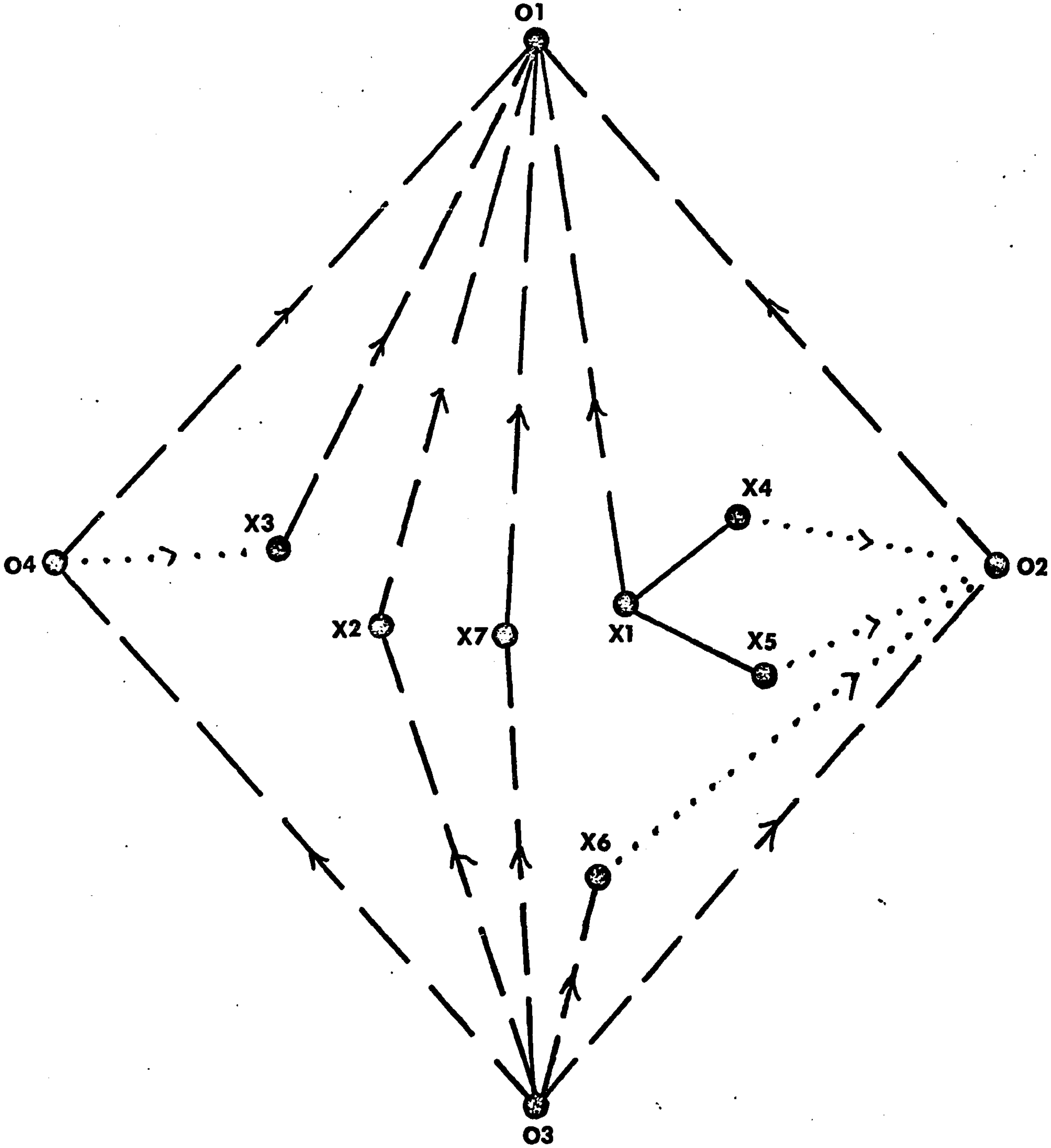


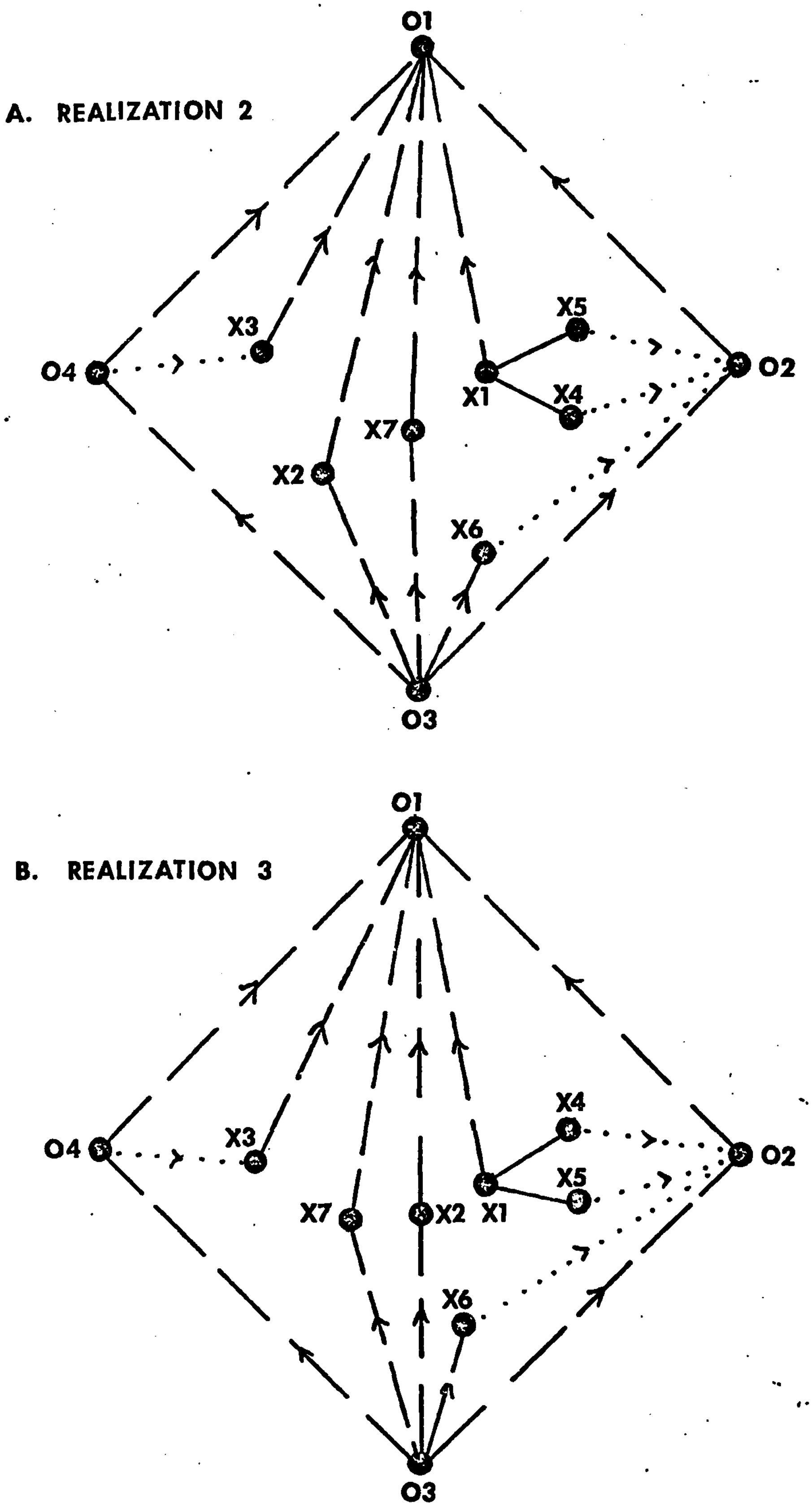
FIGURE 4-18

FIRST REALIZATION FOR EXAMPLE 5

The program started the completion of this realization with region (04, X3, 01, X2, 03) but failed after only 36 moves and 376,226 total interpretation cycles. This was followed in turn by the four realizations shown in Figures 4-19 and 4-20, each of which failed after a few dozen moves. Notice that the variation in realizations 1 through 4 came from the permutations of two pivot pair sets. In realization 5 a new bracing was attempted for node X2, namely to node X1 instead of node 03. Actually a bracing to node 04 was attempted by R150 in the interim, but it was rejected by R126 because it would have meant that both X2 and X3 were trying to occupy the northwest corner of the building at the same time.

For as long as this example was allowed to run, it kept generating new realizations and rejecting them after a few dozen moves. No solution was ever reached, but the example did provide an extensive illustration of the program's realization generation capabilities and its ability to adapt to a different problem formulation.

Four more of the realizations, selected at intervals, are shown in Figures 4-21 and 4-22. One thing all of these realizations have in common is that node X7 is connected to both outside space nodes 01 and 03. Room X7 has allowable dimensions of 1 x 6 (or 6 x 1), whereas the building is supposed to have a north-south dimension of 7. Thus no realization with X7 located as it is in these initial realizations is feasible. The program



A. REALIZATION 2

B. REALIZATION 3

FIGURE 4-19

REALIZATIONS 2 AND 3 FOR EXAMPLE 5

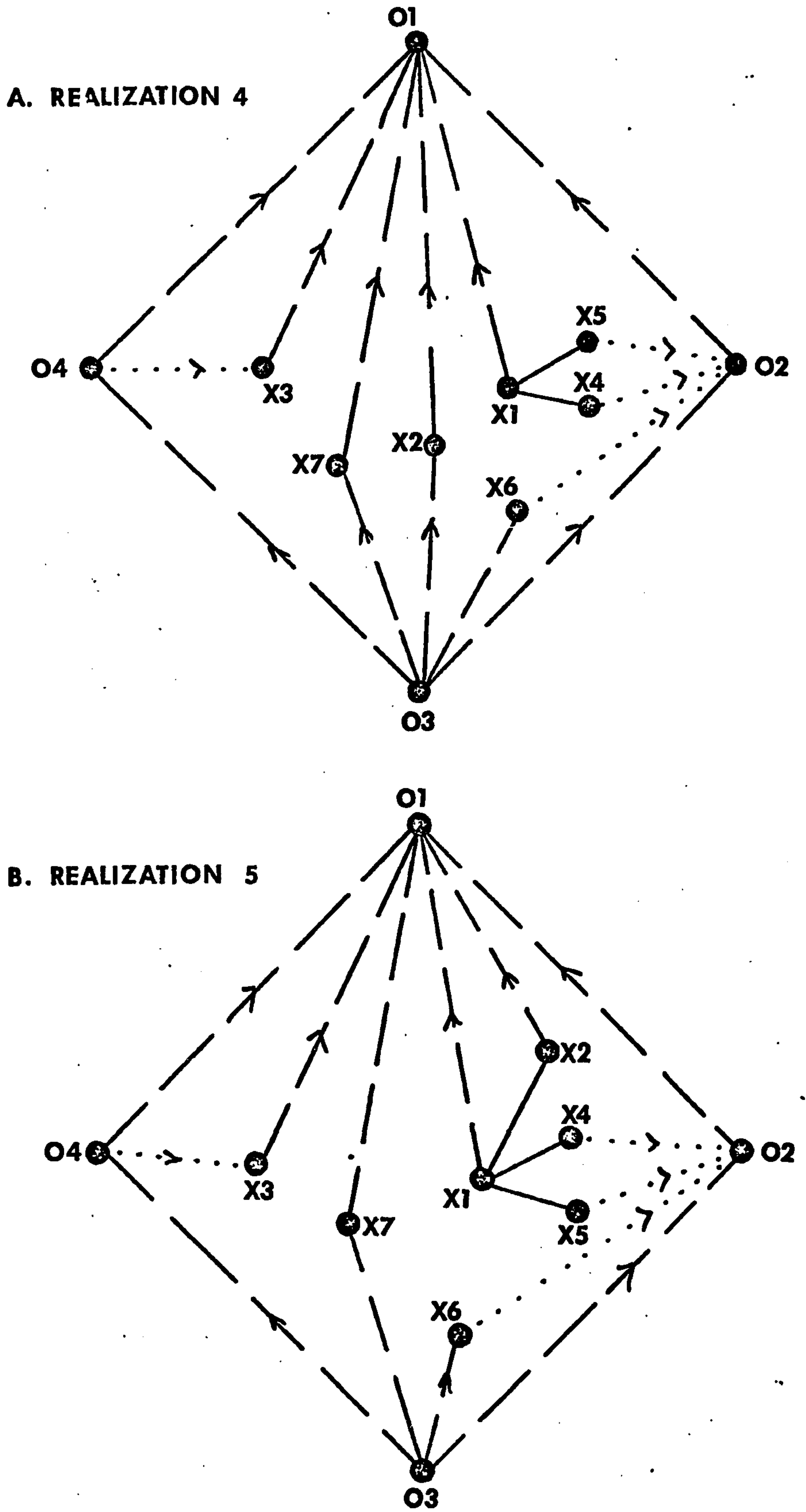
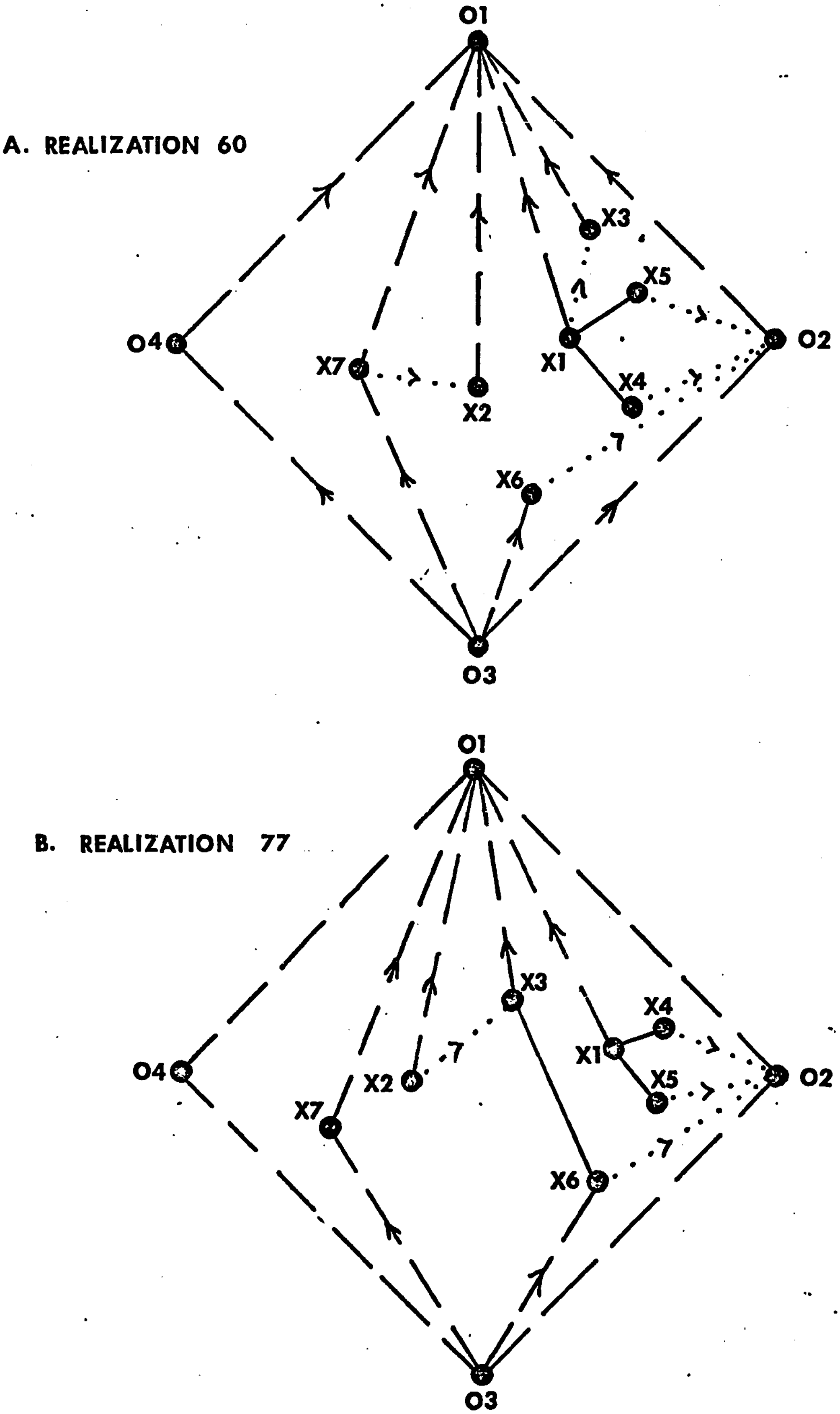


FIGURE 4-20

REALIZATIONS 4 AND 5 FOR EXAMPLE 5



A. REALIZATION 60

B. REALIZATION 77

FIGURE 4-22

REALIZATIONS 60 AND 77 FOR EXAMPLE 5

has no way of knowing this however until it actually tries to complete each realization. This is the reason that each realization was rejected after only a few dozen moves.

One way of remedying this problem is to apply a routine during the generation of realizations that checks to insure that rooms that span the whole building but that are not along an outside wall have the proper dimension. This may seem like an arbitrarily specific type of test, but in the case of this example, for instance, it would remove hundreds of realizations from the search tree. This is an example of a test routine that did not seem necessary until after several problems were run on the computer. Another example of a test routine that was discovered this way is R126 (mentioned earlier in this section), which was actually implemented in the program.

4.6. Conclusions

The purpose of running these examples was to illustrate the current capabilities of GRAMPA. It has been seen that the program is capable of searching a solution space exhaustively, but the size of the problems for which it can do this in a reasonable amount of computer time is rather limited. A main reason for this is that the combinatorial nature of both the generation of realizations and the satisfaction of dimension requirements tends to cause the search space to expand rapidly with seemingly minor changes in the set of design requirements. For instance, the omission of one adjacency requirement may multiply the number of possible realizations for a problem by as much as a factor of ten. Doubling the number of possible room sizes for a given problem much more than doubles the size of the search space.

On the other hand, it has been found that the program is organized so that it is quite easy to make certain extensions to the class of problems it is capable of handling.

These observations lead to two conclusions. First, if the program is ever to be able to produce solutions to problems much larger than these examples in a reasonable amount of time, then the exhaustive search must be abandoned in favor of a heuristic one. This is not inconsistent with the goals of this thesis, since the exhaustive search strategy was merely intended as a framework for the development of the design activities related to the dual graph representation. Further discussion of possible heuristic search techniques will be found in Chapter 5.

The second conclusion, also discussed further in Chapter 5, is that the program can be extended considerably to take into account a much wider class of design requirements. This is also not inconsistent with the goals of this thesis.

5. PROPOSALS FOR FURTHER WORK, AND CONCLUSIONS

5.0. Introduction

The program described in this thesis, GRAMPA, was designed to test and illustrate both the basic principles of computer-implemented design and a particular design methodology, based on the dual graph representation. During the course of writing and testing the program many ideas arose which were deemed to be interesting and valuable, but beyond the scope of the thesis research. Some of these ideas are presented and discussed in this chapter. They illustrate two main points--first, that the current work can fit into a larger framework of research, and second, that there seem to be viable remedies for many of the inadequacies of GRAMPA as a practical design program. Thus the two main themes of this chapter are the possible extensions and alterations to the program, as discussed in the following sections. Many of the topics listed have already been discussed elsewhere in the thesis. They are mentioned again here so that all proposed alterations and extensions will appear in one place.

5.1. Alterations

5.1.0. Introduction

In Chapter 3, several alternate search strategies for Phase II of the program were mentioned. These are described in the first subsection of this section, and all other alterations are discussed in the second subsection.

5.1.1. Alternate Search Strategies

Heuristic Search.-- In Chapter 4 it was shown that exhaustive search is impractical for problems with solution spaces any larger, or even as large as, those of the examples. This indicates the need for some sort of heuristic search technique. In the course of this chapter many ideas related to the use of heuristic search will be discussed. The details of these ideas will not be treated right now, but rather a general framework for the heuristic search will be provided.

The search strategies discussed later in this section all require a completely specified floor plan topology (as specified by a dual graph with unweighted edges, but that is otherwise completed) before they start the introduction of the dimension requirements. This is done under the assumption that the dimension requirements have been rephrased so as not to be so strict as in GRAMPA. That is, area ranges and dimension ranges are specified for each room rather than a list of specific allowable room sizes. This relaxation in the area requirements makes the use of these other search strategies feasible. It also rephrases the problem so that larger, more general design examples can be treated.

However, as mentioned in Chapter 3, the limitation that completed floor plan topologies must be available before the dimension requirements can be introduced throws away some of the power of the dual graph representation. It does not allow the adjacency and dimension requirements to interact as early in the search as they do in the present program.

Thus a search strategy should be considered which would introduce dimension requirements into a partially completed dual graph, as in the case of the current program, but without exhaustive search of the solution space. This will be called the heuristic search strategy in this chapter. The details of this strategy would depend, among other things, on the specific choice of a class of design requirements to be considered. Given this choice, the routines for selecting realizations and the routines for selecting region completion moves would have to be heuristic rather than exhaustive. These routines would have to be designed so as to examine selectively those areas of the search space which were suspected to be rich in solutions. Furthermore, the well-formedness tests would have to be altered to reject all partial solutions with a high likelihood of failure, rather than only those known definitely to be infeasible. In short, all of the powers of already developed heuristic search techniques could be applied to the program. In addition, because of the peculiar nature of the search space (see Section 4.3), some new aspects of heuristic search could be explored as well. Ideas related to many of these items are discussed in more detail in other sections of this chapter.

A Mathematical Programming Approach. -- Assume that a dual graph has been generated that is completely specified in every respect, except that no edge weights have been assigned. This dual graph would correspond to a floor plan which had its topology specified, but not its room dimensions. This type of floor plan constitutes the basis for the next three design strategies to be discussed.

Thomas Moran, a graduate student in Computer Science at Carnegie-Mellon, has written a paper concerning the dimensioning of the rooms of such a topologically specified floor plan.[1] He formulates the problem as a mathematical programming problem, A brief description of that formulation is given here.

The dimensioning problem is taken as that of assigning weights to all the edges of the dual graph under consideration, subject to metric and configurational constraints. Moran is interested in producing the one solution, out of perhaps many feasible solutions, which minimizes some cost objective function. The function chosen is

$$\text{MIN}\left[\sum_i c_i^1 u_i v_i + \sum_i c_i^2 u_i + \sum_i c_i^3 v_i\right]$$

where the u_i and v_i are room dimensions and the c_i are cost coefficients.

The various types of constraints that can be imposed on the dimensions of a room are shown in Figure 5-1. A minimum area requirement is shown as the hyperbolic curve ① in the Figure. Maximum and minimum tolerable dimensions for the room are shown as the linear curves ② and ③ . Acceptable intervals of the room's length-width ratio are shown as ④ , and wall perimeter limitations are shown as ⑤ . In addition, derived constraints

[1] See Ref. 21.

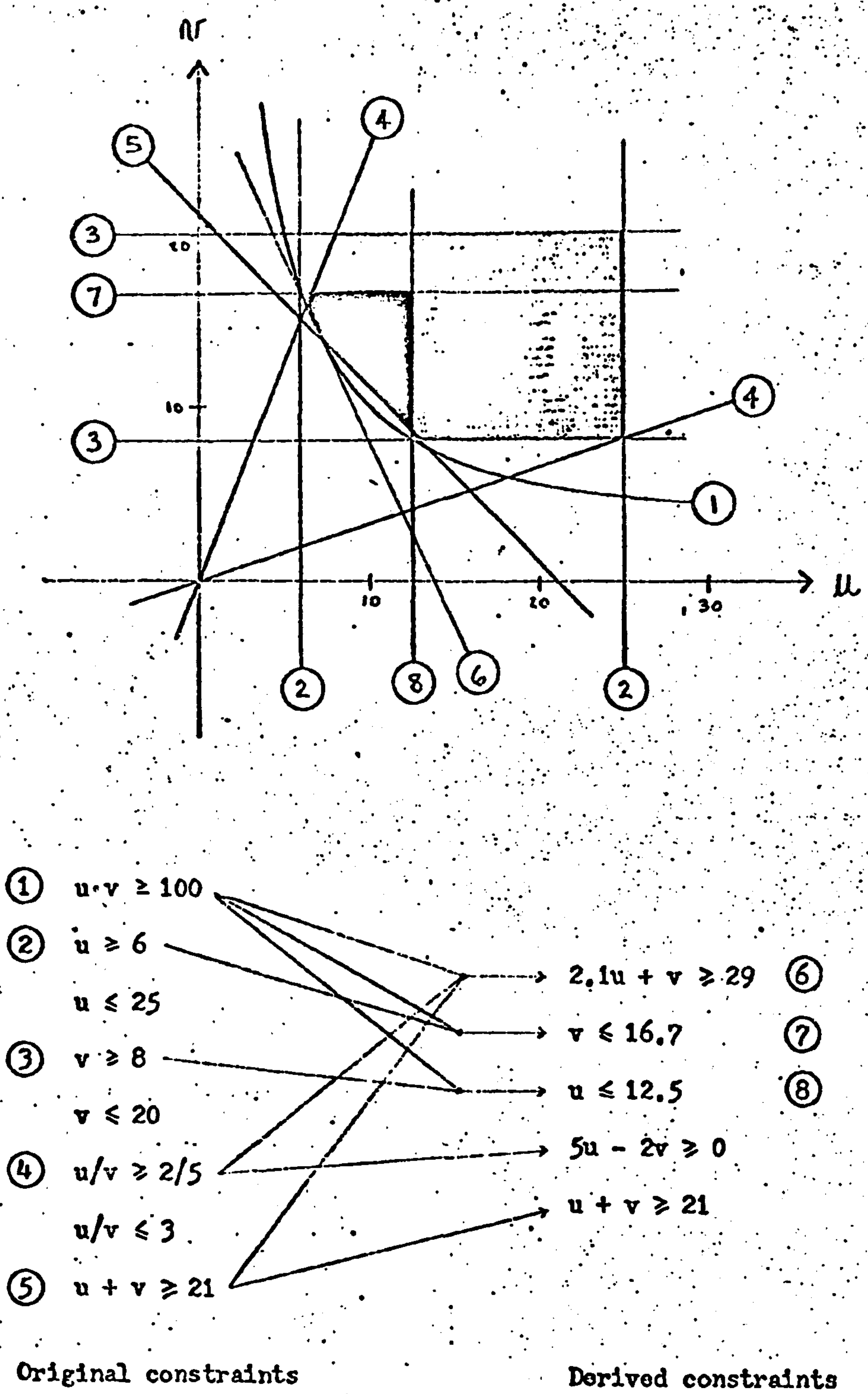


FIGURE 5-1

ILLUSTRATION FROM MORAN'S PAPER

are shown as ⑥ ,⑦ , and ⑧ . All of these combine to produce the shaded area in the Figure that corresponds to the range of allowable values of u and v, the dimensions for the room under consideration.

In setting up the problem for solution, not all dimensions of all rooms are independent variables. Referring to the simple floor plan in Figure 5-2,[2] use can be made of the Kirchhoff flow properties of the dimension weights for the edges of a given color in the dual graph to arrive at a set of independently variable dimensions. These are shown as dimensions X_1 through X_5 in the Figure. A different set might have been chosen as well.

At this point Moran proceeds to simplify the problem for solution by mathematical programming. The interested reader is encouraged to read the referenced paper for details.

A GPS Approach. -- GPS is the General Problem Solver program of Newell and Simon.[3] A class of problems can be formulated for GPS by specifying for the class a set of objects, the concept of the difference between objects, and a set of operators for transforming one object into another. One object is specified as the starting point and another object is specified as the solution to the problem. GPS has built into it a methodology for applying operators selectively to the starting point

[2] This and the preceding figure were reprinted from Moran's paper with permission.

[3] See Ref. 22 .

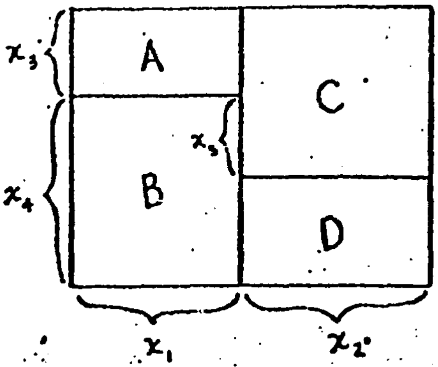


Figure 4

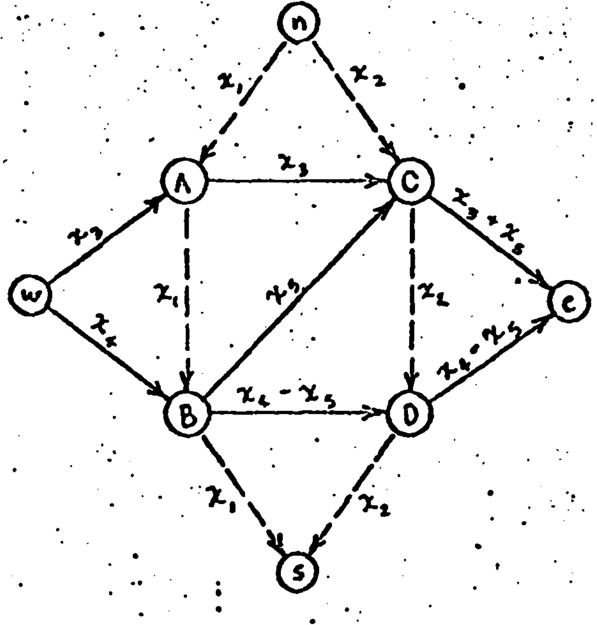


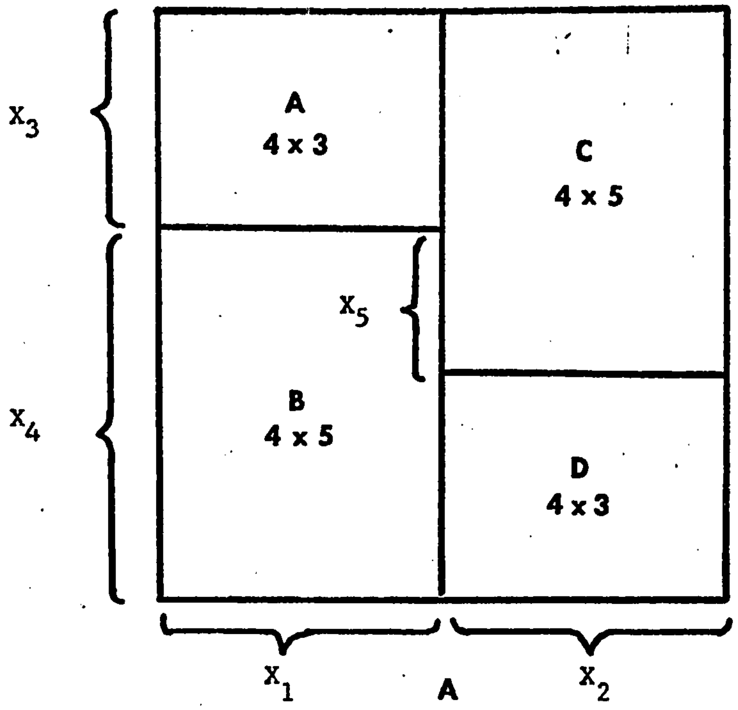
FIGURE 5-2

ILLUSTRATION FROM MORAN'S PAPER

object and intermediate objects in an attempt to transform them into the solution object. It is directed in this attempt by the continual observation of the difference between the current object and the solution object.

A GPS-like formulation can be made for Phase II of the floor plan problem. The starting point object is a topologically specified, but arbitrarily dimensioned floor plan, such as Figure 5-3-A. A solution object will be a floor plan that is fully dimensioned and that satisfies some specified set of dimension requirements. The concept of a difference can be interpreted as a two column matrix, each row of which represents the range of differences between some dimensional attribute of a solution floor plan and the current floor plan. Suppose, for instance, that the desired areas for the rooms of the floor plan in Figure 5-3-A are (1) for room A, 10 to 15 square units, (2) for room B, 10 to 15 square units, (3) for room C, 25 to 30 square units, (4) for room D, 10 to 15 square units, and (5) for the whole building, exactly 64 square units. Then the difference matrix might look like Figure 5-3-B. Notice that a difference in signs in any one row indicates that the present value of the corresponding attribute is satisfactory.

There would be five possible operators for this floor plan, one for each of the independent dimensions. Corresponding to each operator would be a list of the dimensional attributes that it affected, and in which direction. This could be written in vector form, as shown for the five operators for this problem in



$$\left.\begin{array}{l}\text{ROOM A}\\ \text{ROOM B}\\ \text{ROOM C}\\ \text{ROOM D}\\ \text{TOTAL AREA}\end{array}\right\} \Longrightarrow \left|\begin{array}{c}-2 \text{ TO } +3\\ -5 \text{ TO } -10\\ +5 \text{ TO } +10\\ -2 \text{ TO } +3\\ 0 \text{ TO } 0\end{array}\right|$$

B

$$\begin{array}{lccccc}\text{DIMENSION INCREASED} \longrightarrow & X_1 & X_2 & X_3 & X_4 & X_5\\ \text{ROOM A} & + & 0 & + & 0 & 0\\ \text{ROOM B} & + & 0 & 0 & + & 0\\ \text{ROOM C} & 0 & + & + & 0 & +\\ \text{ROOM D} & 0 & + & 0 & + & -\\ \text{TOTAL AREA} & + & + & + & + & 0\end{array}$$

C

FIGURE 5-3

ILLUSTRATIONS FOR THE GPS METHOD

Figure 5-3-C. A further refinement would be to include the rate of change of these attributes with respect to the rate of change of the dimension.

Once the problem is formulated in this way, the full power of the search methods of GPS can be brought to bear on it. These are too lengthy to go into here, so the reference should be consulted for details.

Circuit Simulation Approach.-- The method described here springs from a special representation devised by C. A. B. Smith. While working with William H. Tutte and others on the problem of "squaring the square,"[4] Smith devised a notation for representing rectangles completely filled with squares that is remotely related to the dual graph representation. As adapted for the case of a rectangle completely filled with rectangles, this notation can be described as follows:

Consider a typical floor plan, such as that shown in Figure 5-4-A, and label east-west walls on it in alphabetical order from north to south. A linear graph can be constructed by mapping each east-west wall into a node and each room into a directed edge that is assigned a weight equal to the width of that room (see Figure 5-4-B). Note that Kirchoff's current law applies for the weights and the nodes on the graph. Note also that this is not the dual graph of the floor plan.

Next, the north-south dimensions of the rooms can be introduced by letting each edge of the graph be an electrical resistor

[4] See Ref. 25.

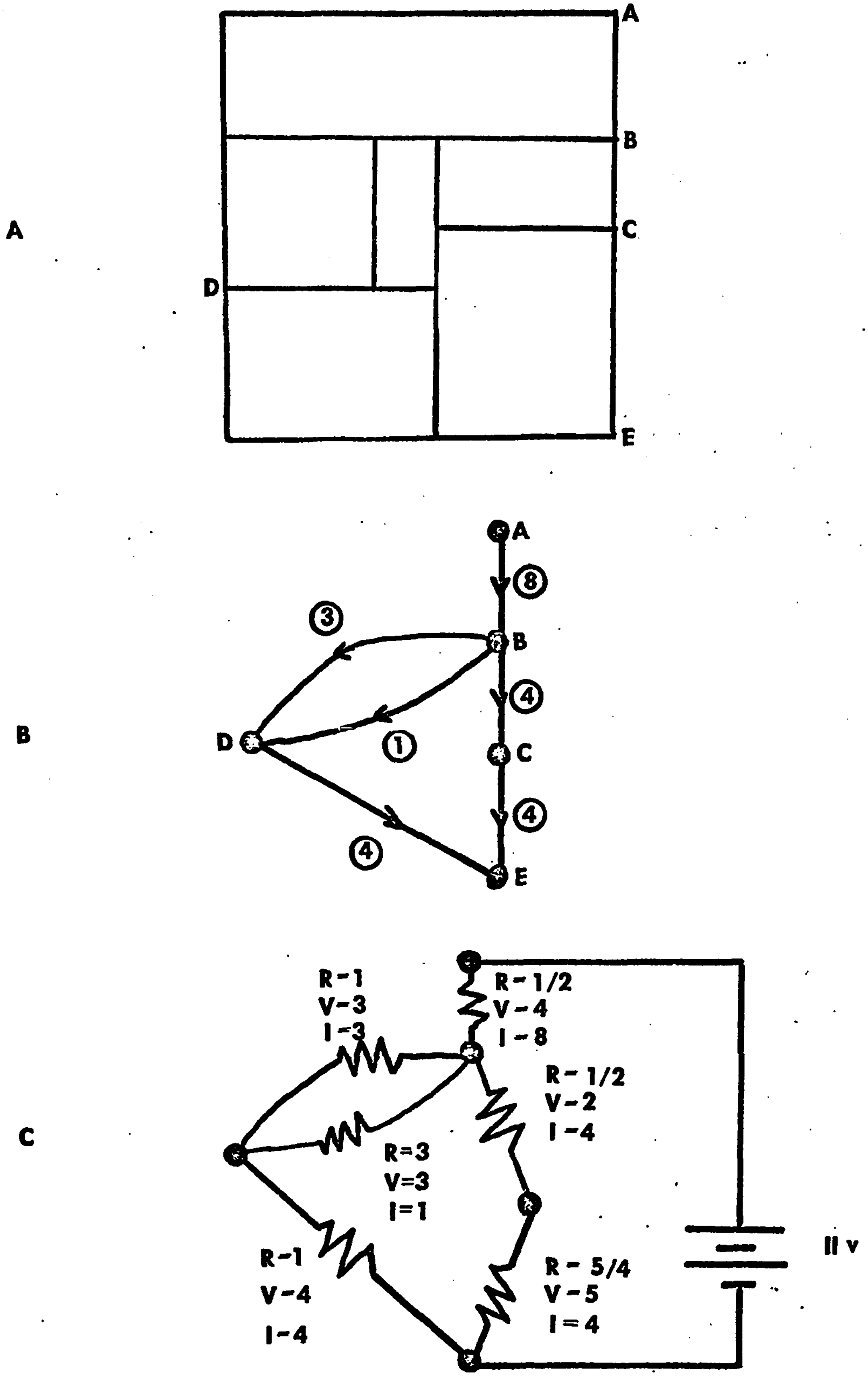


FIGURE 5-4

ILLUSTRATIONS FOR THE SMITH DIAGRAM

which has a resistance equal to the ratio of the length to the width of the room. Then, if one considers a voltage source to be connected across each resistor in volts is equal to the length of its room, the current through each resistor in amps is equal to the width of its room, and the power dissipated by each resistor in watts is equal to the area of its room. Similar analogies hold between the building as a whole and the entire network.

This representation suggests the following approach for Phase II of the floor plan design problem: A network of the type shown could be simulated on a computer. This network would represent a fixed topology for a floor plan. The values of the resistors in the network would be adjustable and would represent the length to width ratios for the various rooms. The overall length of the building could be controlled by using a constant voltage input. As the various resistors were varied, different dimension assignments for the floor plan would be produced.

It appears that it would be feasible to produce a continuous display of the floor plan simulated by this network on the display screen of the computer. Thus one would be able to see directly and immediately the results of changing a given resistance.

Some interesting effects could be produced using such a system, such as producing floor plans in which all rooms are squares (simply set each resistor to one ohm), or floor plans in which all rooms have length and width in the golden ratio. Unfortunately, however, one would not have direct control over any given dimension of the floor plan, since all dimensions would be interrelated by the network.

Note that the number of independent variables in the network is equal to the number of resistors plus one for the outside voltage source. This is equal to the number of rooms plus one--the same result that Moran got.

It should be repeated that one big limitation of all three of the above search strategies is that they require specifying an entire floor plan topology before rooms can be dimensioned. This limits the number of different topologies that can be considered much more than the heuristic search strategy would. The heuristic strategy on the other hand finds it more difficult to deal with a wide class of dimension requirements. Thus it is difficult to predict which strategy is best without specifying the class of problems to be treated.

5.1.2. Other Alterations to the Program

5.1.2.0. Introduction

This section lists briefly areas for possible alterations to the program, other than the search strategy of Phase II. The ideas for these alterations arose either as a result of observed deficiencies in the program or as a result of consideration of capabilities that would be necessary to implement a heuristic search strategy. The section is divided into two subsections--the first on search related alterations and the second on test related alterations. Many of the discussions in these sections are short, for the simple reason that the exact means of implementing the alterations have not been thought out yet.

5.1.2.1. Search Related Alterations

Search of the Space of Search Sequences. -- As was mentioned in Chapter 4, given that there virtually always exists a multiplicity of distinct move sequences leading to a given design solution, it is desirable to be able to choose short sequences. This is the problem of searching the space of search sequences.

In the context of a heuristic search strategy, searching the space of search sequences takes on even more significance. The main purpose of using heuristic search is to allocate the resources of computer time and computer space in as efficient a way as possible to produce a high likelihood of coming up with a design solution. Short search sequences, of course, further this cause. For the problem in question any strategies for selecting short sequences would have to rely on some sort of overall observation of intermediate states of the floor plan design. Anomalies such as exceedingly large or exceedingly small dimensions, large numbers of adjacencies for a given room which was not intended as a corridor, and rooms without windows would tend to be avoided. It is clear that any heuristic search strategy would necessarily have built into it certain judgments as to what constitutes good floor plan design. This is one very good reason that the author, who is not an architect, chose to implement an exhaustive search strategy in this program--one which had to reply only to a given set of explicit design requirements. The change to a heuristic search strategy would not be attempted without familiarity with the principles of good design in the category of problems attempted.

Unbuilding Instead of Saving. -- It was observed in Chapter 4 that the routines that save the various copies of the dual graph at search nodes, and rebuild the graph from the copies whenever a search node is retraced, take a lot of computer time. The copies of the graph also consume much memory space. One alternative is to unbuild a dual graph edge by edge, to return a graph to some previous state.

The use of "unbuilding" routines would also open up the possibility of handling design problems in which not all adjacency requirements had to be satisfied. For example, suppose that half way through the solution of a design problem it was discovered that a certain adjacency was getting in the way of a feasible design solution. The unbuilding routines could be applied to remove the edge of that adjacency from the graph and the search could continue.

Intelligent Choice of Realizations. -- If problems are to be solved which involve a dual graph with a lot of realizations, then some means must be devised for choosing realizations that are likely to lead to design solutions. An initial step in this direction would be to introduce more well-formedness tests into the realization generation routines themselves. That is, one would like to have the capability of rejecting a realization even before it is completely generated, if it is not well-formed. R125 and R126 are examples of such tests.

Other, heuristic tests could be devised as well. For example, one of the reasons that Example 4 could not produce a solution in its allotted time was that too many rooms were concentrated along the north wall of the building. Thus, one heuristic might be to accept only realizations in which the rooms were distributed well along the outside walls. Another heuristic might examine possible building dimensions by checking the allowable dimensions ofthe various rooms either adjacent to the outside walls or which could be adjacent to the outside walls.

Dynamically Varying Dimension Lists. -- Another alteration would allow the program in its current form to handle a larger class of dimension requirements. Recall that the list of allowable sizes for a room is stored as an attribute of its corresponding node. The more allowable room sizes, the larger the search space. On the other hand, a short list of allowable sizes unnecessarily restricts a room.

A compromise is possible. Suppose that initially a range of possible areas was stored as an attribute of a room node, along with a short, non-exhaustive list of possible room sizes which fell into the range. Then whenever the global well-formedness tests were applied (A13), the short list of allowable sizes could be altered to include a few which both passed the tests and fell into the proper area range. These few alternatives would then be used for that room node during the rest of the search, keeping the size of the search space down. The size alternative list could be suitably re-altered any time A13 was applied, as long as the test was passed and the area range was satisfied.

5.1.2.2. Test Related Alterations

Edge Attribute Deduction. -- The set of test and deduction routines for the program in its present form is by no means complete. Two possible additional deduction capabilities are presented in this section.

First consider the subportion of a dual graph shown in Figure 5-5-A. It is clear by inspection that in order for room X1 to be rectangular, edge (X1, X2) must be slashed and directed from X1 to X2, and there must also be a dotted edge directed from X1 to X3. The program currently does not have the ability to make these deductions. However they could be built into the node check routine, A3.

Consider now the situation shown in Figure 5-5-B. Here it is clear that edge (X1, X2) must take a weight which is equal to the minimum of the y dimensions assigned to rooms X1 and X2, provided these rooms are already dimensioned. Although the program has related capabilities, it does not have this particular one, and it could easily be built into the local dimension check routine, A5.

In introducing new tests, one must make sure that they do not consume more time than they are capable of saving. In the first of the two tests above this might in fact be a problem. The second seems less likely to be uneconomical.

Room Spanning Test. -- This proposed test was already described in connection with Example 5 in Chapter 4. It would be used to determine whether a single room spanned the whole floor

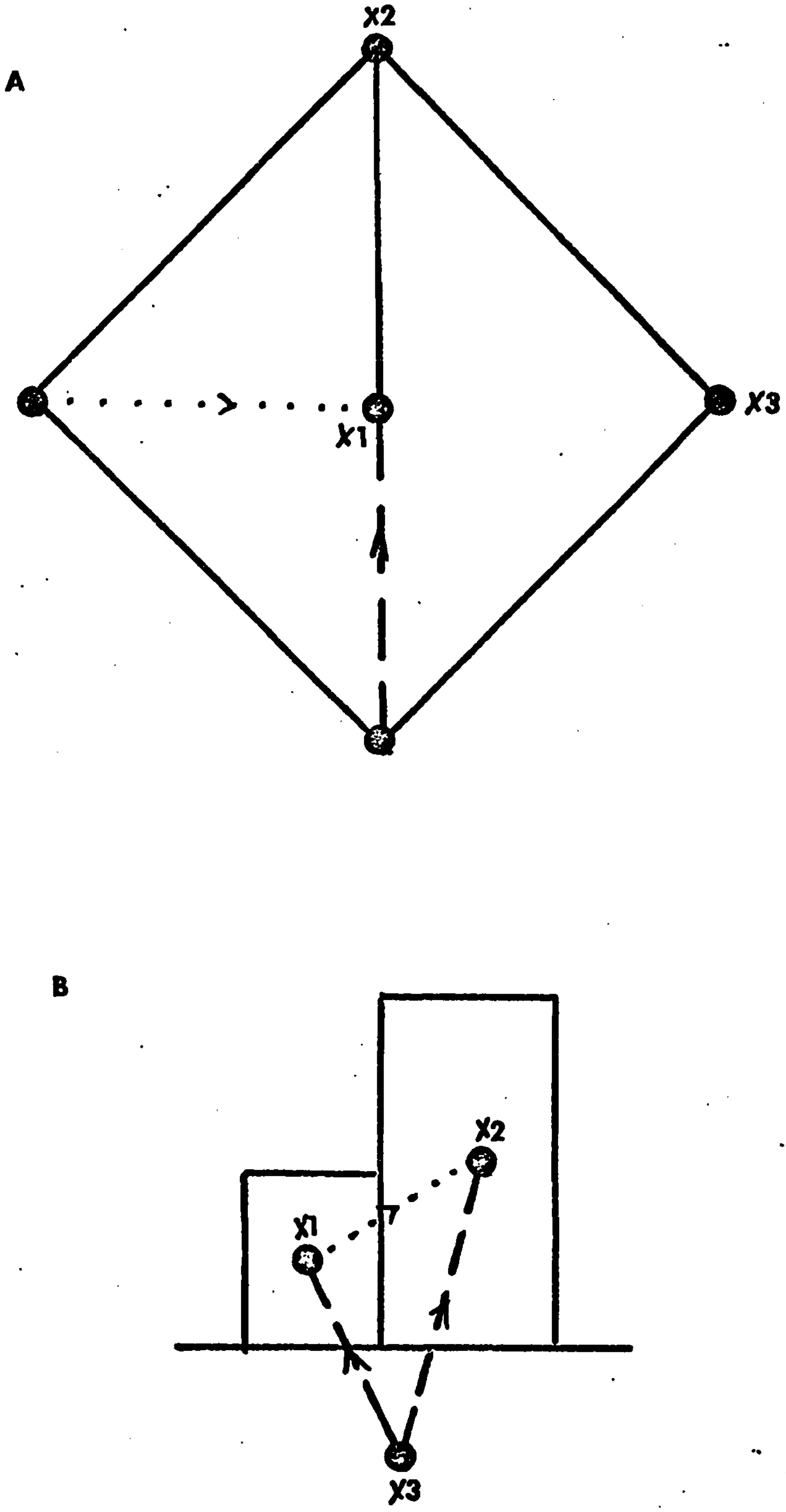


FIGURE 5-5

ILLUSTRATIONS FOR EDGE ATTRIBUTE DEDUCTION plan, and if so, whether its dimensions were compatible with those of the floor plan. An extension to this test would be to give it the ability to check for a whole chain of north-south (or east-west) connected rooms which span the floor plan.

Use of Literal Representation. -- Even though the dual graph notation was devised so that the unnecessary limitations of a literal floor plan representation could be avoided, once a topology is partially specified and some rooms are dimensioned it might be practical to re-introduce the literal representation. Thus, one could sketch portions of the partially completed floor plan without introducing arbitrary dimensions or placements. The author has found that the inspection of these sketches often gives clues to the solution of the problem that the more abstract dual graph cannot provide. Such observations would be especially powerful for the purposes of heuristic search.

This is not to say that the literal representation is better, but rather that the combination of the dual graph representation and the literal floor plan representation is more useful than either taken alone.

5.2. Extensions

Overview. -- Possible alternate search strategies for the program were discussed earlier in this chapter. This section proposes additional types of design requirements that the program might be made to respond to.

Higher Order Requirements. -- One possible way of extending the class of requirements is merely to build requirements out of the already existing adjacency requirements and lists of alternate possible room sizes. Combinations could be made either conjunctively or disjunctively, and negated requirements could be used, too.

For instance, different degrees of accessibility of one space from another might be defined as combinations of adjacency requirements. If one space is adjacent to another, the two spaces could be said to communicate. If two spaces both communicate with a third space, designated as a corridor, then the two spaces could be considered directly accessible to one another. If two spaces both communicate with a third space which is not a corridor, then the two spaces could be considered easily accessible to one another.

The notion of accessibility gives some idea of how many doors one must go through to get from one space to another, but it says very little about how far one must travel to get from one space to another. This could be associated with what might be called proximity requirements. A proximity requirement might put a limit on the dimensions of the rooms or corridors that must be traversed to get from one space to another.

As far as dimensions are concerned, the higher order requirement that a room should have a given area or a given range of areas can always be translated into a list of alternate sizes for the room. However, it has already been seen that large alternative size lists lead to large search spaces. For this and other reasons, classes of requirements that arise in some other way than simply from combinations of the adjacency and dimension list requirements might be desirable. Such requirements are discussed in the following paragraphs.

Area Requirements. -- One possible alternative to explicit size lists is to merely treat the desired area range for a room as an attribute of that room. Then, when it is time to assign a specific pair of dimensions to that room, a pair could be chosen that falls into that range and also happens to allow the room in question to fit the current local metric format of the floor plan. This would make it necessary to have some routine that could make a good heuristic choice of a pair of dimensions. Thus, this type of area requirement would have been impossible to use with the exhaustive search strategy.

Maximum and Minimum Dimensions. -- In much the same way that area requirements could be treated, the maximum and minimum dimensions for a particular room could be specified as well.

Weighted Adjacencies. -- In Krejcirik's RG program, the adjacencies between the various pairs of rooms were given weights, according to their relative degree of desirability. Such a capability might also be added to the program of this thesis. It would mean that the program would also need the ability to unbuild edges, as mentioned earlier. This would be the method that the program could use to remove adjacencies of low weight if it turned out that they interfered with the feasibility of the floor plan.

Proximity. -- The proximity requirement mentioned above could only be tested for on an already existing dual graph. It would be desirable to have a representation for proximity requirements that could be satisfied directly on the dual graph. For instance, some sort of a specialized weighted edge (not an adjacency edge) might designate the fact that two spaces had to be within a certain distance of one another. This edge could perhaps only be allowed to cross at most a certain number of adjacency edges.

This suggests a notion that could be applied to the other proposed requirement types as well. Since this entire thesis has been devoted to the development of a design representation which is purposely matched to the class of design requirements that is used, it would be appropriate that any new class of requirements which was added should also have a well chosen representation. These representations would hopefully fit as extensions of the dual graph representation. The degree of importance of this consideration would depend on whether the new requirement types were being added to make the program a better practical design tool, or for the purpose of exploration of design techniques. These alternatives are discussed briefly in the next two sections.

5.3. Making a Practical Design Tool

This section will summarize the various ways for making GRAMPA a more practical design tool. First, it is obvious that some sort of solution search other than an exhaustive one would have to be used in Phase II of the program. Four different search possibilities were mentioned in Section 5.1.1. Second, an expanded class of design requirements would have to be available. These should probably include weighted adjacencies, area ranges for rooms and buildings, allowable dimension ranges for rooms and buildings, and some sort of proximity requirement. Finally, the program should be able to communicate results to the user in a readily usable form. This would include making it able to print floor plan design solutions, and also making it able to tell the user which design requirements are the bottleneck when no solution can be produced.

5.4. Using the Program as a Research Tool

The approach to making the program a more valuable research tool might differ from the approach used to make it a practical design tool. The author would like to explore the effect that variations in the search strategy would have on the class of solutions that the program could produce. It is quite obvious, for instance, that varying the design requirements for a problem will lead to different design solutions. Now given that a non-exhaustive search is being employed, it is not as obvious just what kinds of variations in the design solutions can be produced through the use of different design strategies.

A computer program, because of its objectivity and its ability to run a large number of examples should provide an ideal tool for testing the effects of varying design strategies. Referring to the design program paradigm described in Chapter 1, the main parameter to be varied in such research would be the decision scheme. However, the effect of using different design representations and different design activities could be explored as well.

5.5 Conclusions

Chapter 4 and the first part of Chapter 5 have already provided an extensive evaluation of the use of the dual graph representation in floor plan design, as implemented in GRAMPA. This section will serve to review what the author feels are the main contributions of this thesis.

First, the dual graph representation provides the basis for a unique approach to floor plan-like design problems. The extensive development of formal properties of the dual graph not only produced interesting results, but these results were quite useful in the implementation of this approach.

One very important feature of the design method developed in this thesis is the ability of a given dual graph to represent abstractly a whole class of potential design solutions. This allows the application of a single well-formedness test to prune a large portion of the search tree.

The abstraction of the dual graph by the planar graph grammar extends this feature even further. The PGG itself is another important contribution of the thesis. It allows the development of a realization generation algorithm, and a planarity checking routine that involves an expenditure of effort that increases only linearly with the size of a graph.

GRAMPA was the proving ground for most of the ideas developed in the thesis research. An interesting aspect of the documentation of GRAMPA is that the formalization developed in the thesis made it possible to provide a reasonably complete proof that the program actually does what it claims to do. This proof was necessary because it was claimed that the program searched the problem solution space exhaustively.

Several logical extensions of the research done in this thesis have already been described in this chapter. These extensions are not necessarily dependent on GRAMPA, but the program could function well as a focal point for them.

To sum up the thesis research in one sentence, one would say that it is a case study of the interrelationship between a representation and the design methods based on that representation.

A.1. PROOFS OF THE THEOREMS

A.1.1. Proof of Theorem 1

Theorem 1:

A necessary and sufficient condition for an isolated unweighted dual graph chain to be physically realizable is that it contain no $\pm180^{\circ}(\pm90^{\circ})\pm180^{\circ}$ sequence.

Proof-- As far as an isolated chain is concerned, a $+180^{\circ}(+90^{\circ})+180^{\circ}$ sequence looks exactly like a $-180^{\circ}(-90^{\circ})-180^{\circ}$ sequence, only in mirror image (or, if you will, traced in the opposite direction); so therefore one only need carry out the proof for the $-180^{\circ}(-90^{\circ})-180^{\circ}$ sequence.

At this point it is convenient to introduce a third basic property of dual graphs to supplement rules W1* and W2*. Any two contiguous nodes in a chain potentially map into a physical realization in which the rectangle corresponding to those two nodes are contiguous to one another along only one wall surface. (Call these two rectangles A and B.) Since, according to the contiguity specifications of the chain, the rectangles immediately surrounding either rectangle A or rectangle B are not necessarily contiguous, it must always be possible to introduce a new rectangle that is contiguous to both of these rectangles. A floor plan graph-dual graph example of this property is shown in Figure A1-1. Thus one has: pp. 20, 21, 23.

> (W3*.) In a physically realizable chain it must always be possible to introduce a new node contiguous to any two mutually contiguous nodes of the chain

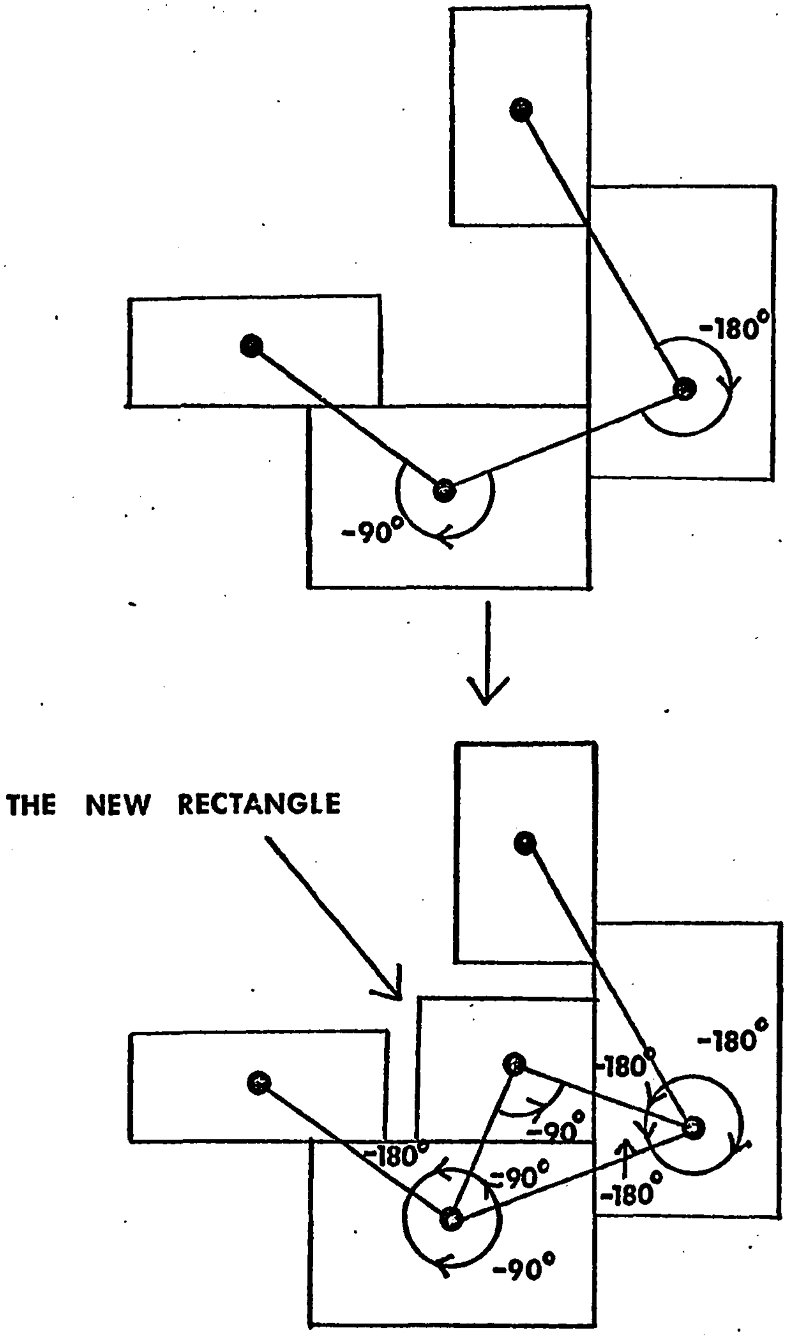


FIGURE A1-1

ILLUSTRATION FOR PROPERTY W3*

without destroying the physical realizability of the resultant dual graph. (This new node should be possible on either side of the chain.)

Now, continuing with the proof of Theorem 1, necessity will be proved first, by showing that if the chain contains a -180° (-90°)-180° sequence, it is not physically realizable. Consider the case of a -180°, -180° sequence, shown in Figure A1-2-A. Using property W1* and the fact that the largest negative turn is -180°, it is seen that any edge connected to either of these two nodes must have two -180° turns surrounding it, as shown in Figure A1-2-B. According to property W3*, if this chain is physically realizable it must be possible to introduce a new node contiguous to both of these nodes on either side of the chain. If the concave side is chosen as in Figure A1-2-C, it is seen that a three sided terminal region with two -180° turns is introduced, which is a contradiction of property W2*; thus the -180°, -180° chain is not physically realizable.

Now consider the -180°, -90°, -180° sequence shown in Figure A1-3-A. Again using property W1* the possible edge connections to the three nodes are as shown in Figure A1-3-B. Constructing the only possible new node connection to two of these nodes in Figure A1-3-C, it is seen that a new -180°, -180° sequence is introduced with the remaining node. However, this has already been shown to be physically unrealizable, so the -180°, -90°, -180° sequence is physically unrealizable. It is

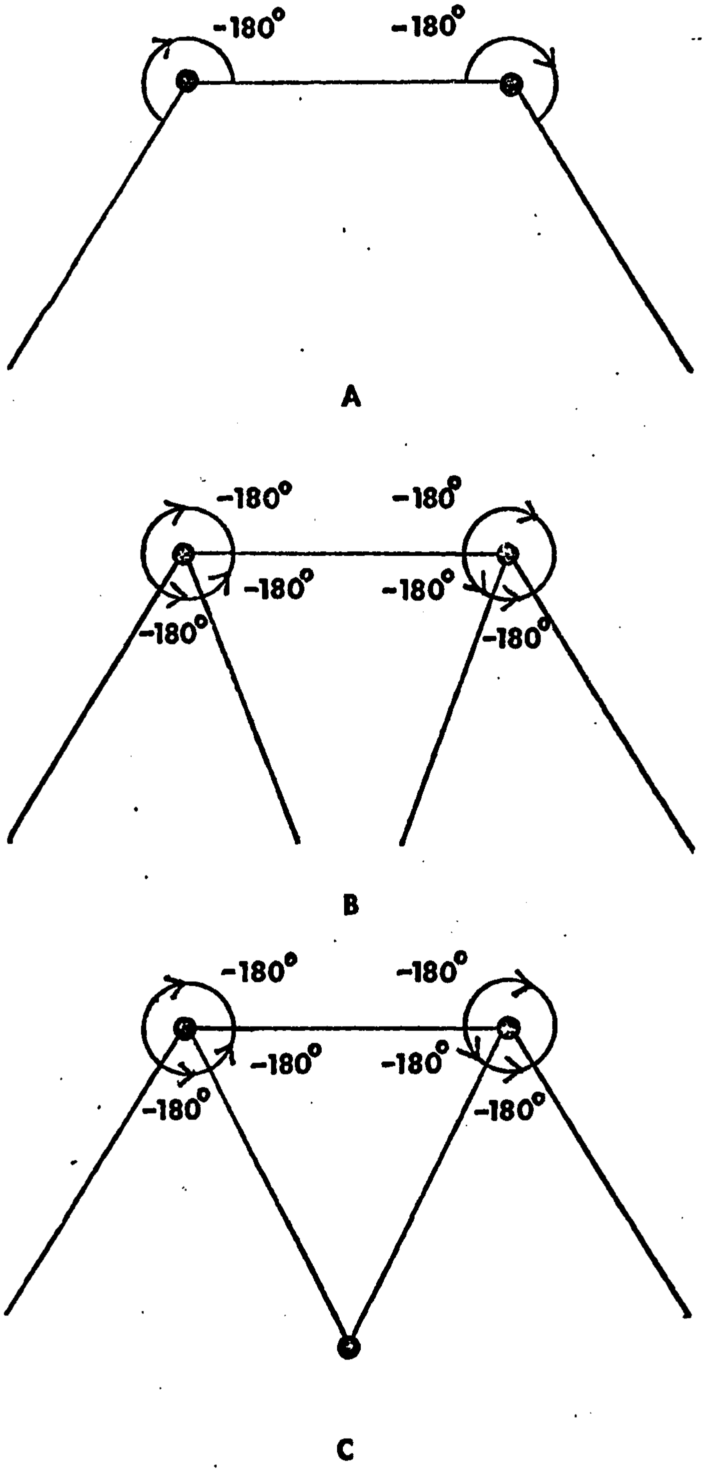


FIGURE A1-2

ILLUSTRATIONS FOR THE PROOF OF THEOREM 7

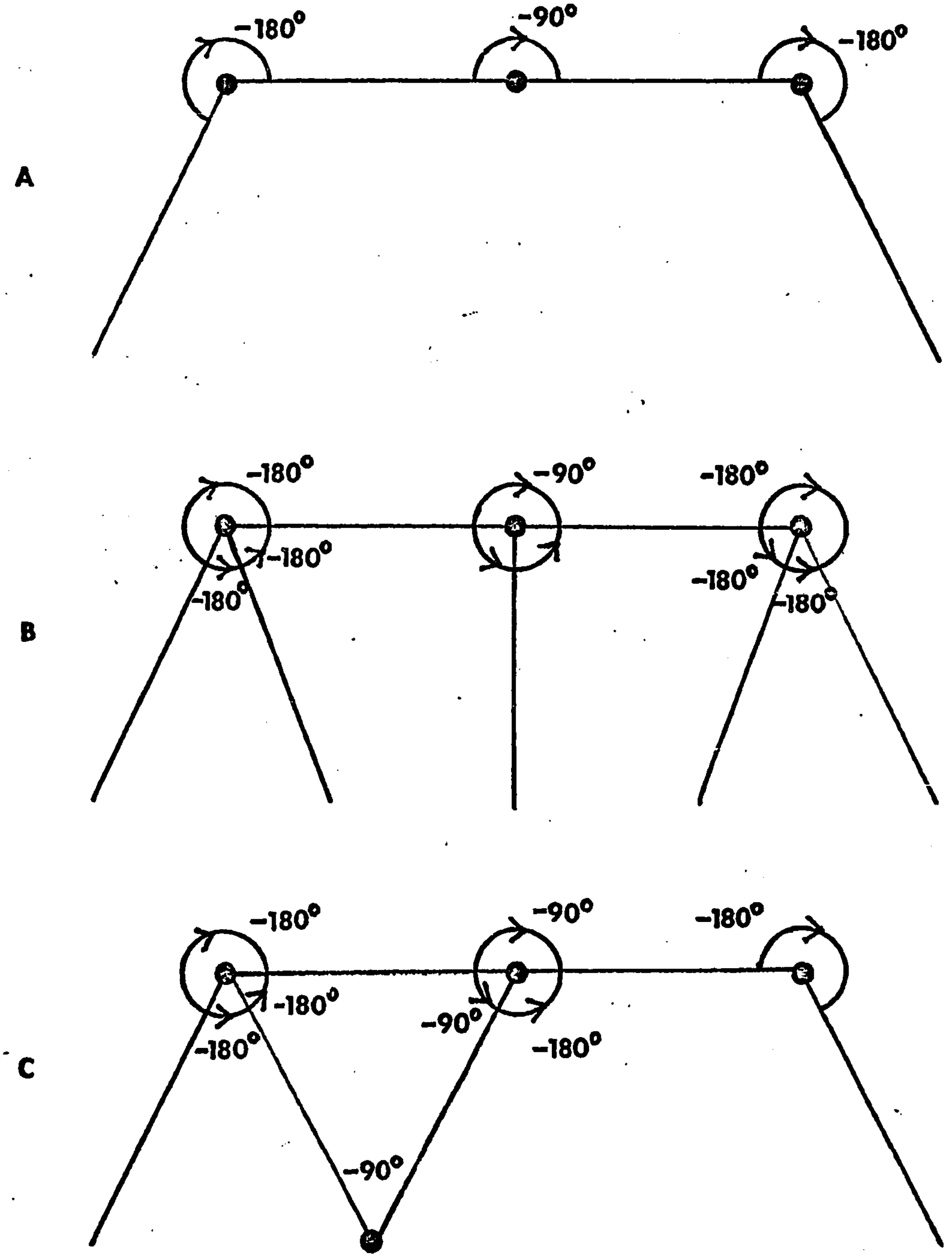


FIGURE A1-3

ILLUSTRATIONS FOR THE PROOF OF THEOREM 1

now obvious that, using mathematical induction, it can be shown that any $\pm180^{\circ}(\pm90^{\circ})\pm180^{\circ}$ sequence is physically unrealizable.

Next sufficiency will be proved. It will be shown that if a chain is not physically realizable then it contains a $\pm180^{\circ}$ $(\pm90^{\circ})\pm180^{\circ}$ sequence. Consider the following approach for constructing the physical realization of a chain: The rectangles corresponding to the nodes of the chain are constructed one at a time, proceeding from one end of the chain to the other. If the rectangles of a chain can be constructed in this "end to end" manner without overlapping, then the chain is physically realizable. Thus, if the chain is not physically realizable, the "end to end" construction cannot be made. To determine under which conditions the "end to end" construction cannot be made, three cases are considered.

Suppose that this "end to end" construction technique is being followed and the next rectangle in the chain is about to be constructed, all previous portions of the construction having been physically realizable. The next rectangle could call for any one of three different types of contiguity. (For this explanation, the most recently constructed rectangle in the chain will be called rectangle B, the rectangle immediately preceding B in the chain will be called rectangle A, and the rectangle immediately following B, i.e., the rectangle that is about to be constructed, will be called rectangle C.)

(1) The turn corresponding to rectangle B could be 0°. This means that rectangle A is contiguous to rectangle

B on a wall directly opposite the wall to which rectangle C is to be added (see Figure A1-4-A). Since no other rectangles are contiguous to rectangle B there must be room to add an arbitrary sized rectangle C as required.

(2) The turn corresponding to rectangle B could be ±90°. Because of symmetry, only the -90° case need be considered. This means that rectangle A is contiguous to rectangle B on a wall -90° from the wall to which rectangle C is to be attached. (See Figure A1-4-B.) Since no other rectangles are contiguous to rectangle B, there must be room to add an arbitrary sized rectangle C as required.

(3) The turn corresponding to rectangle B could be ±180°. Because of symmetry only the -180° case need be considered. This means that rectangle C must be contiguous to rectangle B along the same wall that rectangle A is contiguous to rectangle B. If the critical edge (marked by an "X" in Figure A1-4) of rectangle B is free, then this can be done (see Figure A1-4-C).

There are only two cases in which the connection of rectangle C would be impossible. These are shown in Figures A1-4-D and A1-4-E, in which the turn corresponding to rectangle A is -180° or -90°

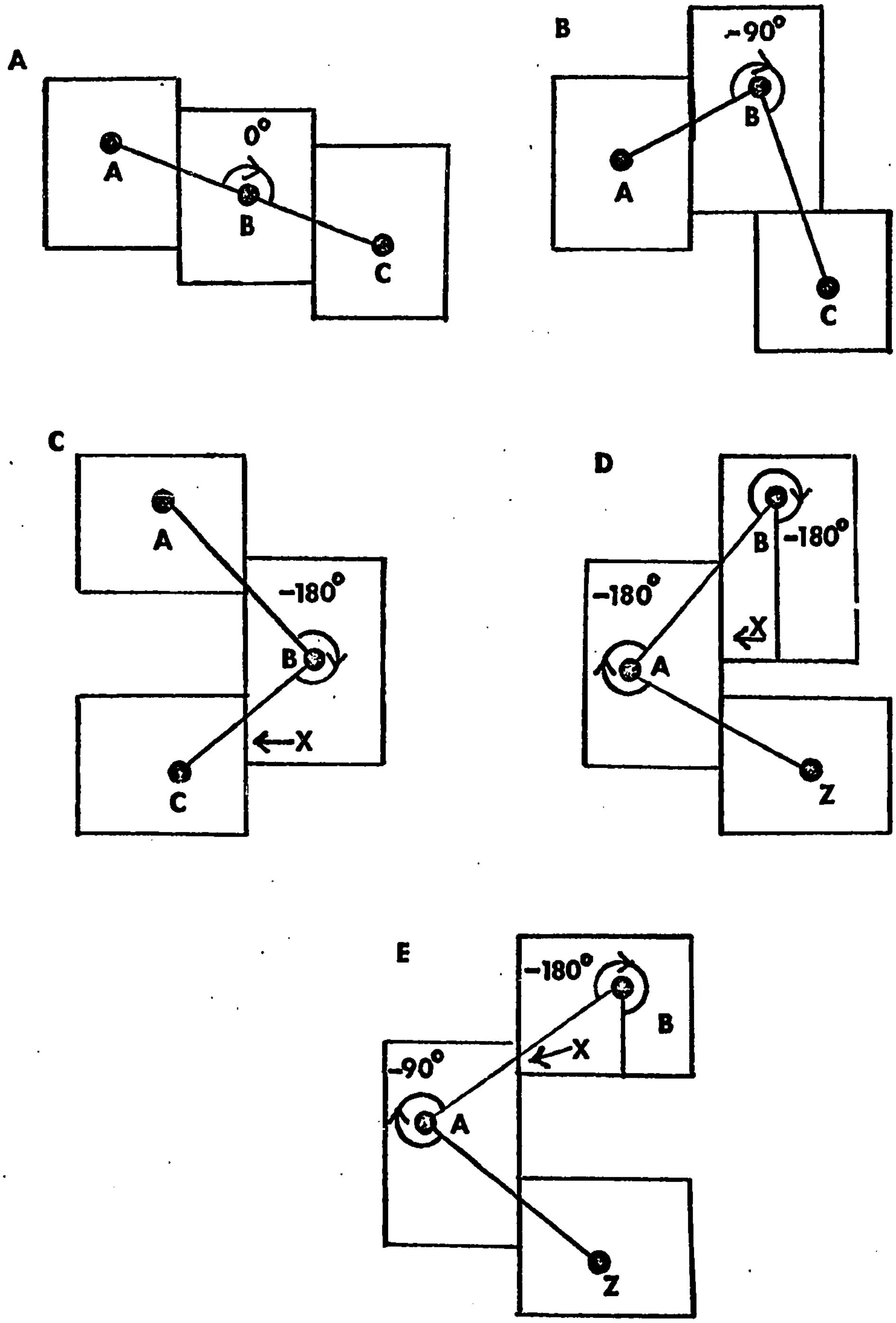


FIGURE A1-4

ILLUSTRATIONS FOR THE PROOF OF THEOREM 1

respectively, and the critical edge of rectangle B is blocked. In the case of the -90° turn (Figure A1-4-E) it is seen that in order for this to be any trouble, it must have been preceeded again by a -90° or a -180° turn. This, then, is the basis for the inductive conclusion that if a rectangle contiguity corresponding to a -180° turn cannot be made, then a $-180^{\circ}(-90^{\circ})-180^{\circ}$ sequence occurred in the chain.

Since the three cases exhaust the ways in which the next step can be taken in realizing the chain in the "end to end" fashion, it is concluded that if the "end to end" construction cannot be made, then a $\pm 180^{\circ}(\pm 90^{\circ})\pm 180^{\circ}$ sequence exists in the chain. It was shown previously that if the chain is not physically realizable, the "end to end" construction can't be made. Thus, by transitive implication, if the chain is not physically realizable, a $\pm 180^{\circ}(\pm 90^{\circ})\pm 180^{\circ}$ sequence exists in the chain, and sufficiency is proved.

A.1.2. Proof of Theorem 2

Theorem 2:

A non-terminal region bounded by unweighted edges is fillable if and only if the sum of the turns of its circumscribing circuit, taken in the clockwise direction, is -360° and the circuit contains no $\pm 180^{\circ}(\pm 90^{\circ})\pm 180^{\circ}$ sequences.

Proof

Sufficiency. -- An algorithm will be provided for constructing a "fill" of a region for which the two properties hold. In order to do this one needs the following two lemmas concerning physical realizability:

Lemma 2.1:
In a dual graph composed entirely of well formed terminal three sided regions (excepting, of course, the "outside region"), the edges surrounding all nodes are arranged in the proper order (see properties W1, W2).

Proof - A typical case of two neighboring well-formed terminal regions is shown in Figure A1-5-A.

Since the clockwise turns of well formed terminal regions are always -180° or -90°, then in enumerating the edges around any node in clockwise fashion, either the same type edge is maintained (-180° turns, see turn matrix, Figure 2-13), or a transition is made to the next appropriate edge type (-90° turns, see turn matrix). This constitutes the proof.

Note, however, that for well-formed nodes this cycle of edge types can only be traversed once. The mere concatenation of well-formed terminal regions about a node does not guarantee this. Only the correct order, not the number of cycles, is guaranteed by Lemma 2.1.

Preceding the second lemma, the following definitions are given:

Definition -- A physically realizable corner of a floor plan graph is one which corresponds to a physically realizable meeting of rectangles in a floor plan. The five physically realizable corners, corresponding to the five well-formed terminal regions, are shown in Figure A1-5-B. (See also Figure 2-12.)

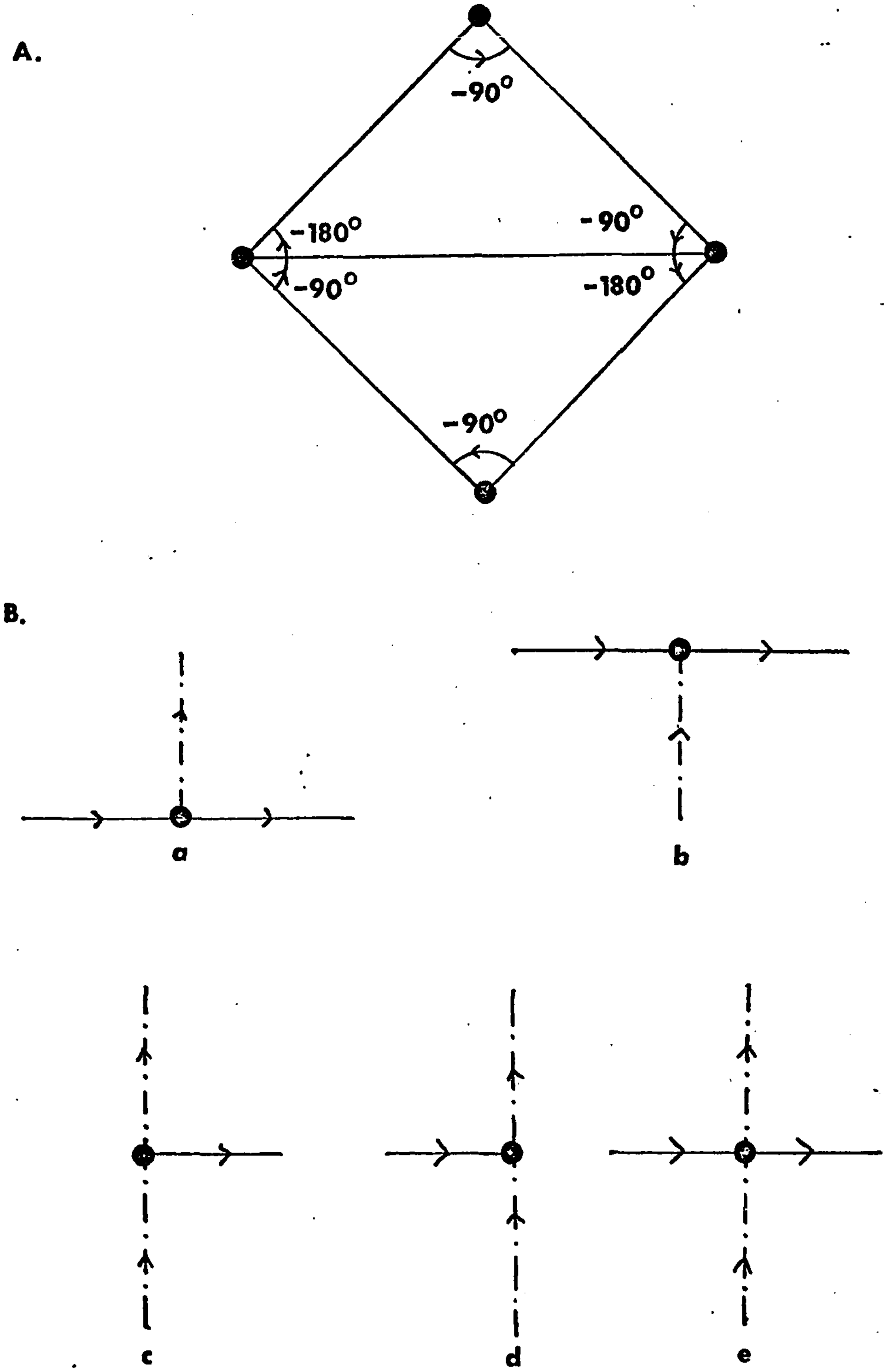


FIGURE A1-5

ILLUSTRATIONS FOR THE PROOF OF THEOREM 2

For identification purposes, the east-west wall segments of the floor plan graph will hereafter be colored solid (________) and directed from west to east and the north-south wall segments will be colored staggered (-•-•-•-) and directed from south to north.

Definition -- A wall segment is a colored, directed edge of the floor plan graph.

Definition -- A wall is a chain of wall segments of the same color and end to end direction.

Definition -- A complete wall is a wall which can't be extended on either end because it is attached to no more edges of the same color.

Definition -- A room of a floor plan graph is a space bounded by four walls, alternately colored staggered and solid, and properly directed, since the floor plan graph is by assumption physically realizable.

Definition -- A recognizable corner of a dual graph is either a corner that would be formed by one of its terminal regions, or a corner that can be deduced from the meeting of those rooms of the associated floor plan graph that correspond to nodes not completely surrounded by terminal regions. An example of a deduced recognizable corner is shown in Figure A1-6.

Lemma 2.2:

If a dual graph consisting entirely of well-formed terminal regions (except for the "outside" region) has the property that all nodes have at most one cycle of edges, and all recognizable corners of the potential corresponding floor plan graph are physically realizable, then that dual graph is physically realizable.

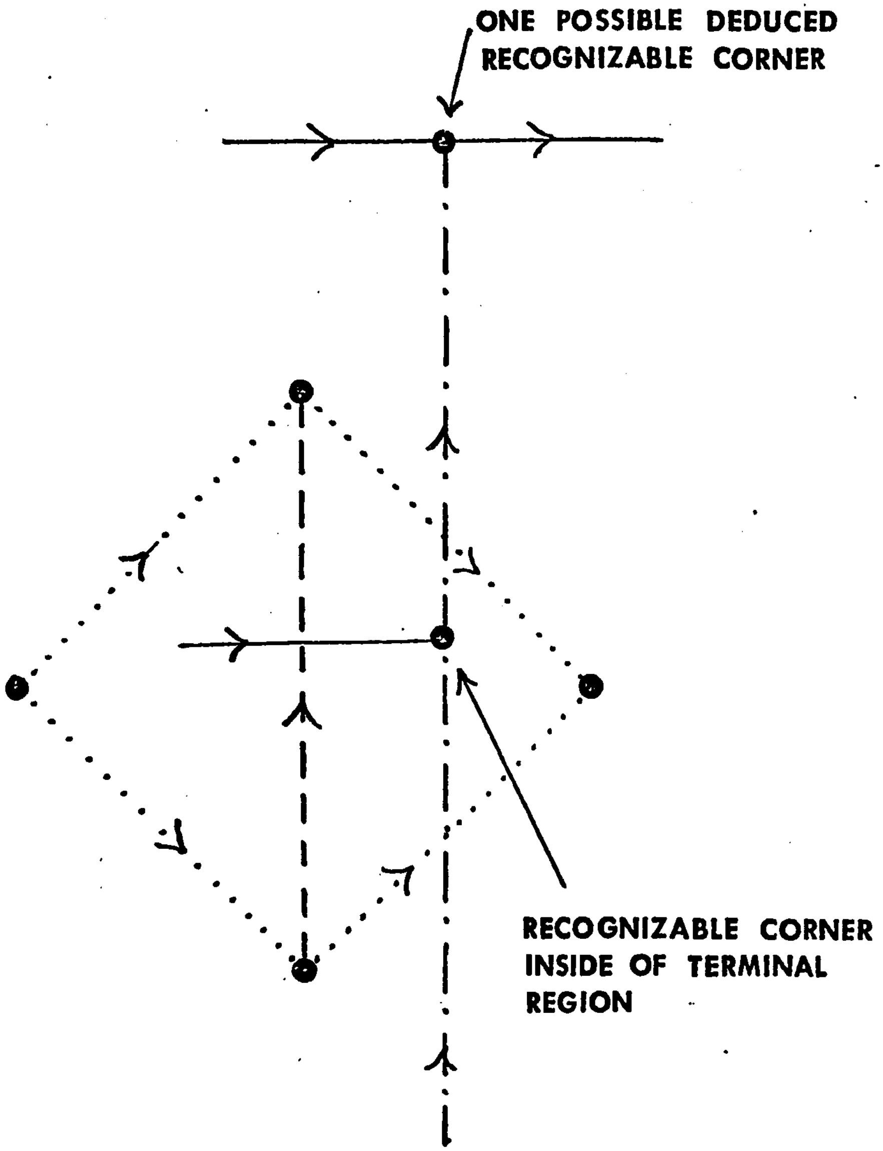

ONE POSSIBLE DEDUCED RECOGNIZABLE CORNER
RECOGNIZABLE CORNER INSIDE OF TERMINAL REGION

FIGURE A1-6

ILLUSTRATION FOR THE PROOF OF THEOREM 2

Proof - Since the graph consists entirely of well-formed terminal regions, Lemma 2.1 guarantees that the edges will be in the right order about the nodes. Furthermore, the condition of the lemma guarantees that there is at most one cycle of edges per node. Therefore it is possible to construct a unique room corresponding to each node of the dual graph (for there is nothing to prevent filling in walls for those nodes along the outside of the dual graphs that aren't completely bounded by terminal regions). Thus, since there exists a unique room for each node, since all recognizable corners are physically realizable (by condition of the lemma), and since the dual graph is planar (by definition), it must be possible to construct the corresponding floor plan graph, for all of its local structures are defined and well-formed and there will be no edges crossing in the plane.

Now it must be shown that the walls of the floor plan graph can be arranged in the proper orientations perpendicular to one another to produce a floor plan. This is proved by providing an algorithm for generating a floor plan from a floor plan graph. Consider for example the floor plan graph shown in Figure A1-7. It possesses the three properties of planarity, physically realizable corners, and rooms, provided by the lemma. The physically realizable corners guarantee that all unique complete walls can be identified and labelled, as shown in the Figure. Furthermore, the south to north ordering of the east-west complete walls and the west to east ordering of the north-south complete walls can be established by examining the order of walls adjacent to

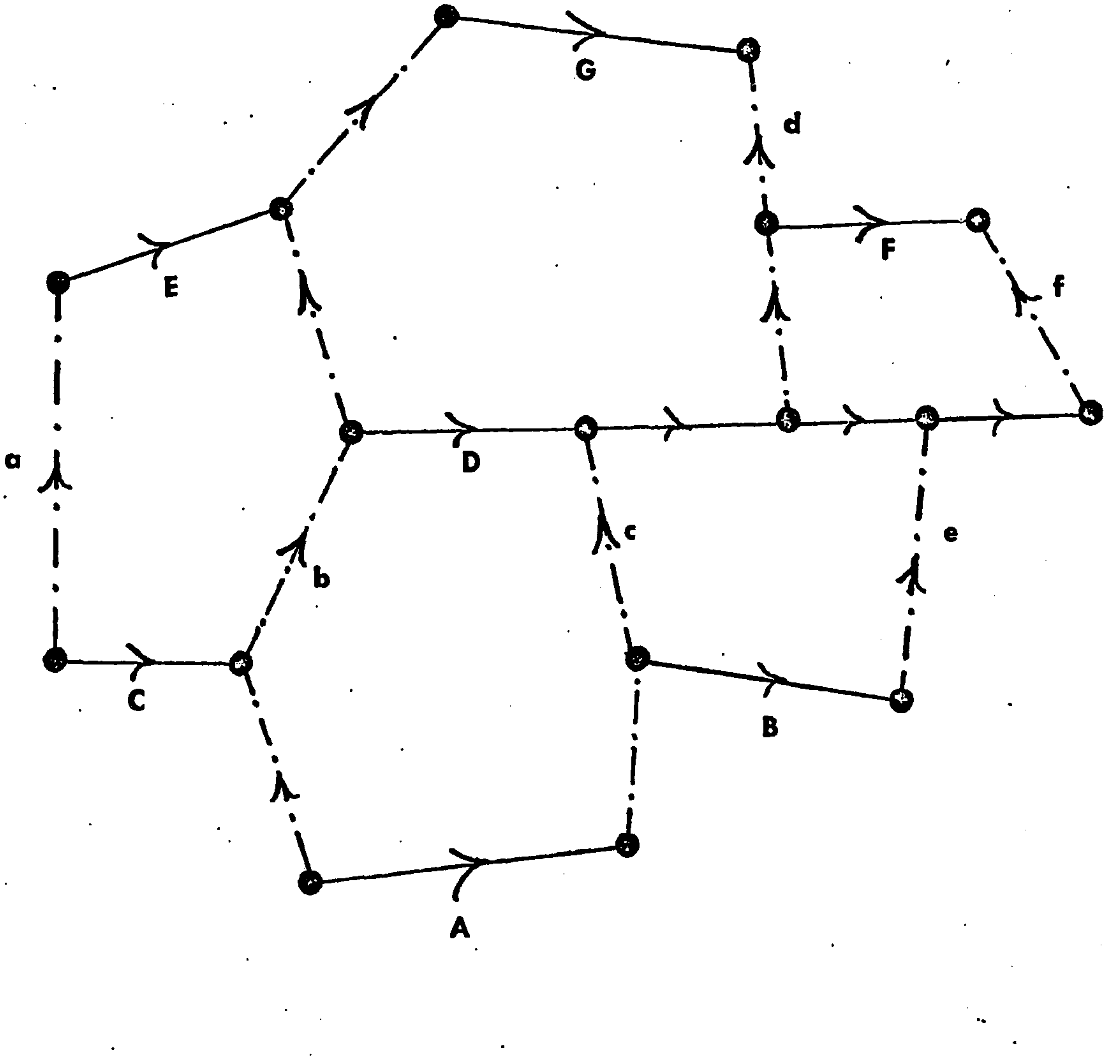


FIGURE A1-7

ILLUSTRATION FOR THE PROOF OF THEOREM 2

each room in turn, again because of the property of physically realizable corners. The ordering for Figure A1-7 is, for the east-west complete walls in a south to north fashion, A, (B,C), D, (E,F), G, where parentheses indicate that the ordering of the enclosed complete walls is arbitrary because these complete walls occur in the middle of opposite walls of a room. The ordering for the north-south complete walls in a west to east fashion is a,b,c,d,e,f. To generate the floor plan, the east-west complete walls are first layed out in order on a grid, as shown in Figure A1-8-A. Next the north-south complete walls are layed out perpendicular to the east-west complete walls one by one in order, as shown in Figures A1-8 and A1-9. It is clear from the physically realizable corners just where east-west complete walls should begin and end, so appropriate erasures are made, resulting in the floor plan of Figure A1-9-H.

Thus, if the conditions of the lemma are satisfied, the dual graph is physically realizable.

Now to return to the sufficiency portion of the proof of Theorem 2. It will be shown that, given a region for which the turn sum is -360° and for which there are no $\pm 180^\circ(\pm 90^\circ)\pm 180^\circ$ sequences, the interior of that region can be filled with well formed terminal regions so that a dual graph is created with at most one cycle of edges per node and with all recognizable corners physically realizable. By Lemma 2.2 then, such a dual graph would be physically realizable and constitute the desired well formed "fill" for the sufficiency proof. It is necessary

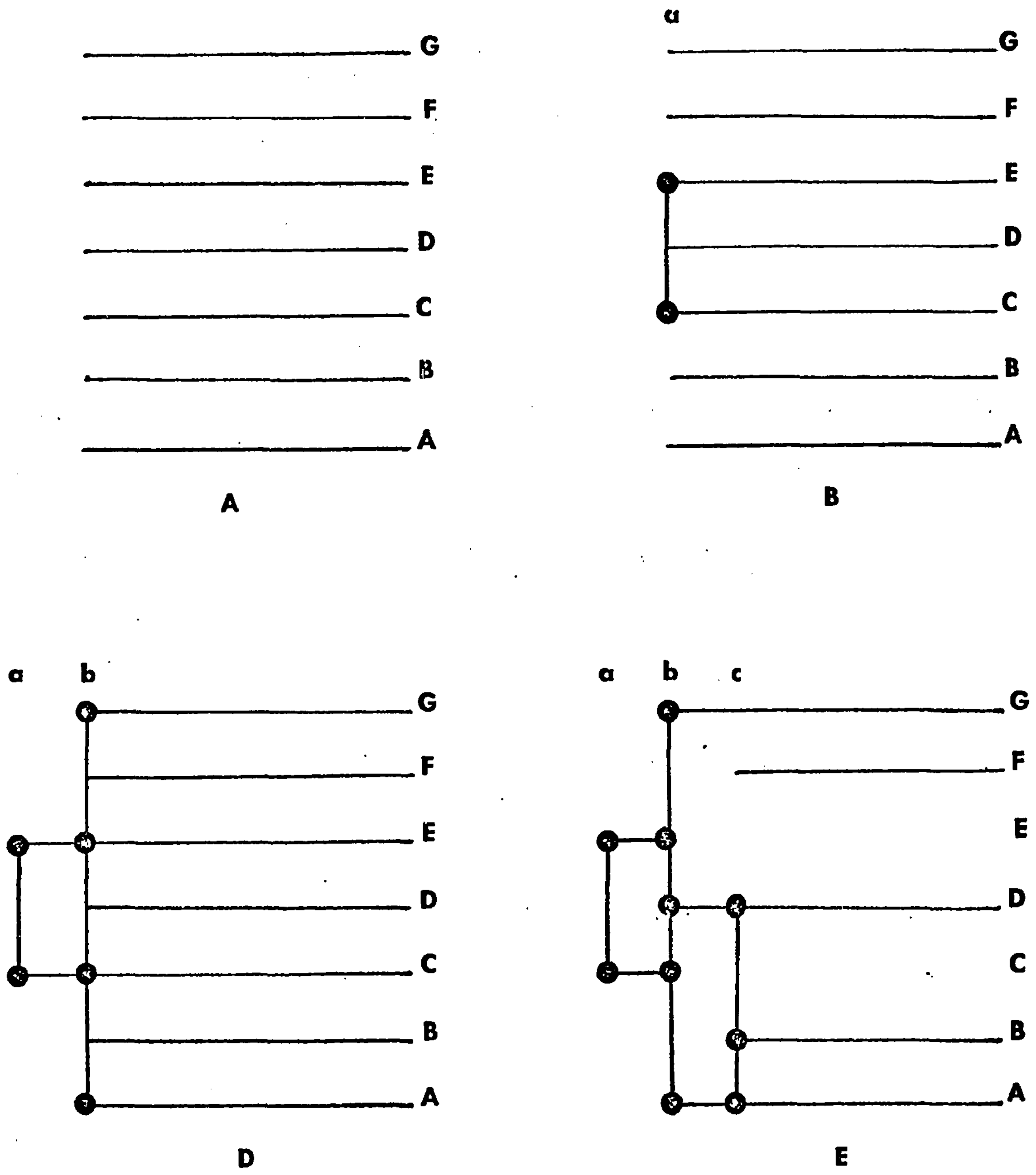


FIGURE A1-8

ILLUSTRATION FOR THE PROOF OF THEOREM 2

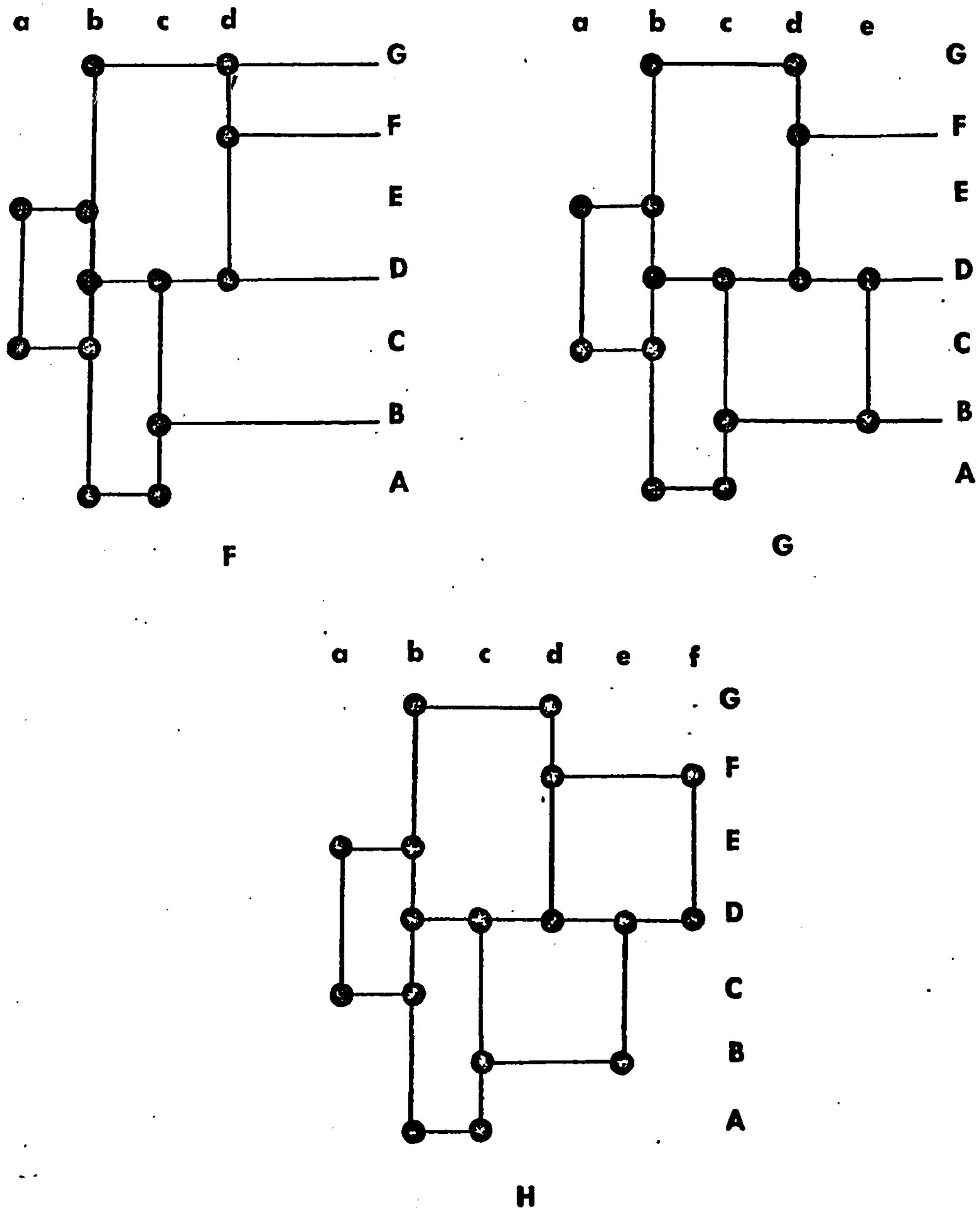


FIGURE A1-9

ILLUSTRATION FOR THE PROOF OF THEOREM 2

to provide first the algorithm for filling the interior of the region with well-formed terminal regions, and then to show that the resulting "fill" must satisfy the physical realizability requirements of Lemma 2.2.

The region under consideration can be thought of as a two dimensional convex polygon. The subdivision into terminal regions can be carried out by introducing edges one at a time, each of which subdivides the previous polygonal region into a triangle plus a region with one less side. Thus one must show that, given a region for which the sum of the turns is -360° and for which there are no $\pm 180^\circ (\pm 90^\circ) \pm 180^\circ$ sequences, one can construct a "legal lop"; that is, introduce an edge that subdivides that region into a well-formed terminal triangle plus a resultant region that also satisfies the above two conditions.

In constructing a lop, since the triangle that is generated must correspond to a well formed terminal region (see property W2*), the node across which it is constructed must have a -90° or -180° turn associated with it (see Figure A1-10 for instance). Furthermore, the turns of the region that are generated at the two ends of the subdividing edge are calculated according to property W1* (see Figure A1-10-B).

From property W1* it is seen that in adding up the turns around a node in the counter clockwise direction, a +180° virtual turn is added for every edge that is traversed. Therefore, once the triangle is constructed, the sum of the actual turns on the inside of the original region (labeled area in Figure A1-10) is

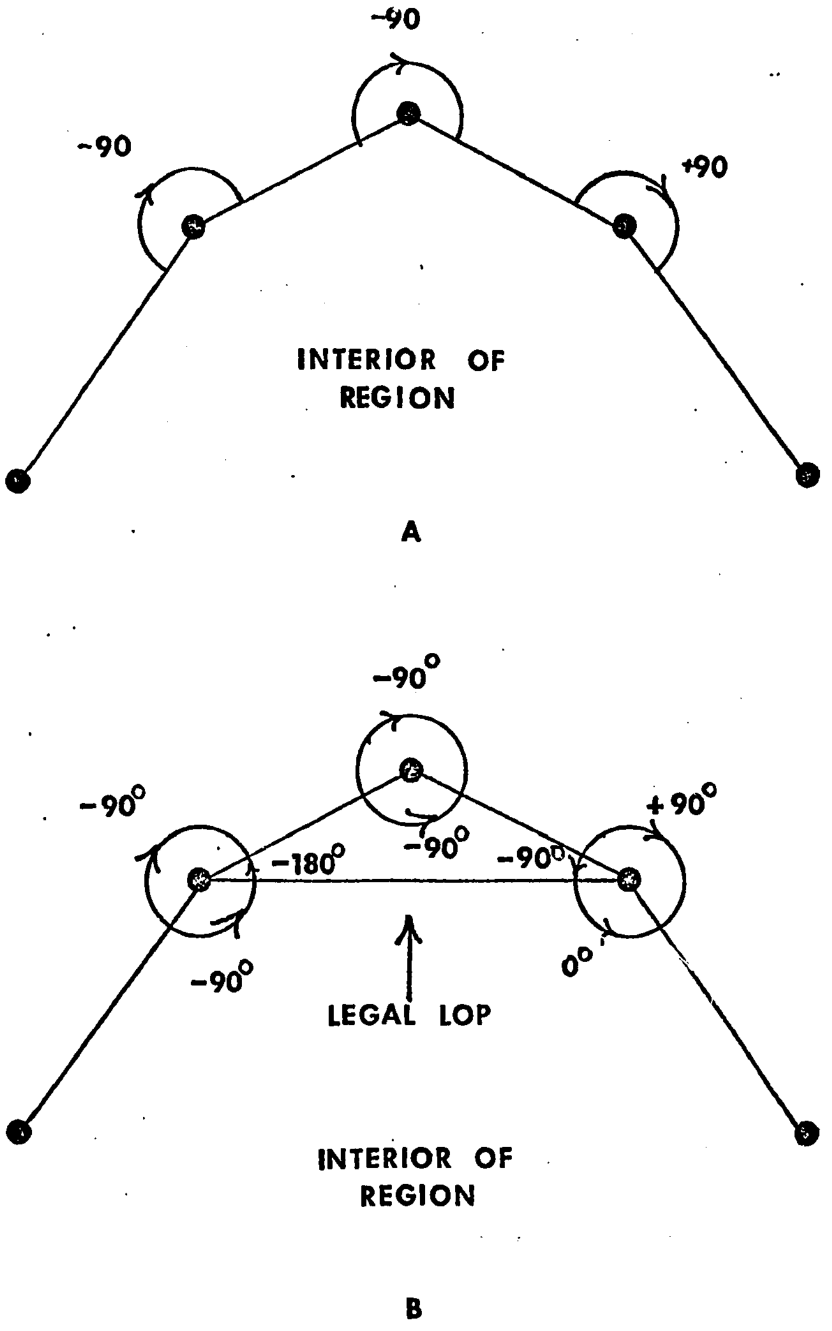


FIGURE A1-10

ILLUSTRATION FOR THE PROOF OF THEOREM 2

equal to the sum of the turns on the outside of the original region minus 360° (i.e., for Figure A1-10-B,

$$\overbrace{(-90^\circ)+(-180^\circ)+(-90^\circ)+(-90^\circ)+(0^\circ)}^{\text{inside turns}} = \overbrace{(-90^\circ)+(-90^\circ)+(+90^\circ)+(-360^\circ)}^{\text{outside turns}}).$$

However, the sum of the turns inside the triangle is also -360° (see property W2*). Since these turns do not count in summing up the turns of the resultant region formed by the lop, one has the fact that when a lop is made, the sum of the turns of the original region is equal to the sum of the turns of the resultant region. Thus, if the sum is -360° before a lop, it is -360° after a lop, and the first condition of the theorem is maintained.

The fact that a lop maintains the absence of the $\pm 180^\circ(\pm 90^\circ)\pm 180^\circ$ sequence is now treated. It will first be shown that the property can be maintained in regions which contain no "plus" turns (+90° or +180°; hereafter, for brevity, $\pm 180^\circ$, $\pm 90^\circ$, and 0° will be abbreviated +2, +1, and 0 respectively). There are only three combinations of turns that can produce a region with a turn sum of -4(-360°) and no plus turns. They are:

(1) -1, -1, -1, -1 with an arbitrary number of interspersed zeros.

(2) -2, -1, -1 with an arbitrary number of interspersed zeros.

(3) -2, -2 with an arbitrary number of interspersed zeros.

Consider now the 6 lops shown in case 1 above, one of the lops A1-11-A, B, or C must be possible. None of these introduces a $\pm 2(\pm 1)\pm 2$ $(\pm 180^{\circ}(\pm 90^{\circ})\pm 180^{\circ})$ sequence. For case 2, one of the lops A1-11-A, B, D, E, or F must be possible (unless the region is a triangle, in which case no lop is necessary), and none of these can introduce a $\pm 2(\pm 1)\pm 2$ sequence. For case 3, lop F must be possible, since no $\pm 2(\pm 1)\pm 2$ sequences exist already, and this lop does not introduce any such sequences.

Now consider the remaining class of regions, those which contain at least one plus turn. In such a region there must exist a minus turn (-1 or -2, which is necessary for the vertex of a lop) located next to either a plus turn or a string of zeros which end in a plus turn.

Consider the case of a minus turn located next to a zero. The prospective lops in which this can occur are shown in Figure A1-11-A, B, D, E, and F, the zero always being on the right and the other turn being non-plus, as plus turns will be considered later. Lops A, B, and F can introduce no $\pm 2(\pm 1)\pm 2$ sequences, because the zero on the right is followed by a (possibly empty) string of zeros and a plus turn. Lops D and E can introduce no $\pm 2(\pm 1)\pm 2$ sequences on the right for the same reason as lops A, B, and F, and can introduce no $\pm 2(\pm 1)\pm 2$ sequences on the left because that would imply that such a sequence already existed in the region, which is impossible by hypothesis.

Finally, consider the case of a minus turn located next to a plus turn. It will be shown that every three turn sequence with

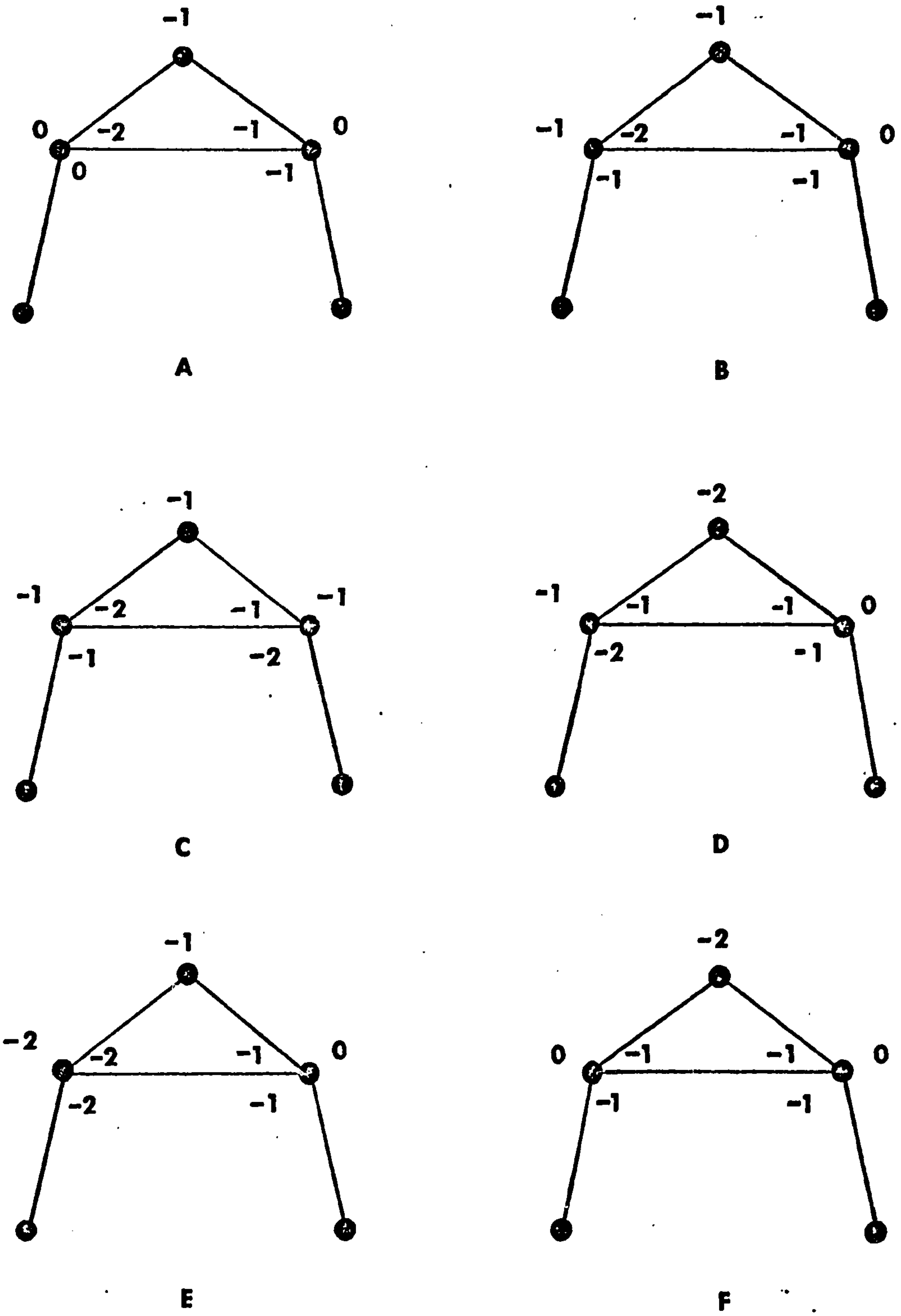


FIGURE A1-11

ILLUSTRATION FOR THE PROOF OF THEOREM 2

a minus turn in the middle and containing a plus turn can generate a lop which preserves the property of no $\pm 2(\pm 1)\pm 2$ sequences. First -2(-1)-2 sequences will be considered. For the purpose for defining terms, consider the typical lop shown in Figure A1-12-Z. Turns A, B, and C will be called the three outside turns, turns a and b the inside turns corresponding to A and B, and turns C, t_1 and t_2 the triangle turns. Given the fact that all triangle turns are -1 or -2, several conclusions about inside turns can be drawn using property W1*.

(1) A plus outside turn will always generate a zero or plus inside turn, and can do so for either -1 or -2 triangle turns.

(2) A -2 inside turn can only be generated by a -1 outside turn and a -1 triangle turn, or a -2 outside turn and a -2 triangle turn.

(3) The triangle turn corresponding to a -2 outside turn must be -2, and the resultant inside turn will also be -2.

Conclusion 1 indicates that a three turn sequence with one plus turn will always generate one zero or plus inside turn. Conclusions 1, 2 and 3 together indicate that if the non-plus, non-vertex outside turn is -1, and the -2 triangle turn can be placed next to it and a -1 inside turn will result. If the non-plus, non-vertex outside turn is -2, then the corresponding triangle turn and the resultant inside turn must both be -2. However,

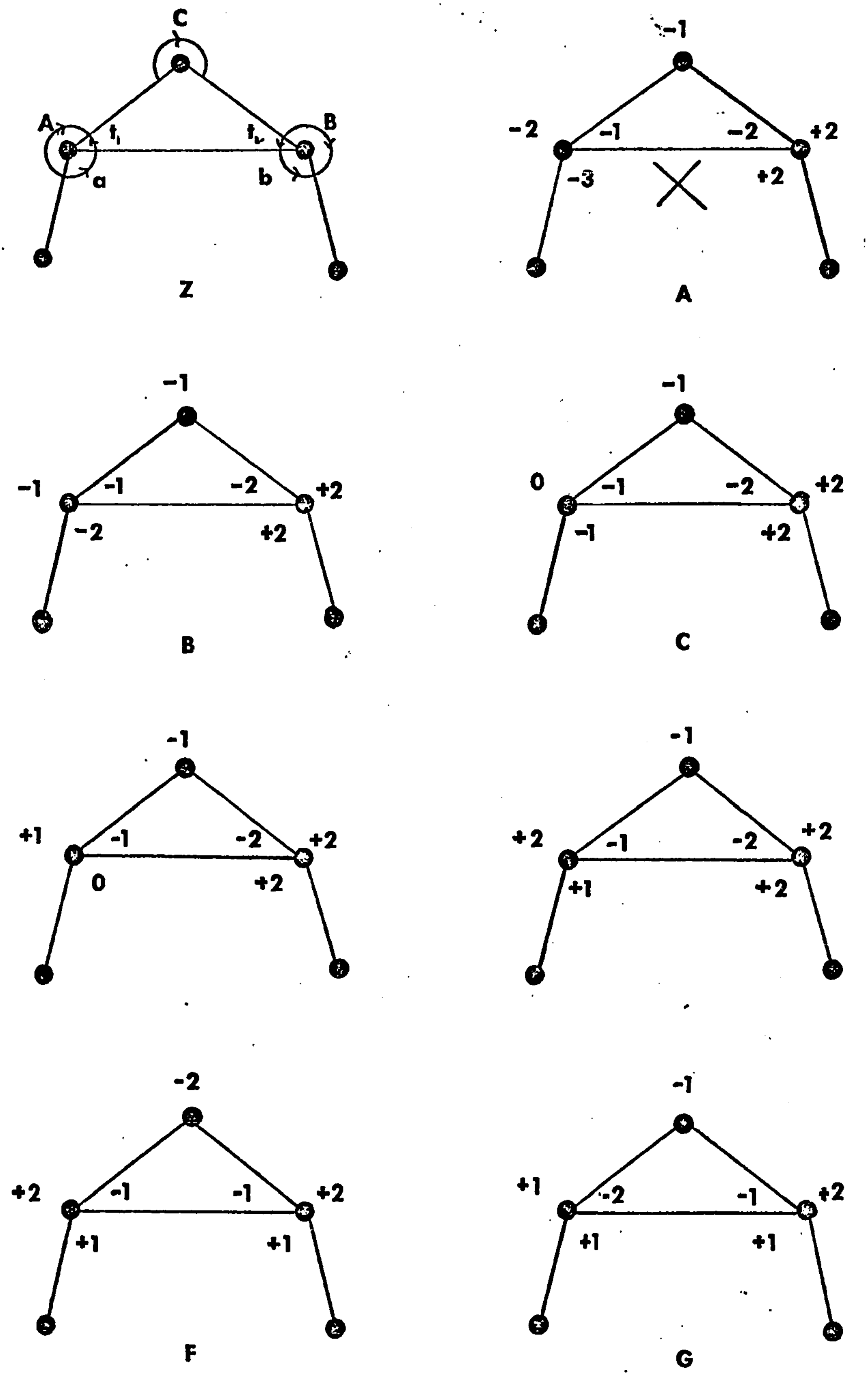


FIGURE A1-12

ILLUSTRATION FOR THE PROOF OF THEOREM 2

since one of the two inside turns will always be non-minus, and since the worst the other inside turn can be is a -2, corresponding to a -2 outside turn, it is seen that no new -2(-1)-2 sequences can be generated by the lop if none existed before.

Now the remaining case of +2(+1)+2 sequences will be considered, for lops taken from three-turn sequences containing at least one plus turn. There is potential danger of generating a +2(+1)+2 sequence if either inside turn is +2 or if both inside turns are +1. There are seven lops in which this can occur, and these are shown in Figure A1-12-A through G. The enumeration is complete because by property W1* it takes a +2 outside turn and a -2 triangle turn to generate a +2 inside turn, and a +2 or +1 outside turn accompanied by -1 or -2 inside turn to generate a +1 inside turn. Lop A, though not possible because of the necessity of a -3 turn, presents no problem because if the -2 and -1 triangle turns are reversed, the inside turns become -2 and +1 and it is covered under the -2(-1)-2 sequence case just discussed. For lops B, C, D, E, F, and G, to generate a +2(+1)+2 sequence, one would have to have existed previously, which is impossible by hypothesis. Thus all region types (those containing no plus turns and those containing plus turns) and all sequence types (-2(-1)-2 and +2(+1)+2) have been covered and it has been shown that for any region, the sum of whose turns is -4 and which contains no ±2(±1)±2 sequences, a lop can be made creating a resultant well-formed triangular terminal region and another (possibly) nonterminal region which also satisfies the above two conditions.

Now to complete the proof of sufficiency it must be shown that the dual graph resulting from the subdivision just described has at most one cycle of edges per node, and the property that all recognizable corners are physically realizable, so that Lemma 2.2 can be applied.

The fact that property W1* was used in generating the inside turns resulting from the lops guarantees that there is at most one cycle of edges per node, since none of the turns of the original region exceeded +2, the highest possible well-formed turn.

With respect to the corners, first note that all corners generated inside of well—formed terminal regions are by definition physically realizable. That leaves to be considered those corners formed by rooms bordering one another along the outside edges of the dual graph that has been formed. Since the proof has been limited to three-wall-segment corners, each of these outside corners must have two wall segments of one color and direction, and one wall segment of the other color. The fact that the pairs of rooms under consideration are contiguous to one another insures that if the colors of the wall segments of the corner are right, the directions will be right, for two of the three wall segments will be from one wall of a room, and it is already known that these are unidirectional. This leaves one with the task of showing that the colors will be right if the conditions of the theorem are satisfied. There are 5 types of outside turns corresponding to the nodes along the outside edge of the dual graph.

One can use arbitrarily colored wall segments and edges without loss of generality.

(1) -2 outside turns (see Figure A1-13-A) -- In this case the two corners that will be generated with the neighboring rooms (see two nodes with ends of three wall segments) already have two different colored wall segments, so the wall segment from the neighboring rooms can be any color and still result in a physically realizable corner.

(2) -1 outside turns (see Figure A1-13-B) --Same argument as case 1.

(3) 0 outside turns (see Figure A1-13-C) -- Here, the use of property W1* in generating the lops insures that the north-south edge will have been inserted in the 0 turn during the subdivision process, thus creating the room shown. The argument for the physically realizable corners is still the same.

(4) +1 outside turns (see Figure A1-13-D) -- Here the case is somewhat different. In general, to insure three wall corners, one of the walls outside will have to be broken into two wall segments. Just which wall it will be depends on what the wall segments from the two neighboring rooms are, thus the question mark. If the wall segments from both neighboring rooms are the same color, one of the resultant corners cannot be physically realizable. It shall be shown in case 5 that this only happens when

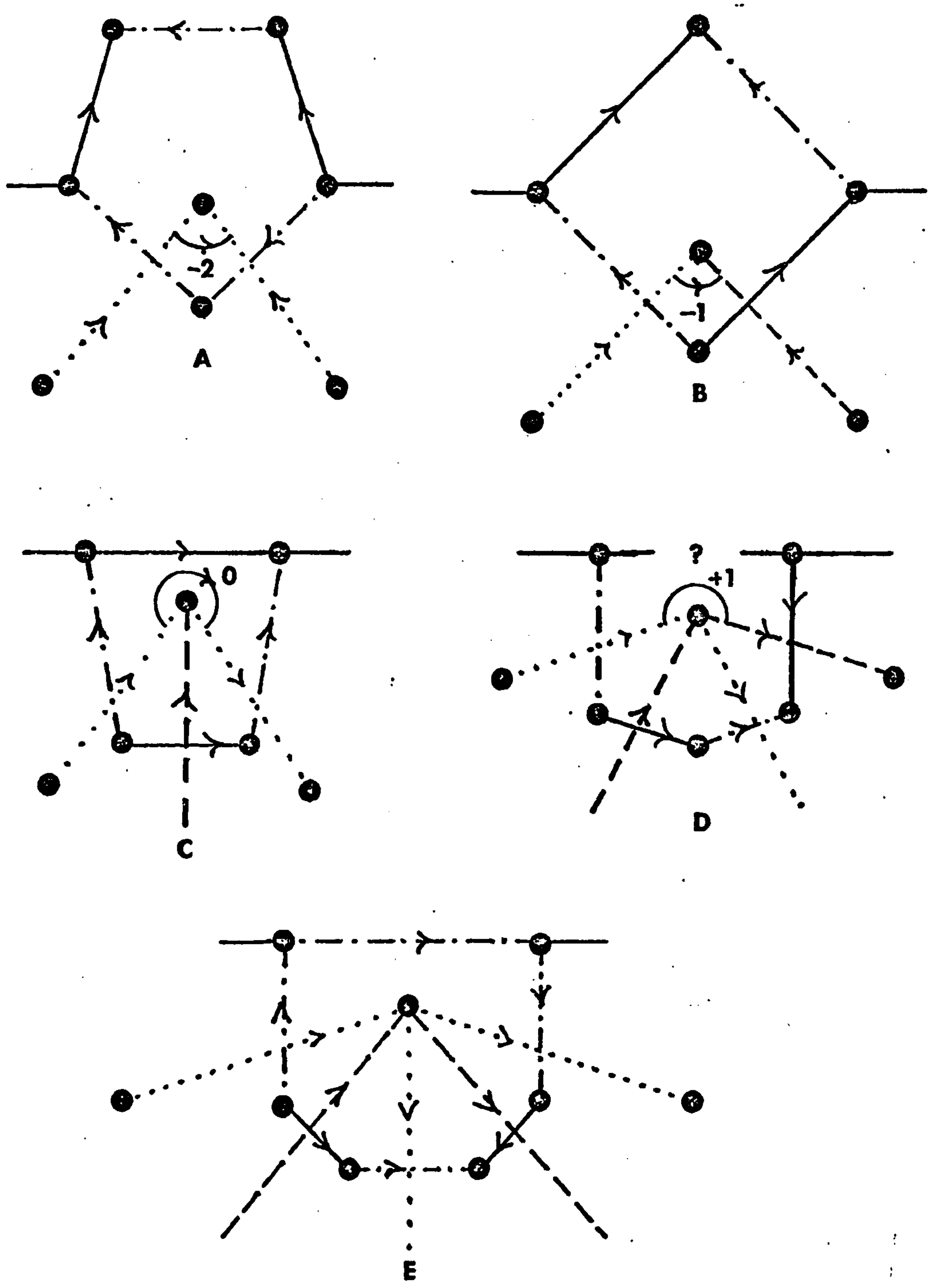


FIGURE A1-13

ILLUSTRATION FOR THE PROOF OF THEOREM 2

a +2(+1)+2 sequence occurs, and such sequences are not allowed by the conditions of the theorem.

(5) +2 outside turns (see Figure A1-13-E) -- Here the wall segments from the two neighboring rooms must be the same color, solid in this case. If either neighbor is also a +2 turn it will also only have staggered wall segments to give, and so that corner will not be physically realizable. But this would mean that there was a +2, +2 sequence, which is not allowed. If either neighbor were a +1 turn, then its corresponding undetermined wall segment would then be determined, as would those of any other +1 turns next to it. However, if that string of +1 turns terminated in a +2 turn, the corresponding corner would not be physically realizable because all three wall segments would be the same color. Again, however, this situation cannot occur because +2(+1)+2 turn sequences are not allowed. Finally, cases 1 through 3 show that if the neighboring turns were 0, -1, or -2, physically realizable corners would result.

Thus, if the conditions of the theorem are satisfied, all recognizable corners are physically realizable and sufficiency is proved.

Necessity. -- A physically realizable dual graph is by definition one for which a corresponding floor plan can be drawn. Thus, any fillable region can be generated by considering the dual graph of an appropriately constructed floor plan--that is, a floor plan in which no rectangular space is completely surrounded by other rectangular spaces, in which every rectangle is contiguous to at least two other rectangles, and which fills a simply connected area. If only the outside edges of that dual graph are preserved, the desired fillable region results.

Since that region was generated from a physically realizable floor plan, it can contain no ±2(±1)±2 sequences, by Theorem 1, since a circuit is simply a chain closed on itself.

Finally, the fillable region can be thought of as having been generated by starting with one of the terminal regions of the dual graph and attaching further regions to it one at a time, erasing interior edges along the way, until the final region was achieved. In each case that a new terminal region was attached, it can be argued in exactly the reverse manner as was done in the sufficiency proof that, since all terminal regions have a turn sum of -4 and since the first region was a terminal region, the turn sum of the resultant fillable region must be -4. Thus necessity is established and Theorem 2 is proved.

A.1.3. Proof of Theorem 3

Theorem 3:
A circuit of unweighted edges is physically realizable if and only if the sum of its turns taken in the clockwise direction is -360^o and it contains no $\pm 180^o(\pm 90^o)\pm 180^o$ sequence.

Proof - As in the proof of Theorem 3, $\pm 180^c$, $\pm 90^o$, and 0^o will be abbreviated as ±2, ±1, and 0 respectively in this proof.

Sufficiency. -- The realization is based on the fill, which it is known can be generated, by reference to Theorem 2. Such a realization satisfies the contiguity relationships of the circuit in question. However it includes other contiguity relationships as well, which result from the region having been filled in with terminal regions. These extra contiguities can be eliminated by shrinking and stretching appropriate rectangles in appropriate manners in the floor plan. All lops were made with -1 or -2 turns as vertices, and because of the arbitrary dimensions of the rectangles, all extra contiguities that were generated in this way can be drawn back by shrinking a rectangle. An example is shown in Figure A1-14-A, B, and C.

Necessity. -- Again, all physically realizable circuits can be generated by taking a special dual graph (no links to outside spaces) of an actual partially completed floor plan. Since a necessary condition for a chain to be physically realizable is the absence of the ±2(±1)±2 sequence, it is a necessary condition for physically realizable circuits as well.

As for the -4 turn sum, note that the edges of the dual graph for a physically realized circuit can all be oriented either north-south or east-west with corresponding 0° or $\pm 90^{\circ}$ turns from edge to edge. For $\pm 180^{\circ}$ turns it is necessary to make two turns (which still add up to 180°) and for some 0° turns it may be necessary to do a $+90^{\circ}$, -90° jog, but this still adds up to 0°. See Figure A1-14-D for an example.

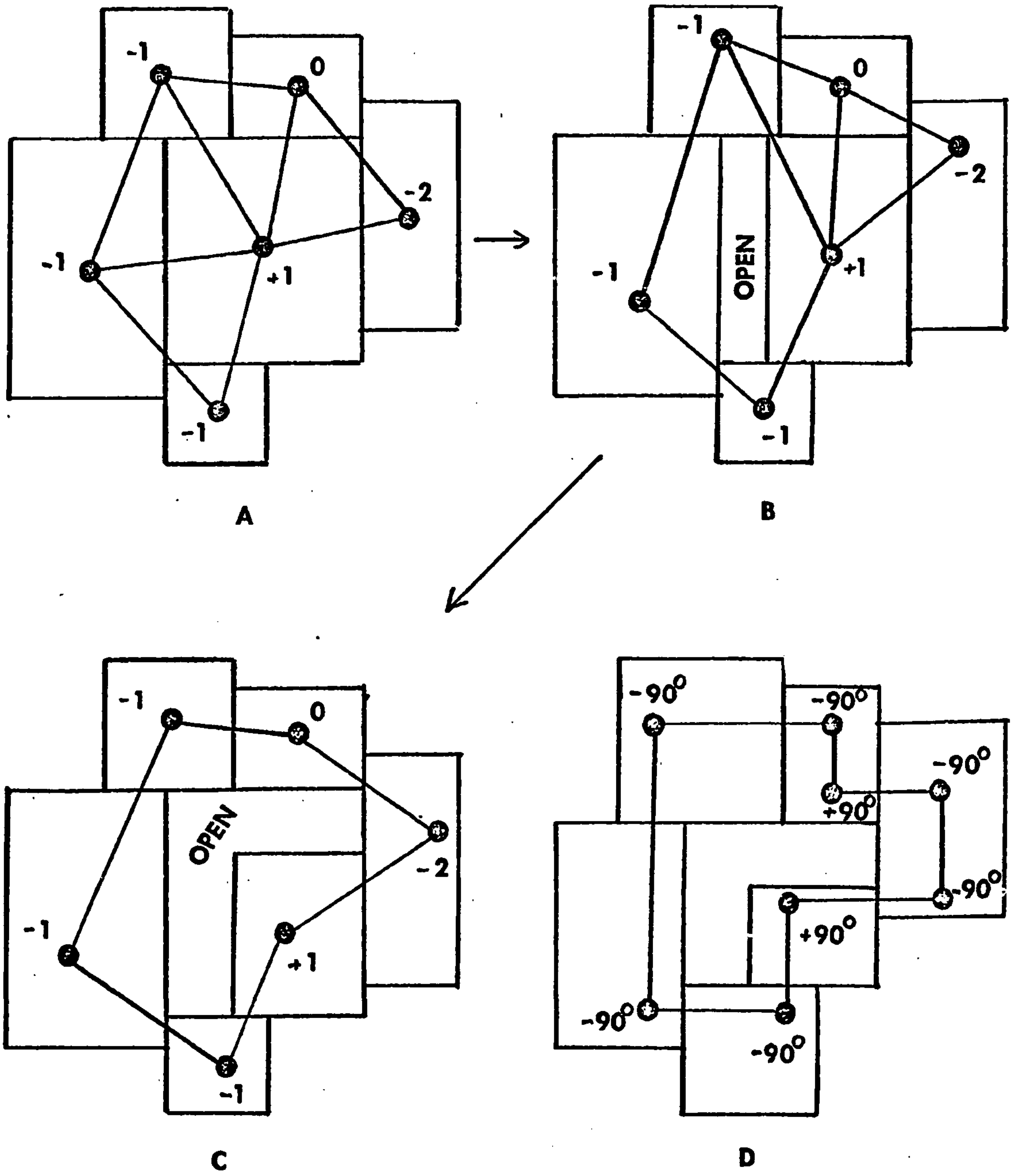


FIGURE A1-14

ILLUSTRATIONS FOR THE PROOF OF THEOREM 3

Since these ±90° turns correspond directly to the ±1 turns of the edges of the dual graph, the sum of the turns should equal the total angle traversed divided by 90°. But the Figure that has been constructed is a simple closed curve for which we know that the total angle traversed must be -360°. Thus the turn sum for a physically realizable circuit must be -4, and necessity is proved.

A.1.4. Proof of Corollary 3.1

The region corresponding to a physically realizable circuit of unweighted edges is fillable.

Proof -- By Theorem 3, if a circuit is physically realizable, it has a -360° turn sum and contains no ±180°(±90°)±180° sequence. Thus by Theorem 2, the circuit is fillable.

A.1.5. Proof of Theorem F1

See Section 3.2 (page 93) for the statement of the theorem. The proof is as follows:

It must be shown that if any one of the three conditions is not satisfied it is possible for the program to generate an infeasible partially completed design solution. Each condition can be considered separately.

(1) If condition (a) is not satisfied, the partially completed design solution will be infeasible by the definition of feasibility (see Section 3.1.).

(2) If condition (b) is not satisfied, the dual graph of the partially completed design solution will not be physically realizable, and hence the partial solution will be infeasible.

(3) The same thing holds for condition (c) as for condition (b).

A.1.6. Proof of Theorem F2

See Section 3.2 (page 93) for the statement of the theorem. The proof is divided into sections on necessity and sufficiency.

Necessity. -- It must be shown that if any one of the three conditions is not satisfied it is not possible for the program to generate all feasible partially completed design solutions. Each condition will be considered separately.

(1) If a planar graph rejected by the planarity check otherwise corresponds to a feasible partial solution, then a feasible partial solution will have been missed by the program.

(2) If a planar realization of the dual graph that was skipped corresponded to a feasible partial solution, then a feasible partial solution will have been skipped.

(3) The same reasoning holds for condition (c) as held for the first two conditions.

Sufficiency. -- It must be shown that if a feasible partial solution was skipped by the program, then at least one of the conditions of the theorem was not satisfied. The reasoning is as follows:

There are only two ways that a feasible partial solution could have been skipped by the program. First, it may not have been considered as a possible solution, and second, it may have been considered but was rejected. These cases are taken up individually below.

(1) If a feasible partial solution was never considered, this must mean one of three things.

(a) It was never generated as a planar realization of the adjacency requirement dual graph. In this case, condition (b) of the theorem was not satisfied.

(b) Some feasible partial solution leading up to it was incorrectly rejected. This reduces to a consideration of case (2) below.

(c) Some infeasible partial solution leading up to it was correctly rejected, hence it was never generated. Now the only two ways a partial solution can be rejected in Phase I are by the planarity test or by the well-formedness tests. Thus, for a feasible solution to follow from an infeasible solution would mean that conditions (a) and/or (c) of the theorem did not hold.

(2) If a feasible partial solution was rejected, this must mean that conditions (a) and/or (b) did not hold, since the planarity test and the well-formedness tests are the only ways of rejecting a partial solution.

A.1.7. Proof of Theorem F3

See Section 3.3 (page 146) for the statement of this theorem. The proof is divided into sections on necessity and sufficiency.

Necessity. -- This proof relies on the definition of feasibility in exactly the same way as the necessity part of the proofs of Theorems F1 and F2.

Sufficiency. -- It must be shown that if a completely specified graph is infeasible, then one of the three conditions is not satisfied.

One way for a completely specified solution to be infeasible is if not all design requirements specified for the problem have been satisfied. This would mean that condition (c) was not satisfied.

The only other way for a completely specified solution to be infeasible is if it is not physically realizable. A solution is considered to be physically unrealizable if it contains a nonrectangular room specification or if it contains a physically unrealizable corner at which rooms are to meet. If either of these is true then conditions (a) and/or (b) of the theorem are not satisfied.

A.1.8. Proof of Theorem F4

See Section 3.3 (page 146) for the statement of this theorem. The proof of this theorem exactly parallels the proof of Theorem F2.

A.1.9. Proof of R86 Deduction Routine

Routine R86 can deduce the length of a wall segment of a wall of a room, provided it is not the only wall segment of that wall and all other wall segments of that wall have been already dimensioned. This section is to show that if all single wall segment room walls in a floor plan have been dimensioned (R85 and R87 do this), R86 can dimension all the rest.[1]

Call any wall segment that takes up the entire wall of at least one of the two rooms that border it a type 1 wall segment. Call all other wall segments type 2. Then R85 and R87 dimension type 1 wall segments and it is up to R86 to dimension all type 2 wall segments.

It should be clear that any room wall that contains at most one type 2 wall segment will present no difficulties to R86. There are only two different ways a wall may contain two type 2 wall segments (the maximum possible number), and these are shown in Figures A1-15-A and B. The only way configurations of this type can present any difficulty is if there is an unbroken string of them running across a floor plan, as shown in Figure A1-15-C. However, such a string always must terminate at an outside wall and is therefore accompanied by a type 1 wall segment. This then leads to the deduction of all the type 2 wall segments along the string.

For instance, in Figure A1-15-C, once the type 1 wall segments A, B, and G are deduced by R85 or R87, the type 2 wall segments C, D, E, and F can be deduced in cascade by R86. Since

[1] Recall that a wall segment is the portion of a wall that is bordered in common by two given rooms.

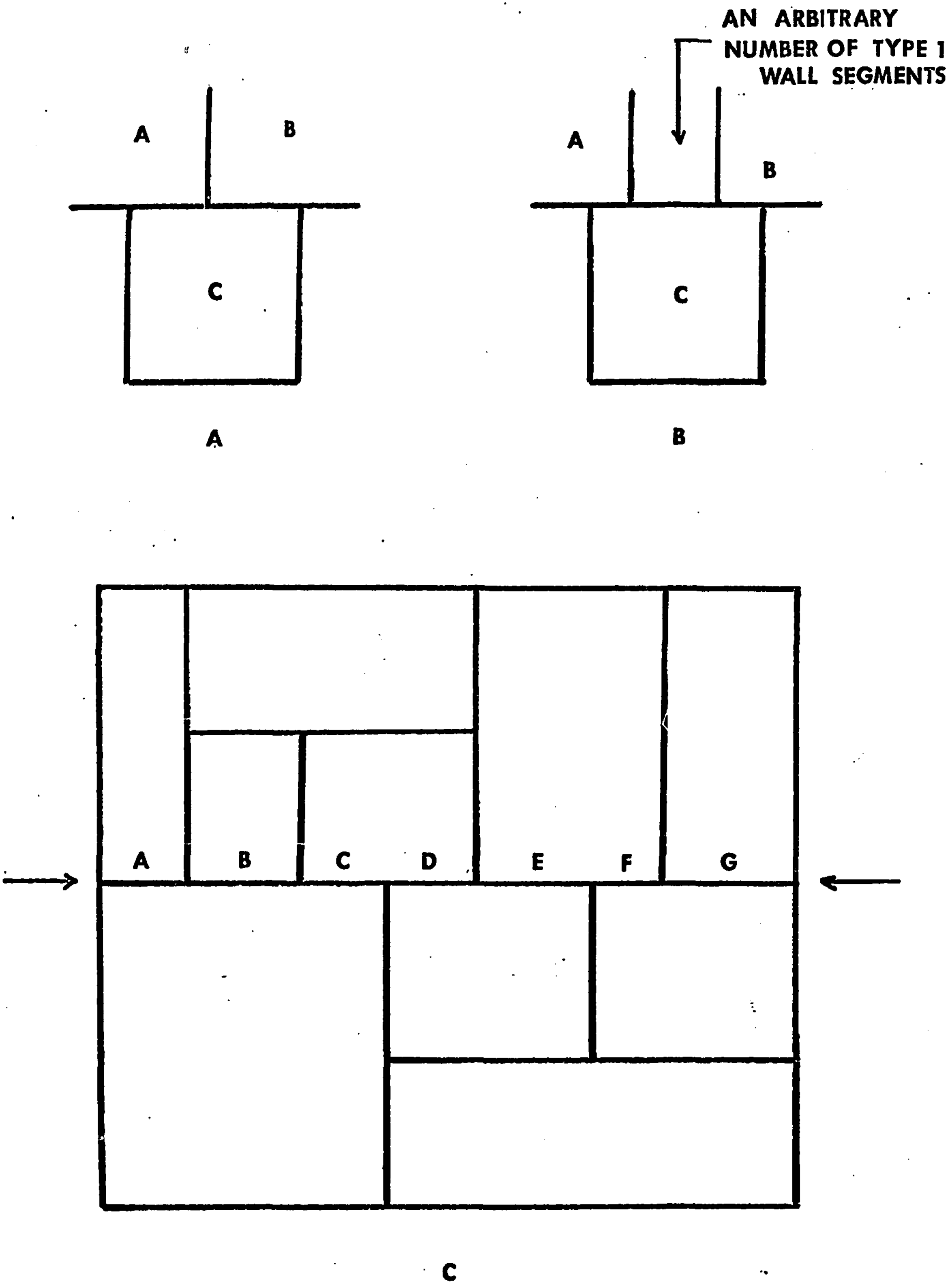
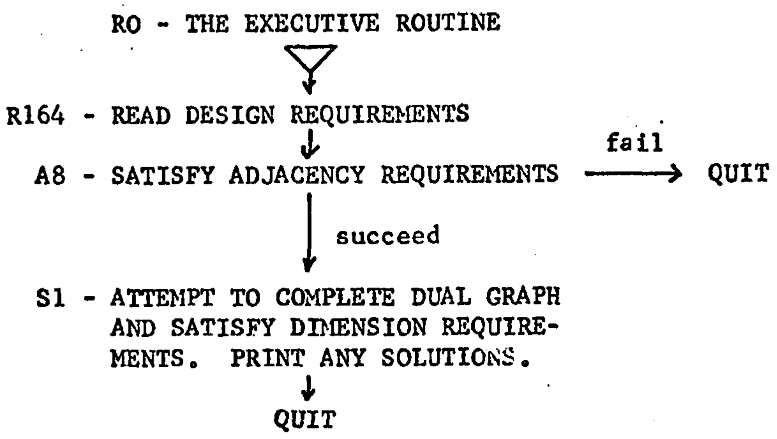


FIGURE A1-15

ILLUSTRATIONS FOR SECTION A.1.9

all cases have been covered, this proves that R86 can accomplish what it is supposed to.

CHART F1



RO - THE EXECUTIVE ROUTINE

R164 - READ DESIGN REQUIREMENTS

A8 - SATISFY ADJACENCY REQUIREMENTS

fail

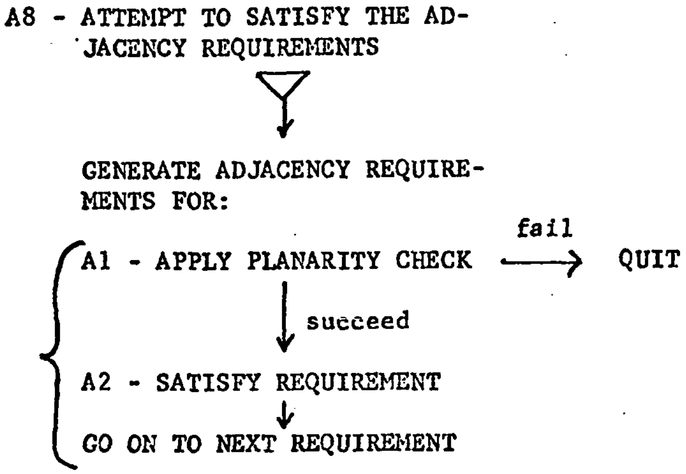
QUIT

succeed

S1 - ATTEMPT TO COMPLETE DUAL GRAPH AND SATISFY DIMENSION REQUIREMENTS. PRINT ANY SOLUTIONS.

QUIT

CHART F2



A8 - ATTEMPT TO SATISFY THE ADJACENCY REQUIREMENTS

GENERATE ADJACENCY REQUIREMENTS FOR:

A1 - APPLY PLANARITY CHECK

fail

QUIT

sueceed

A2 - SATISFY REQUIREMENT

GO ON TO NEXT REQUIREMENT

CHART F3

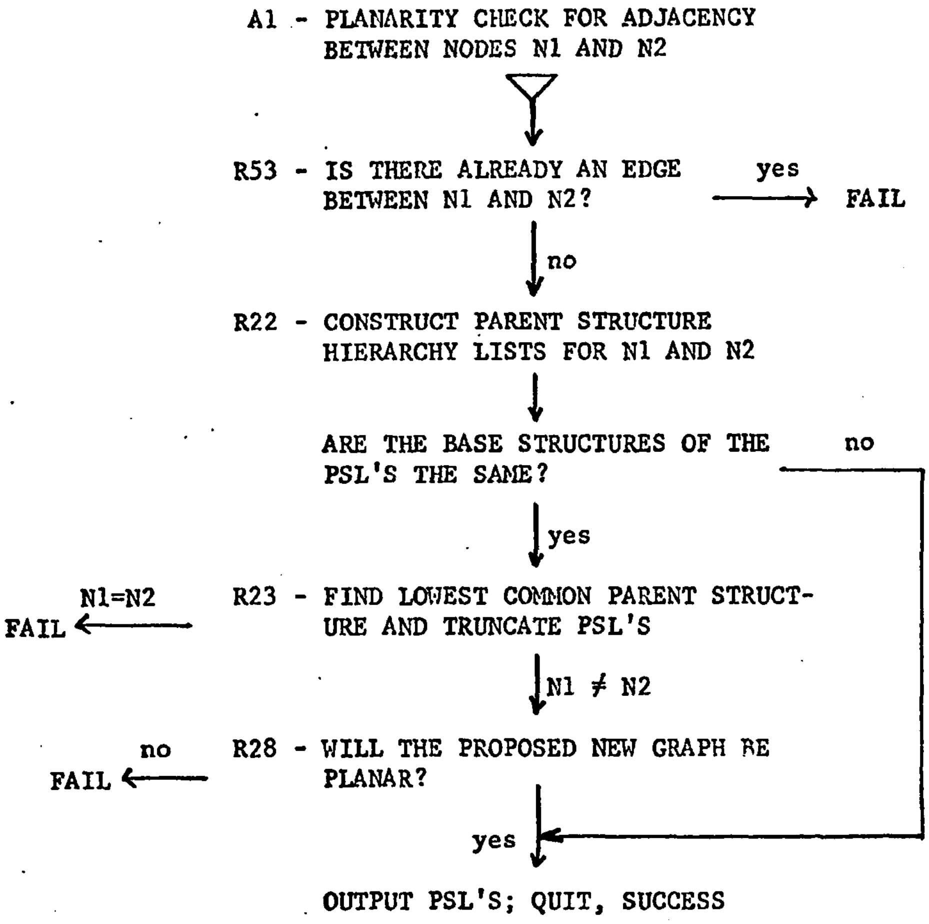

A1 - PLANARITY CHECK FOR ADJACENCY BETWEEN NODES N1 AND N2
R53 - IS THERE ALREADY AN EDGE BETWEEN N1 AND N2?
yes
FAIL
no
R22 - CONSTRUCT PARENT STRUCTURE HIERARCHY LISTS FOR N1 AND N2
ARE THE BASE STRUCTURES OF THE PSL'S THE SAME?
no
yes
N1=N2
FAIL
R23 - FIND LOWEST COMMON PARENT STRUCTURE AND TRUNCATE PSL'S
N1 ≠ N2
no
FAIL
R28 - WILL THE PROPOSED NEW GRAPH BE PLANAR?
yes
OUTPUT PSL'S; QUIT, SUCCESS

CHART F4

R22 - CONSTRUCT PARENT STRUCTURE HIERARCHY LIST (PSL) FOR NODE N

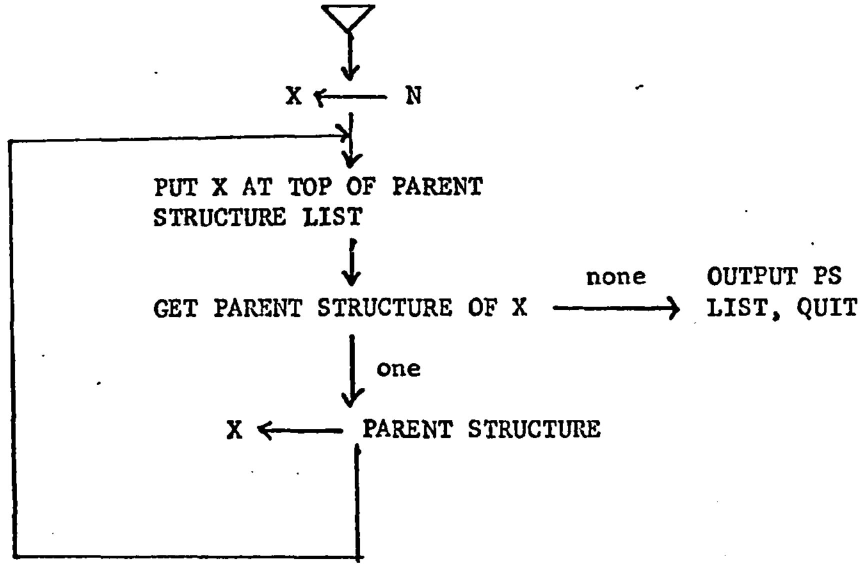

X ← N
PUT X AT TOP OF PARENT STRUCTURE LIST
GET PARENT STRUCTURE OF X
none
OUTPUT PS LIST, QUIT
one
X ← PARENT STRUCTURE

CHART F5

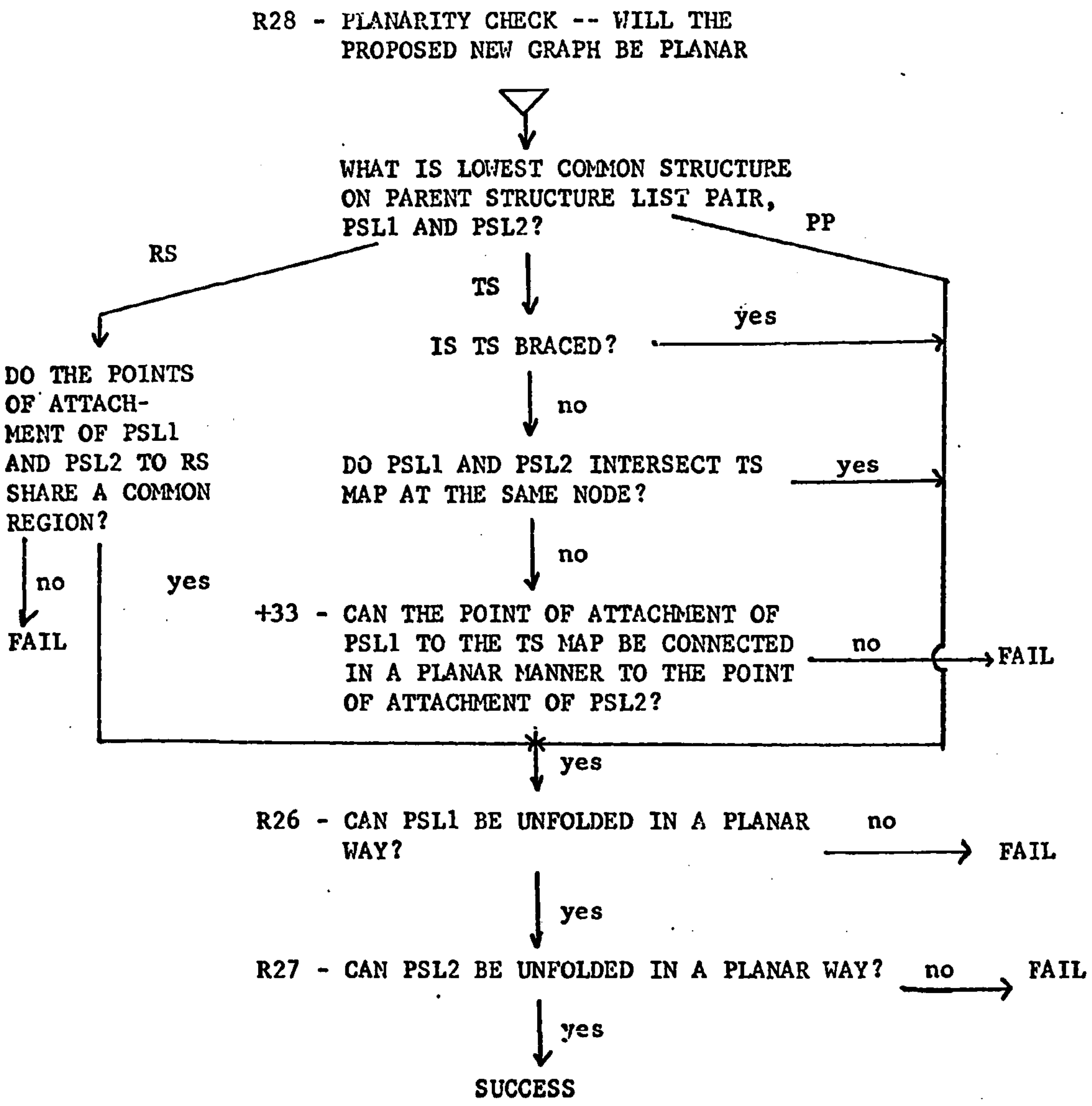

R28 - PLANARITY CHECK -- WILL THE PROPOSED NEW GRAPH BE PLANAR
WHAT IS LOWEST COMMON STRUCTURE ON PARENT STRUCTURE LIST PAIR, PSL1 AND PSL2?
RS
TS
PP
IS TS BRACED?
yes
no
DO THE POINTS OF ATTACHMENT OF PSL1 AND PSL2 TO RS SHARE A COMMON REGION?
no
FAIL
yes
DO PSL1 AND PSL2 INTERSECT TS MAP AT THE SAME NODE?
yes
no
+33 - CAN THE POINT OF ATTACHMENT OF PSL1 TO THE TS MAP BE CONNECTED IN A PLANAR MANNER TO THE POINT OF ATTACHMENT OF PSL2?
no
FAIL
yes
R26 - CAN PSL1 BE UNFOLDED IN A PLANAR WAY?
no
FAIL
yes
R27 - CAN PSL2 BE UNFOLDED IN A PLANAR WAY?
no
FAIL
yes
SUCCESS

CHART F6

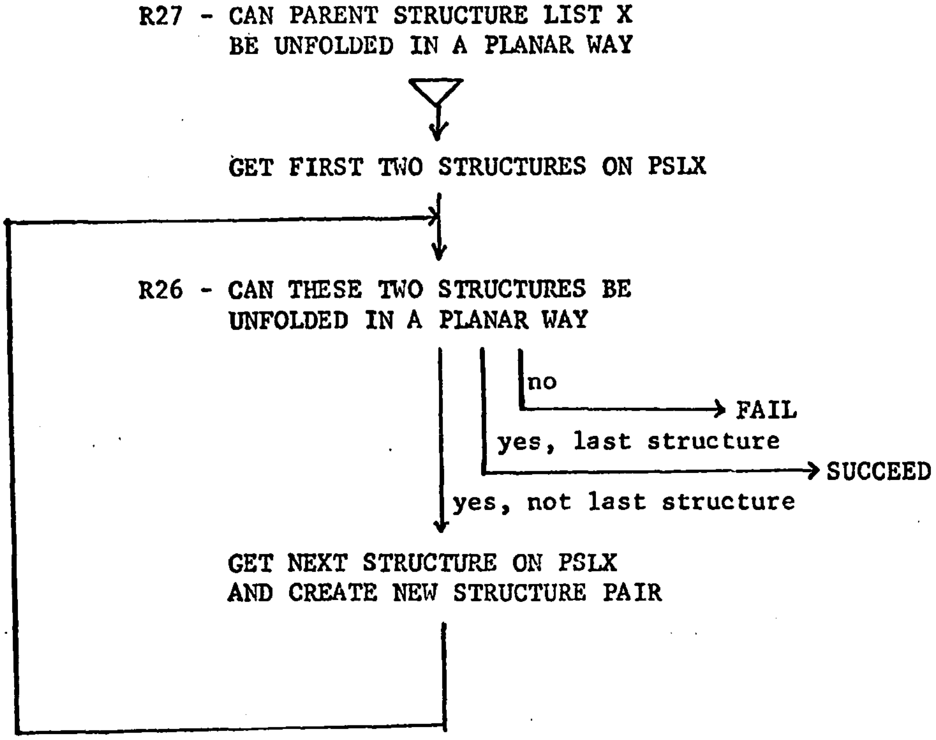

R27 - CAN PARENT STRUCTURE LIST X
BE UNFOLDED IN A PLANAR WAY
GET FIRST TWO STRUCTURES ON PSLX
R26 - CAN THESE TWO STRUCTURES BE
UNFOLDED IN A PLANAR WAY
no
FAIL
yes, last structure
SUCCEED
yes, not last structure
GET NEXT STRUCTURE ON PSLX
AND CREATE NEW STRUCTURE PAIR

CHART F7

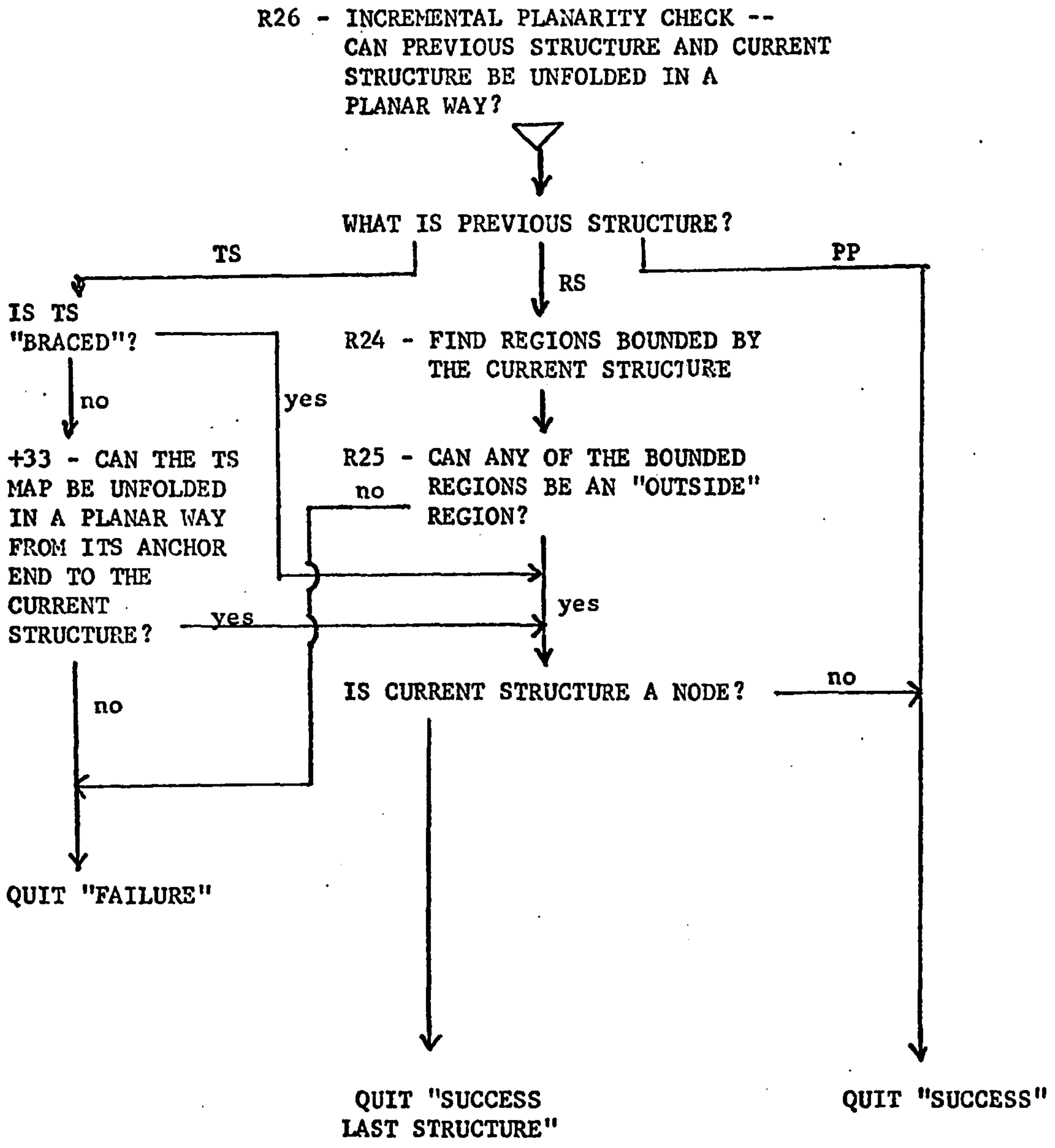
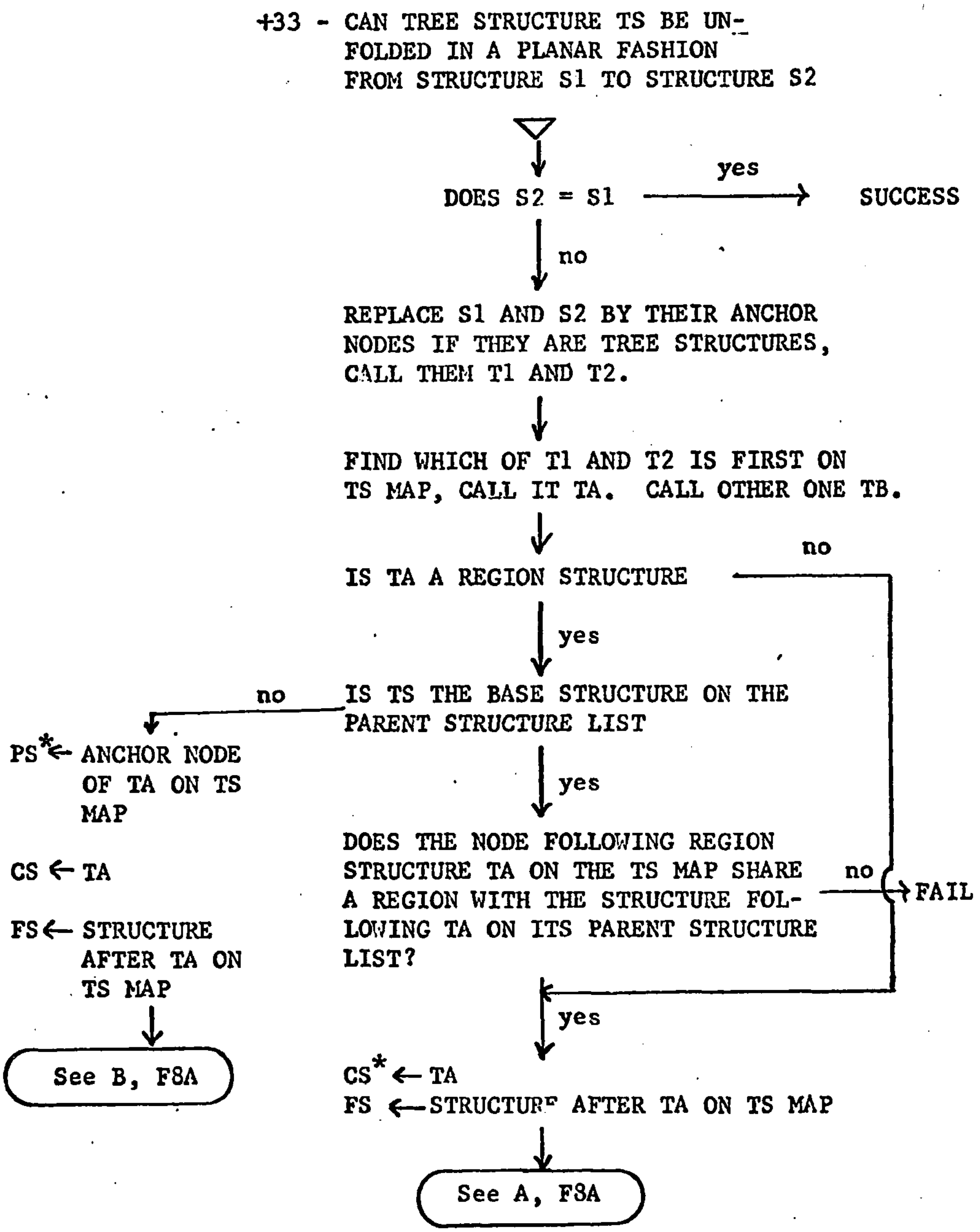

R26 - INCREMENTAL PLANARITY CHECK -- CAN PREVIOUS STRUCTURE AND CURRENT STRUCTURE BE UNFOLDED IN A PLANAR WAY?
WHAT IS PREVIOUS STRUCTURE?
TS
RS
PP
IS TS "BRACED"?
no
yes
R24 - FIND REGIONS BOUNDED BY THE CURRENT STRUCTURE
+33 - CAN THE TS MAP BE UNFOLDED IN A PLANAR WAY FROM ITS ANCHOR END TO THE CURRENT STRUCTURE?
R25 - CAN ANY OF THE BOUNDED REGIONS BE AN "OUTSIDE" REGION?
no
yes
yes
IS CURRENT STRUCTURE A NODE?
no
no
QUIT "FAILURE"
QUIT "SUCCESS LAST STRUCTURE"
QUIT "SUCCESS"

CHART F8


+33 - CAN TREE STRUCTURE TS BE UN-FOLDED IN A PLANAR FASHION FROM STRUCTURE S1 TO STRUCTURE S2
DOES S2 = S1
yes
SUCCESS
no
REPLACE S1 AND S2 BY THEIR ANCHOR NODES IF THEY ARE TREE STRUCTURES, CALL THEM T1 AND T2.
FIND WHICH OF T1 AND T2 IS FIRST ON TS MAP, CALL IT TA. CALL OTHER ONE TB.
IS TA A REGION STRUCTURE
no
yes
IS TS THE BASE STRUCTURE ON THE PARENT STRUCTURE LIST
no
PS*← ANCHOR NODE OF TA ON TS MAP
CS ← TA
FS← STRUCTURE AFTER TA ON TS MAP
See B, F8A
yes
DOES THE NODE FOLLOWING REGION STRUCTURE TA ON THE TS MAP SHARE A REGION WITH THE STRUCTURE FOL-LOWING TA ON ITS PARENT STRUCTURE LIST?
no
FAIL
yes
CS* ← TA
FS ←STRUCTURF AFTER TA ON TS MAP
See A, F8A

*READ PS AS "PREVIOUS STRUCTURE", CS AS "CURRENT STRUCTURE", AND FS AS "FUTURE STRUCTURE".

CHART F8A

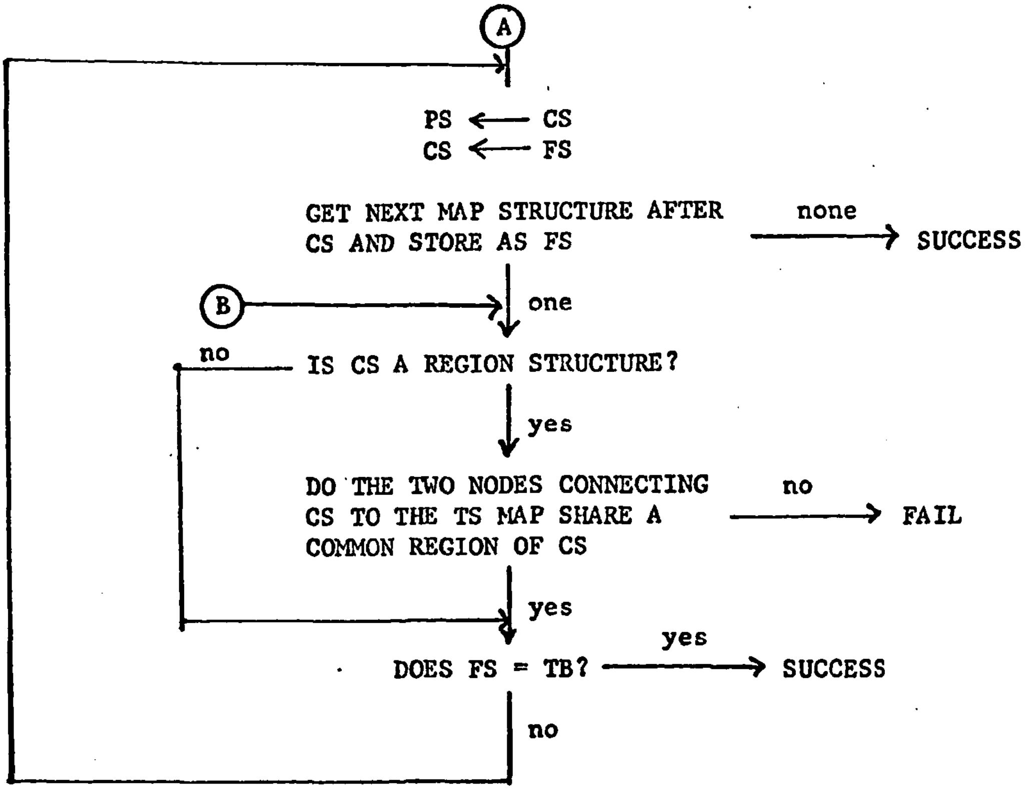

A
PS ← CS
CS ← FS
GET NEXT MAP STRUCTURE AFTER CS AND STORE AS FS
none
SUCCESS
B
one
no
IS CS A REGION STRUCTURE?
yes
DO THE TWO NODES CONNECTING CS TO THE TS MAP SHARE A COMMON REGION OF CS
no
FAIL
yes
yes
DOES FS = TB?
SUCCESS
no

CHART F9

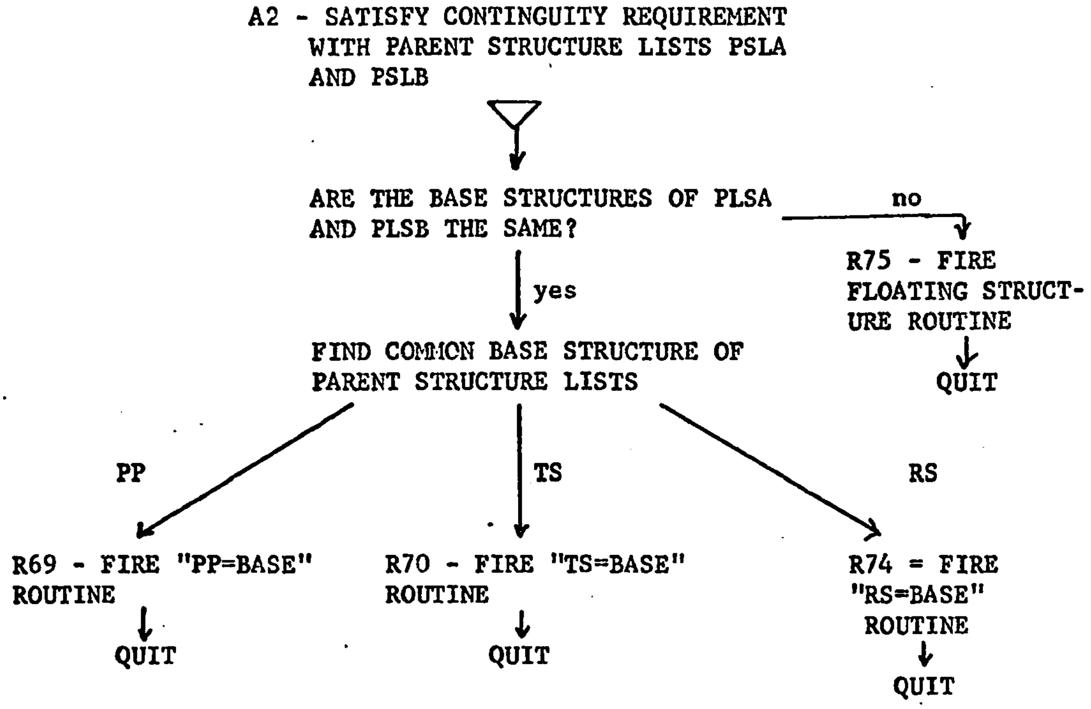
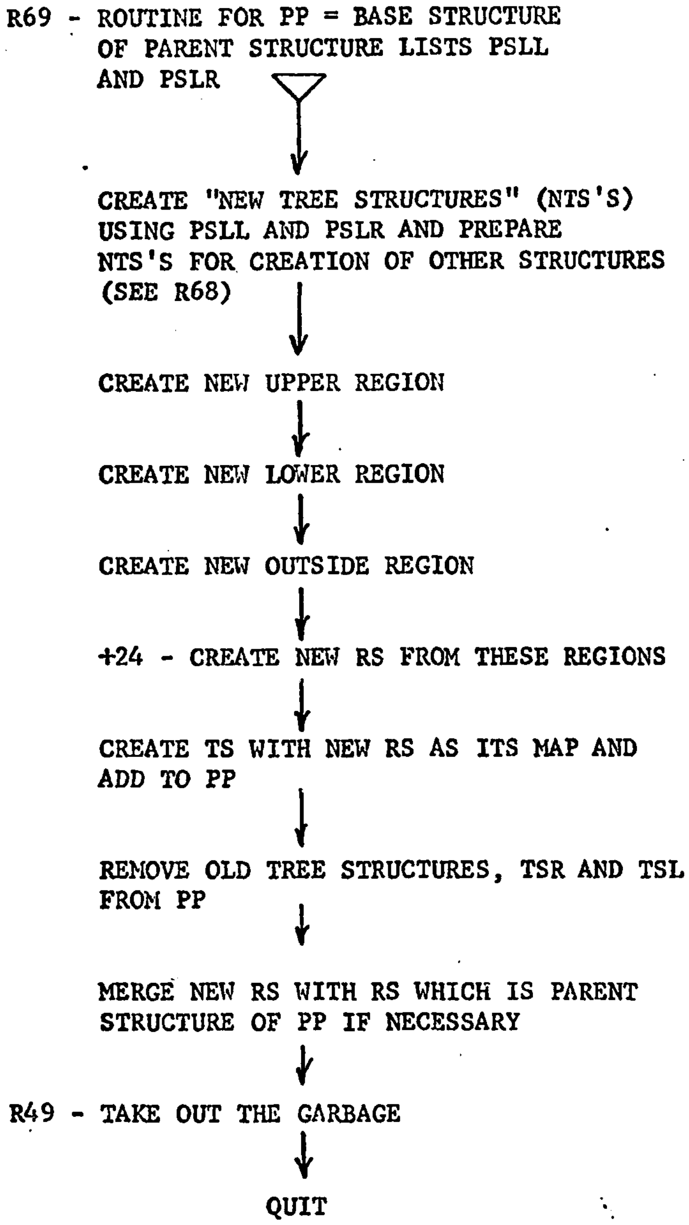

A2 - SATISFY CONTINGUITY REQUIREMENT WITH PARENT STRUCTURE LISTS PSLA AND PSLB
ARE THE BASE STRUCTURES OF PLSA AND PLSB THE SAME?
no
R75 - FIRE FLOATING STRUCTURE ROUTINE
QUIT
yes
FIND COMMON BASE STRUCTURE OF PARENT STRUCTURE LISTS
PP
TS
RS
R69 - FIRE "PP=BASE" ROUTINE
QUIT
R70 - FIRE "TS=BASE" ROUTINE
QUIT
R74 = FIRE "RS=BASE" ROUTINE
QUIT

CHART F10



R69 - ROUTINE FOR PP = BASE STRUCTURE OF PARENT STRUCTURE LISTS PSLL AND PSLR

CREATE "NEW TREE STRUCTURES" (NTS'S) USING PSLL AND PSLR AND PREPARE NTS'S FOR CREATION OF OTHER STRUCTURES (SEE R68)

CREATE NEW UPPER REGION

CREATE NEW LOWER REGION

CREATE NEW OUTSIDE REGION

+24 - CREATE NEW RS FROM THESE REGIONS

CREATE TS WITH NEW RS AS ITS MAP AND ADD TO PP

REMOVE OLD TREE STRUCTURES, TSR AND TSL FROM PP

MERGE NEW RS WITH RS WHICH IS PARENT STRUCTURE OF PP IF NECESSARY

R49 - TAKE OUT THE GARBAGE

QUIT

CHART F11

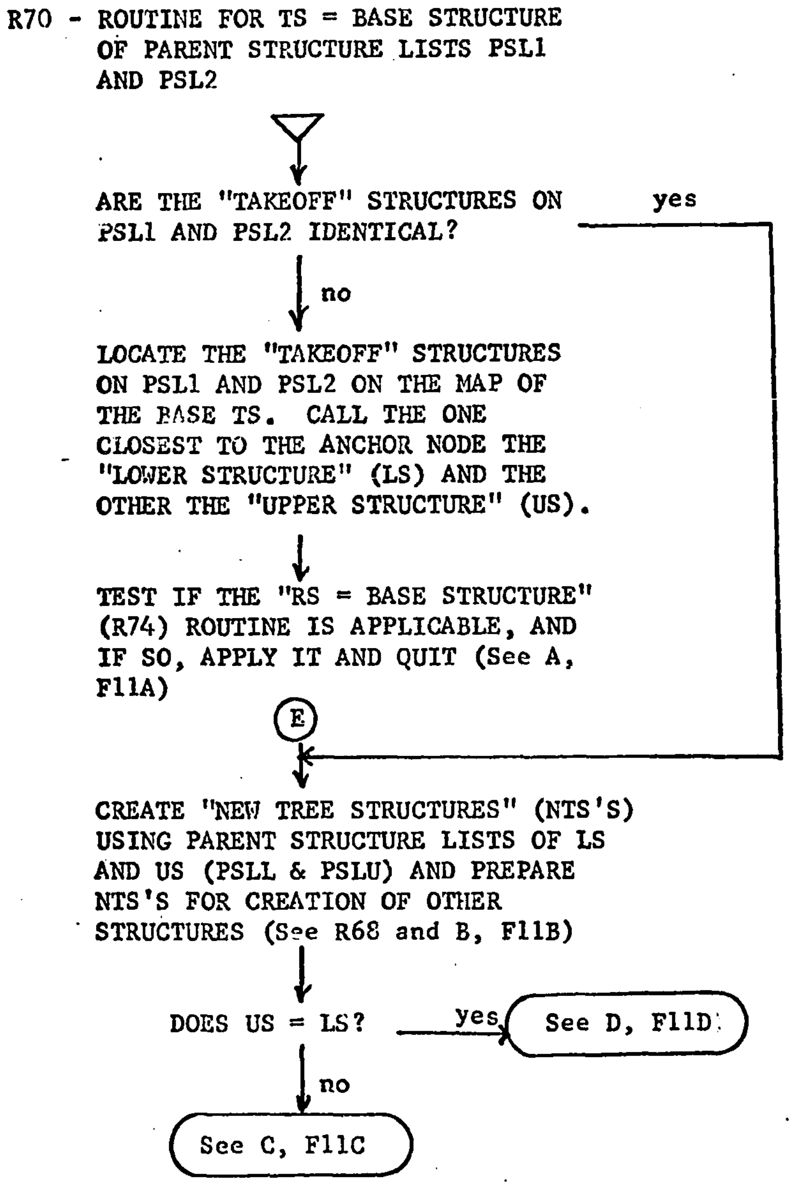

R70 - ROUTINE FOR TS = BASE STRUCTURE OF PARENT STRUCTURE LISTS PSL1 AND PSL2
ARE THE "TAKEOFF" STRUCTURES ON PSL1 AND PSL2 IDENTICAL?
yes
no
LOCATE THE "TAKEOFF" STRUCTURES ON PSL1 AND PSL2 ON THE MAP OF THE BASE TS. CALL THE ONE CLOSEST TO THE ANCHOR NODE THE "LOWER STRUCTURE" (LS) AND THE OTHER THE "UPPER STRUCTURE" (US).
TEST IF THE "RS = BASE STRUCTURE" (R74) ROUTINE IS APPLICABLE, AND IF SO, APPLY IT AND QUIT (See A, F11A)
E
CREATE "NEW TREE STRUCTURES" (NTS'S) USING PARENT STRUCTURE LISTS OF LS AND US (PSLL & PSLU) AND PREPARE NTS'S FOR CREATION OF OTHER STRUCTURES (See R68 and B, F11B)
DOES US = LS?
yes
See D, F11D
no
See C, F11C

CHART F11A

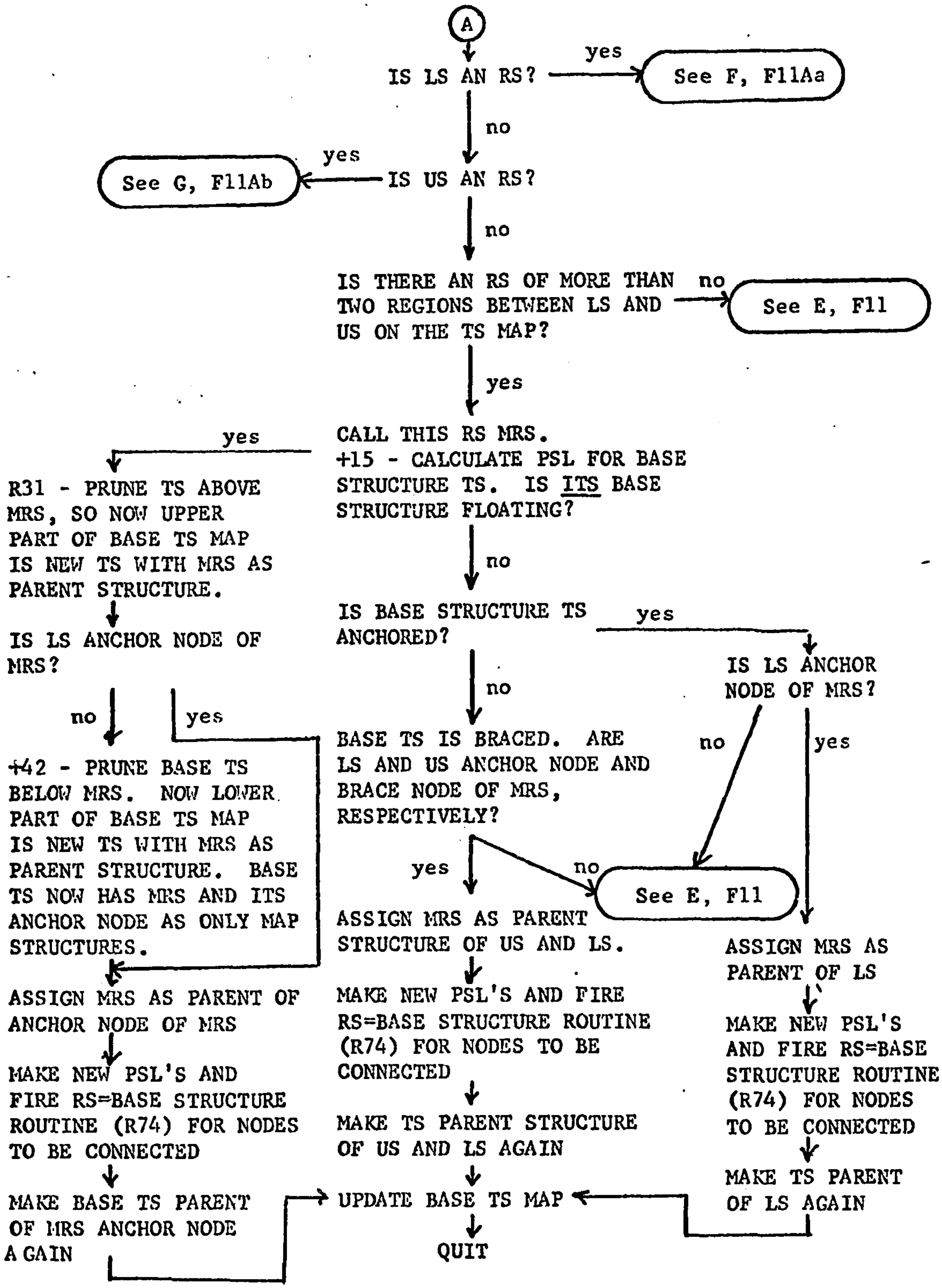

A
IS LS AN RS?
yes
See F, F11Aa
no
IS US AN RS?
yes
See G, F11Ab
no
IS THERE AN RS OF MORE THAN TWO REGIONS BETWEEN LS AND US ON THE TS MAP?
no
See E, F11
yes
CALL THIS RS MRS. +15 - CALCULATE PSL FOR BASE STRUCTURE TS. IS ITS BASE STRUCTURE FLOATING?
yes
R31 - PRUNE TS ABOVE MRS, SO NOW UPPER PART OF BASE TS MAP IS NEW TS WITH MRS AS PARENT STRUCTURE.
IS LS ANCHOR NODE OF MRS?
no
yes
+42 - PRUNE BASE TS BELOW MRS. NOW LOWER PART OF BASE TS MAP IS NEW TS WITH MRS AS PARENT STRUCTURE. BASE TS NOW HAS MRS AND ITS ANCHOR NODE AS ONLY MAP STRUCTURES.
ASSIGN MRS AS PARENT OF ANCHOR NODE OF MRS
MAKE NEW PSL'S AND FIRE RS=BASE STRUCTURE ROUTINE (R74) FOR NODES TO BE CONNECTED
MAKE BASE TS PARENT OF MRS ANCHOR NODE AGAIN
no
IS BASE STRUCTURE TS ANCHORED?
yes
IS LS ANCHOR NODE OF MRS?
no
yes
no
BASE TS IS BRACED. ARE LS AND US ANCHOR NODE AND BRACE NODE OF MRS, RESPECTIVELY?
yes
no
See E, F11
ASSIGN MRS AS PARENT STRUCTURE OF US AND LS.
MAKE NEW PSL'S AND FIRE RS=BASE STRUCTURE ROUTINE (R74) FOR NODES TO BE CONNECTED
MAKE TS PARENT STRUCTURE OF US AND LS AGAIN
UPDATE BASE TS MAP
QUIT
ASSIGN MRS AS PARENT OF LS
MAKE NEW PSL'S AND FIRE RS=BASE STRUCTURE ROUTINE (R74) FOR NODES TO BE CONNECTED
MAKE TS PARENT OF LS AGAIN

CHART F11Aa

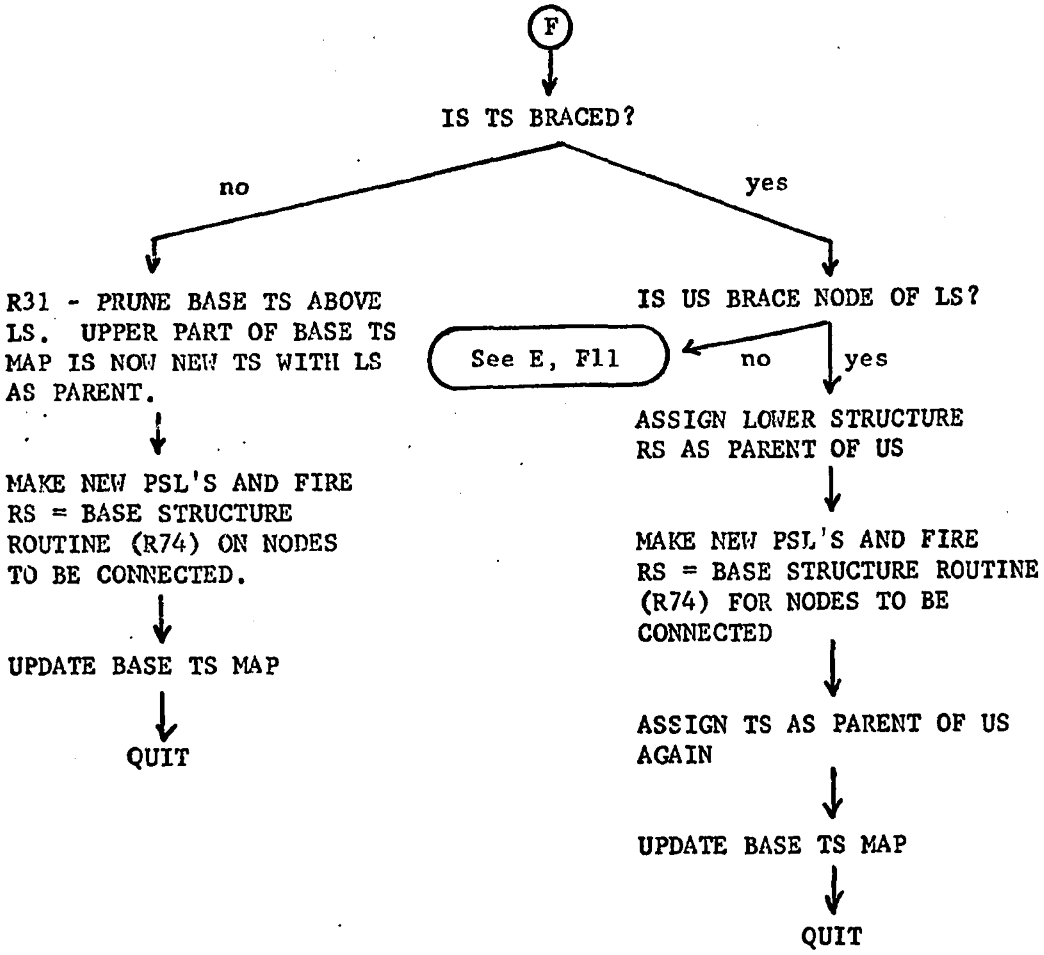

F
IS TS BRACED?
no
yes
R31 - PRUNE BASE TS ABOVE LS. UPPER PART OF BASE TS MAP IS NOW NEW TS WITH LS AS PARENT.
MAKE NEW PSL'S AND FIRE RS = BASE STRUCTURE ROUTINE (R74) ON NODES TO BE CONNECTED.
UPDATE BASE TS MAP
QUIT
IS US BRACE NODE OF LS?
See E, F11
no
yes
ASSIGN LOWER STRUCTURE RS AS PARENT OF US
MAKE NEW PSL'S AND FIRE RS = BASE STRUCTURE ROUTINE (R74) FOR NODES TO BE CONNECTED
ASSIGN TS AS PARENT OF US AGAIN
UPDATE BASE TS MAP
QUIT

CHART F11Ab

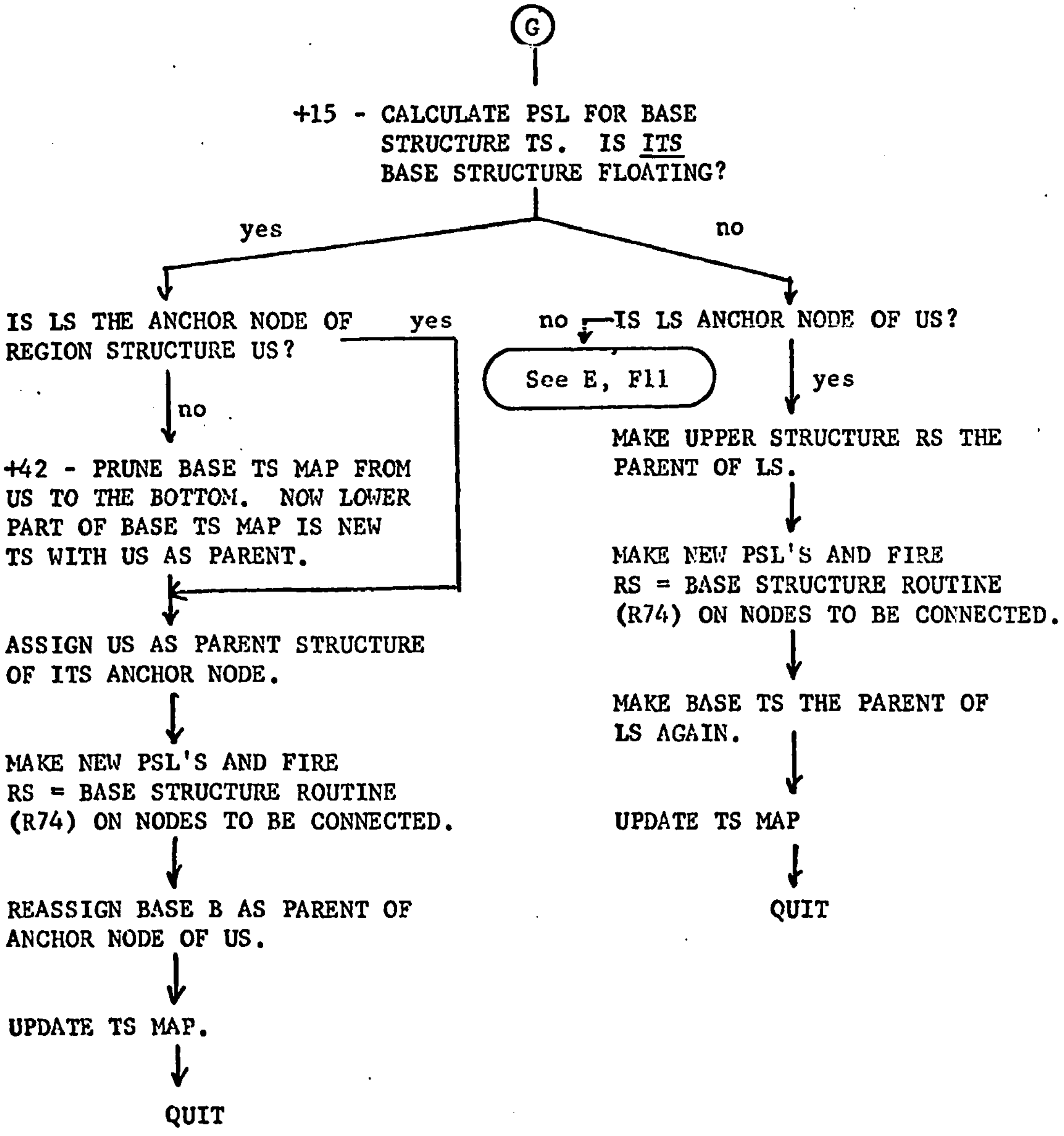
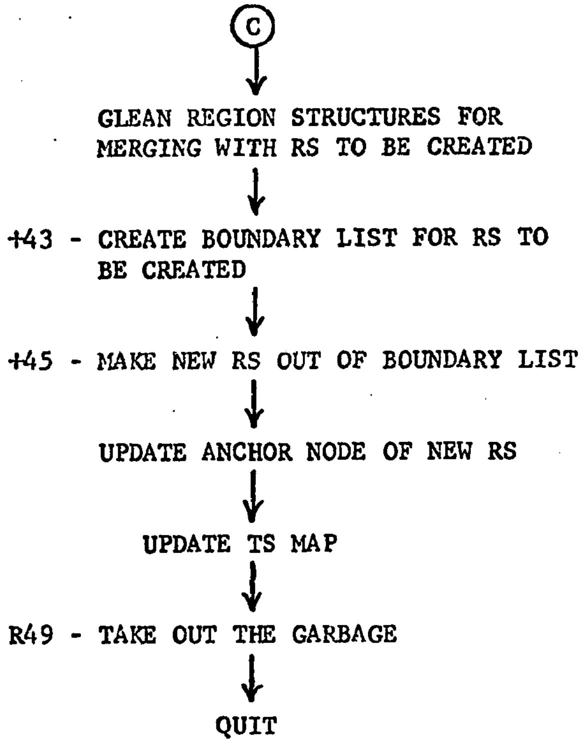

G
+15 - CALCULATE PSL FOR BASE STRUCTURE TS. IS ITS BASE STRUCTURE FLOATING?
yes
no
IS LS THE ANCHOR NODE OF REGION STRUCTURE US?
yes
no
+42 - PRUNE BASE TS MAP FROM US TO THE BOTTOM. NOW LOWER PART OF BASE TS MAP IS NEW TS WITH US AS PARENT.
ASSIGN US AS PARENT STRUCTURE OF ITS ANCHOR NODE.
MAKE NEW PSL'S AND FIRE RS = BASE STRUCTURE ROUTINE (R74) ON NODES TO BE CONNECTED.
REASSIGN BASE B AS PARENT OF ANCHOR NODE OF US.
UPDATE TS MAP.
QUIT
no
IS LS ANCHOR NODE OF US?
See E, F11
yes
MAKE UPPER STRUCTURE RS THE PARENT OF LS.
MAKE NEW PSL'S AND FIRE RS = BASE STRUCTURE ROUTINE (R74) ON NODES TO BE CONNECTED.
MAKE BASE TS THE PARENT OF LS AGAIN.
UPDATE TS MAP
QUIT

CHART F11C



GLEAN REGION STRUCTURES FOR MERGING WITH RS TO BE CREATED

+43 - CREATE BOUNDARY LIST FOR RS TO BE CREATED

+45 - MAKE NEW RS OUT OF BOUNDARY LIST

UPDATE ANCHOR NODE OF NEW RS

UPDATE TS MAP

R49 - TAKE OUT THE GARBAGE

QUIT

CHART F11D

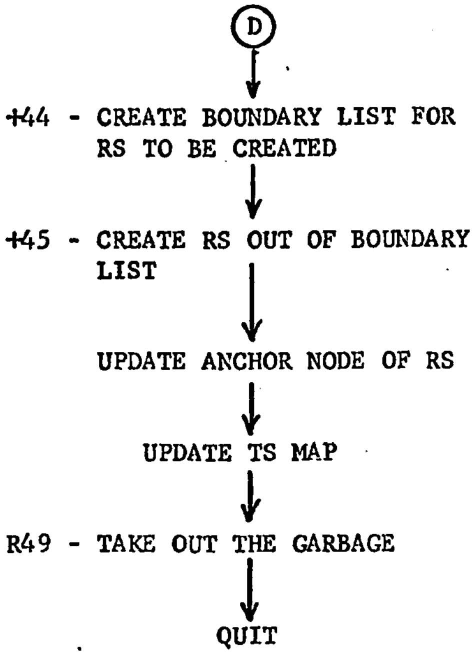

D
+44 - CREATE BOUNDARY LIST FOR RS TO BE CREATED
+45 - CREATE RS OUT OF BOUNDARY LIST
UPDATE ANCHOR NODE OF RS
UPDATE TS MAP
R49 - TAKE OUT THE GARBAGE
QUIT

CHART F12

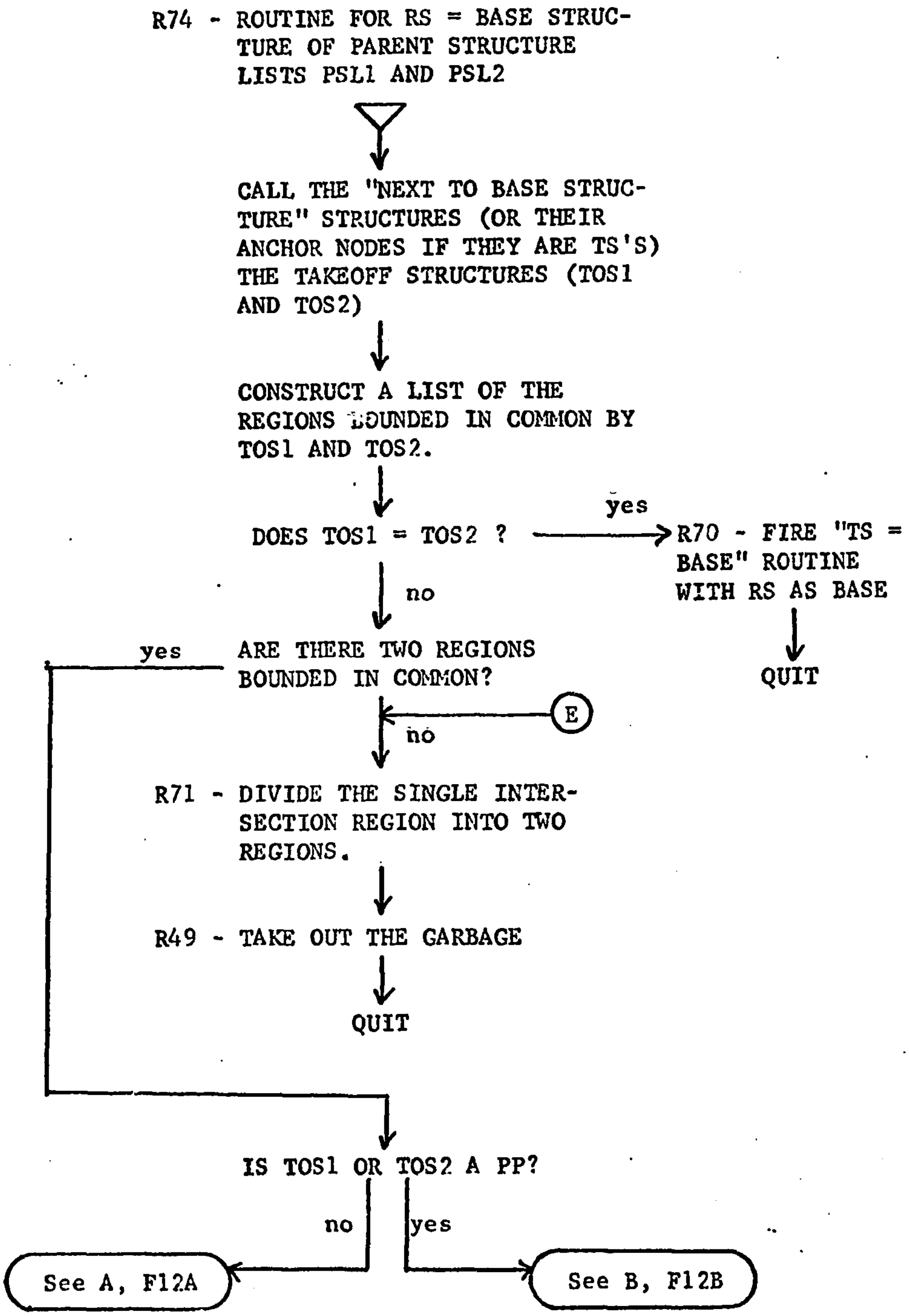

R74 - ROUTINE FOR RS = BASE STRUCTURE OF PARENT STRUCTURE LISTS PSL1 AND PSL2
CALL THE "NEXT TO BASE STRUCTURE" STRUCTURES (OR THEIR ANCHOR NODES IF THEY ARE TS'S) THE TAKEOFF STRUCTURES (TOS1 AND TOS2)
CONSTRUCT A LIST OF THE REGIONS BOUNDED IN COMMON BY TOS1 AND TOS2.
DOES TOS1 = TOS2 ?
yes
R70 - FIRE "TS = BASE" ROUTINE WITH RS AS BASE
QUIT
no
yes
ARE THERE TWO REGIONS BOUNDED IN COMMON?
E
no
R71 - DIVIDE THE SINGLE INTERSECTION REGION INTO TWO REGIONS.
R49 - TAKE OUT THE GARBAGE
QUIT
IS TOS1 OR TOS2 A PP?
no
yes
See A, F12A
See B, F12B

CHART F12A

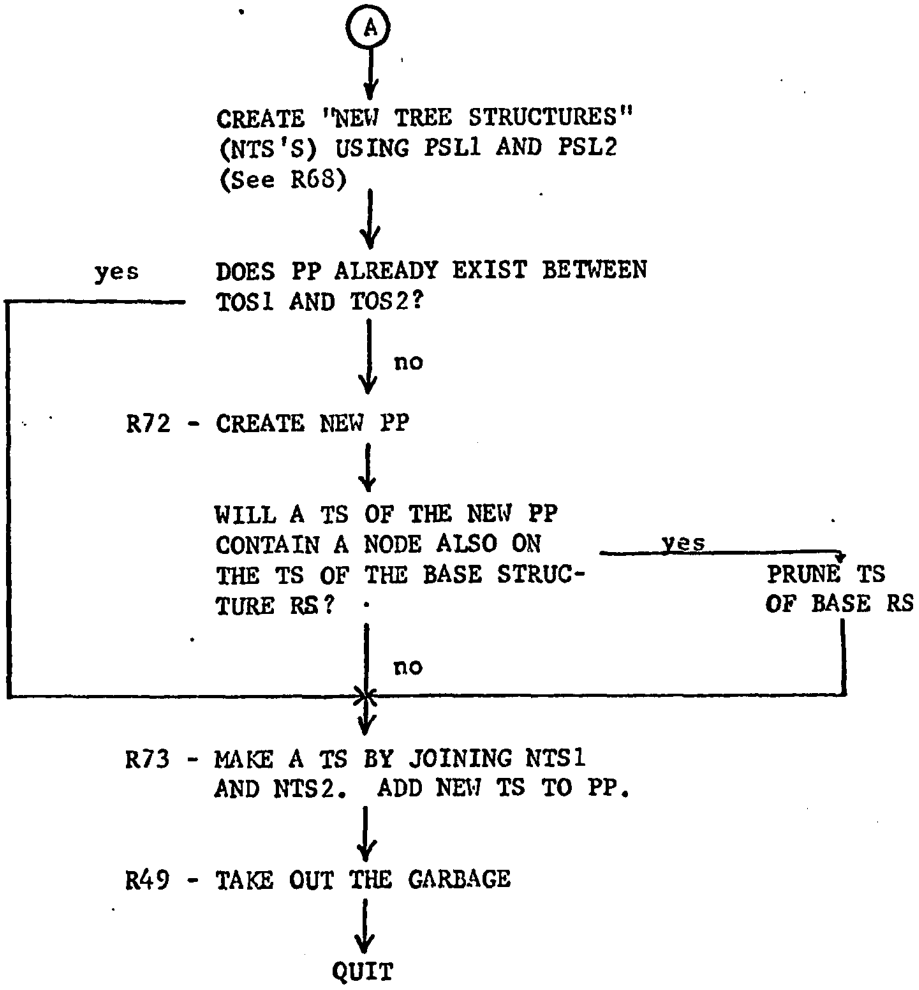
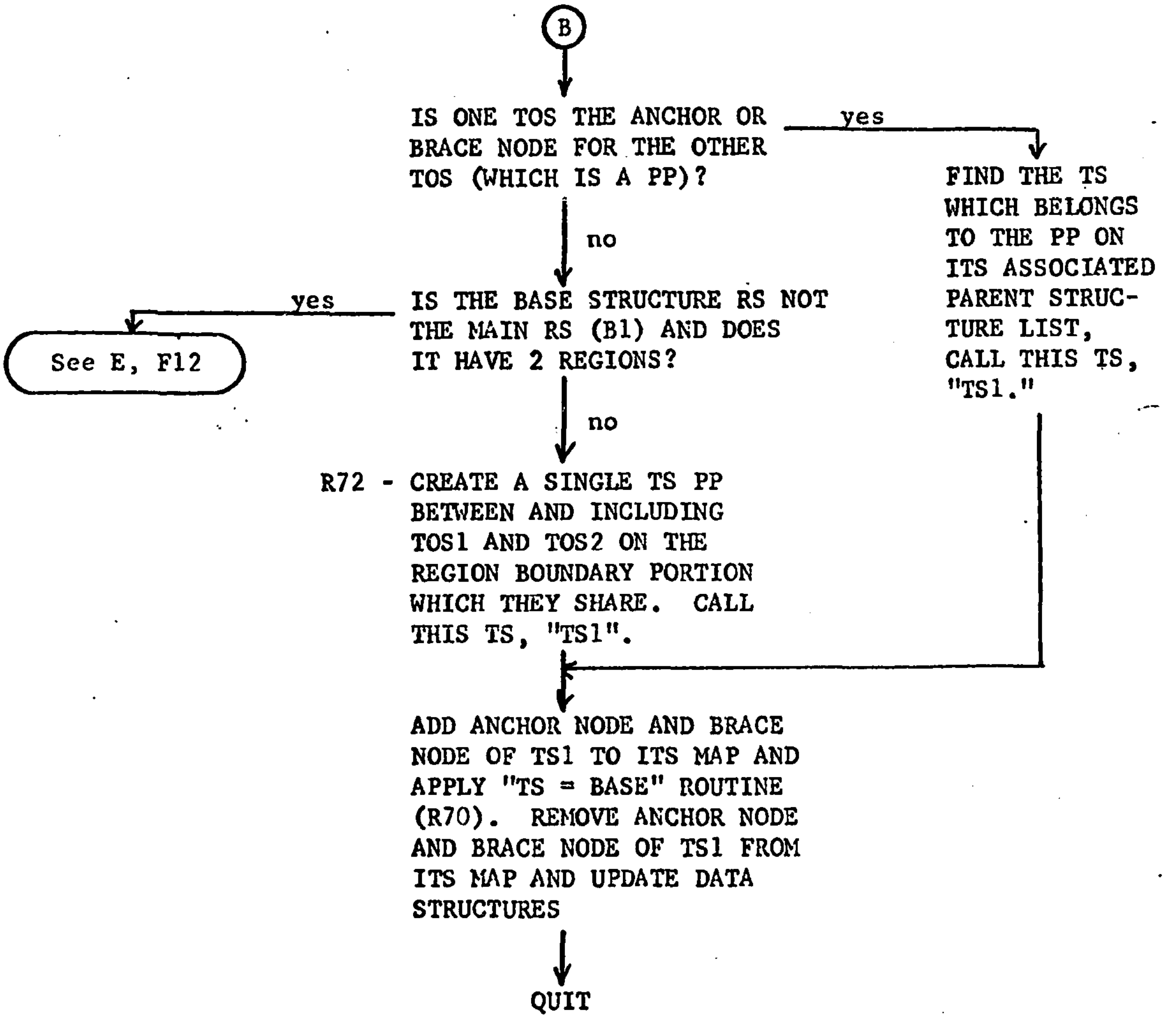

A
CREATE 'NEW TREE STRUCTURES" (NTS'S) USING PSL1 AND PSL2 (See R68)
yes
DOES PP ALREADY EXIST BETWEEN TOS1 AND TOS2?
no
R72 - CREATE NEW PP
WILL A TS OF THE NEW PP CONTAIN A NODE ALSO ON THE TS OF THE BASE STRUCTURE RS?
yes
PRUNE TS OF BASE RS
no
R73 - MAKE A TS BY JOINING NTS1 AND NTS2. ADD NEW TS TO PP.
R49 - TAKE OUT THE GARBAGE
QUIT

CHART F12B



B

IS ONE TOS THE ANCHOR OR BRACE NODE FOR THE OTHER TOS (WHICH IS A PP)?

yes

FIND THE TS WHICH BELONGS TO THE PP ON ITS ASSOCIATED PARENT STRUCTURE LIST, CALL THIS TS, "TS1."

no

IS THE BASE STRUCTURE RS NOT THE MAIN RS (B1) AND DOES IT HAVE 2 REGIONS?

yes

See E, F12

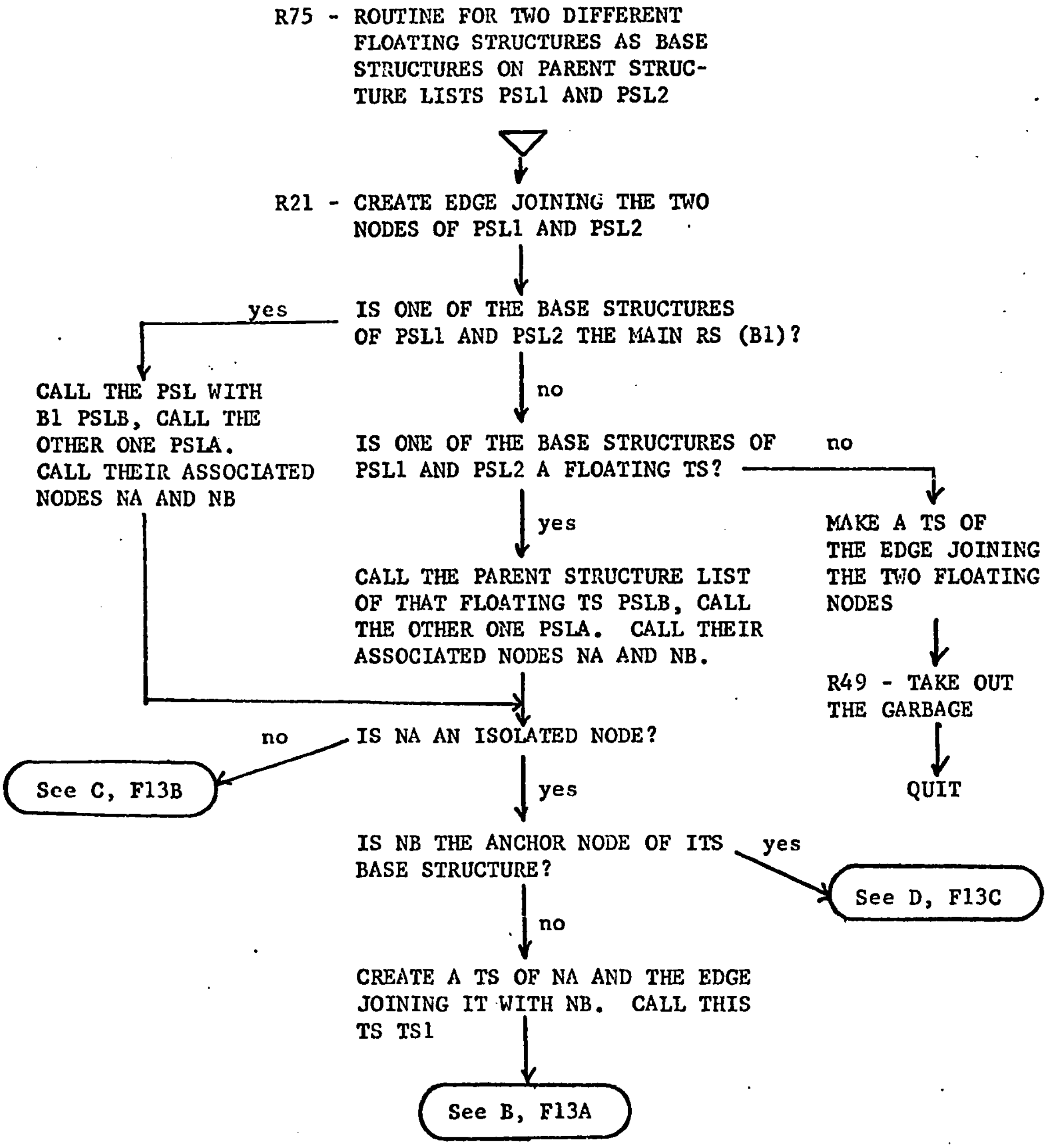
no

R72 - CREATE A SINGLE TS PP BETWEEN AND INCLUDING TOS1 AND TOS2 ON THE REGION BOUNDARY PORTION WHICH THEY SHARE. CALL THIS TS, "TS1".

ADD ANCHOR NODE AND BRACE NODE OF TS1 TO ITS MAP AND APPLY "TS = BASE" ROUTINE (R70). REMOVE ANCHOR NODE AND BRACE NODE OF TS1 FROM ITS MAP AND UPDATE DATA STRUCTURES

QUIT

CHART F13



R75 - ROUTINE FOR TWO DIFFERENT FLOATING STRUCTURES AS BASE STRUCTURES ON PARENT STRUCTURE LISTS PSL1 AND PSL2

R21 - CREATE EDGE JOINING THE TWO NODES OF PSL1 AND PSL2

IS ONE OF THE BASE STRUCTURES OF PSL1 AND PSL2 THE MAIN RS (B1)?

yes: CALL THE PSL WITH B1 PSLB, CALL THE OTHER ONE PSLA. CALL THEIR ASSOCIATED NODES NA AND NB

no: IS ONE OF THE BASE STRUCTURES OF PSL1 AND PSL2 A FLOATING TS?

no: MAKE A TS OF THE EDGE JOINING THE TWO FLOATING NODES

R49 - TAKE OUT THE GARBAGE

QUIT

yes: CALL THE PARENT STRUCTURE LIST OF THAT FLOATING TS PSLB, CALL THE OTHER ONE PSLA. CALL THEIR ASSOCIATED NODES NA AND NB.

IS NA AN ISOLATED NODE?

no: See C, F13B

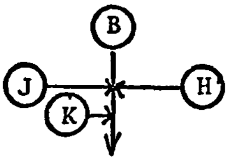
yes: IS NB THE ANCHOR NODE OF ITS BASE STRUCTURE?

yes: See D, F13C

no: CREATE A TS OF NA AND THE EDGE JOINING IT WITH NB. CALL THIS TS TS1

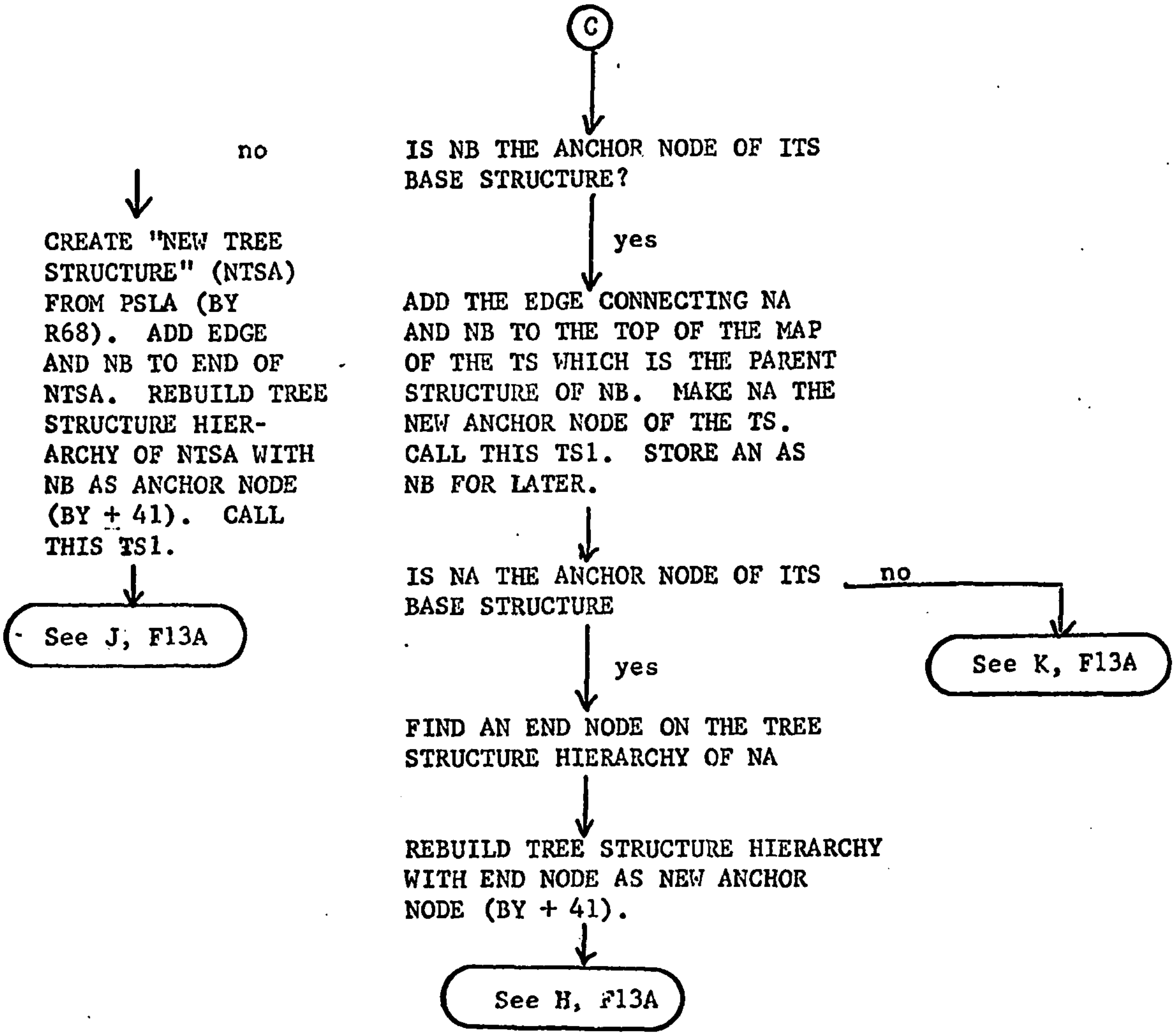
See B, F13A

CHART F13A



ADJOIN TS1 TO THE PARENT STRUCTURE OF NB. UPDATE DATA STRUCTURES

CHART F13B



IS NB THE ANCHOR NODE OF ITS BASE STRUCTURE?

ADD THE EDGE CONNECTING NA AND NB TO THE TOP OF THE MAP OF THE TS WHICH IS THE PARENT STRUCTURE OF NB. MAKE NA THE NEW ANCHOR NODE OF THE TS. CALL THIS TS1. STORE AN AS NB FOR LATER.

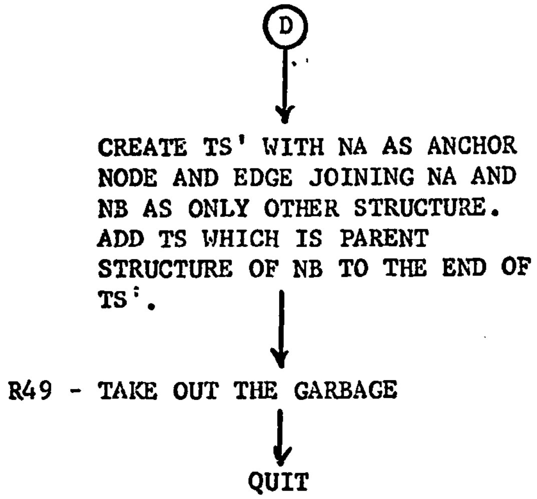
CREATE "NEW TREE STRUCTURE" (NTSA) FROM PSLA (BY R68). ADD EDGE AND NB TO END OF NTSA. REBUILD TREE STRUCTURE HIERARCHY OF NTSA WITH NB AS ANCHOR NODE (BY + 41). CALL THIS TS1.

IS NA THE ANCHOR NODE OF ITS BASE STRUCTURE

FIND AN END NODE ON THE TREE STRUCTURE HIERARCHY OF NA

REBUILD TREE STRUCTURE HIERARCHY WITH END NODE AS NEW ANCHOR NODE (BY + 41).

CHART F13C



D

CREATE TS' WITH NA AS ANCHOR NODE AND EDGE JOINING NA AND NB AS ONLY OTHER STRUCTURE. ADD TS WHICH IS PARENT STRUCTURE OF NB TO THE END OF TS'.

R49 - TAKE OUT THE GARBAGE

QUIT

CHART F14

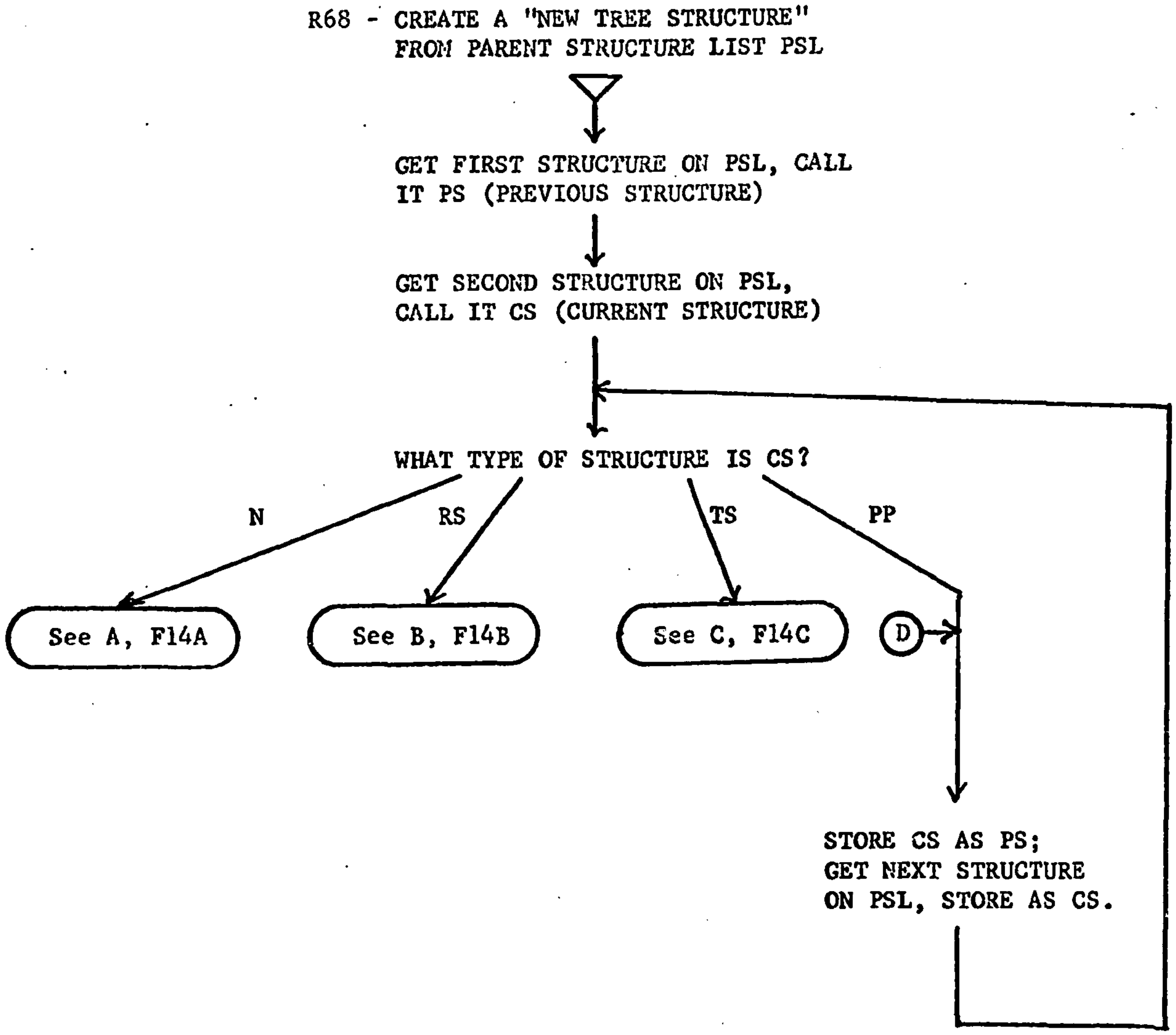

R68 - CREATE A "NEW TREE STRUCTURE"
FROM PARENT STRUCTURE LIST PSL
GET FIRST STRUCTURE ON PSL, CALL
IT PS (PREVIOUS STRUCTURE)
GET SECOND STRUCTURE ON PSL,
CALL IT CS (CURRENT STRUCTURE)
WHAT TYPE OF STRUCTURE IS CS?
N
RS
TS
PP
See A, F14A
See B, F14B
See C, F14C
D
STORE CS AS PS;
GET NEXT STRUCTURE
ON PSL, STORE AS CS.

CHART F14A

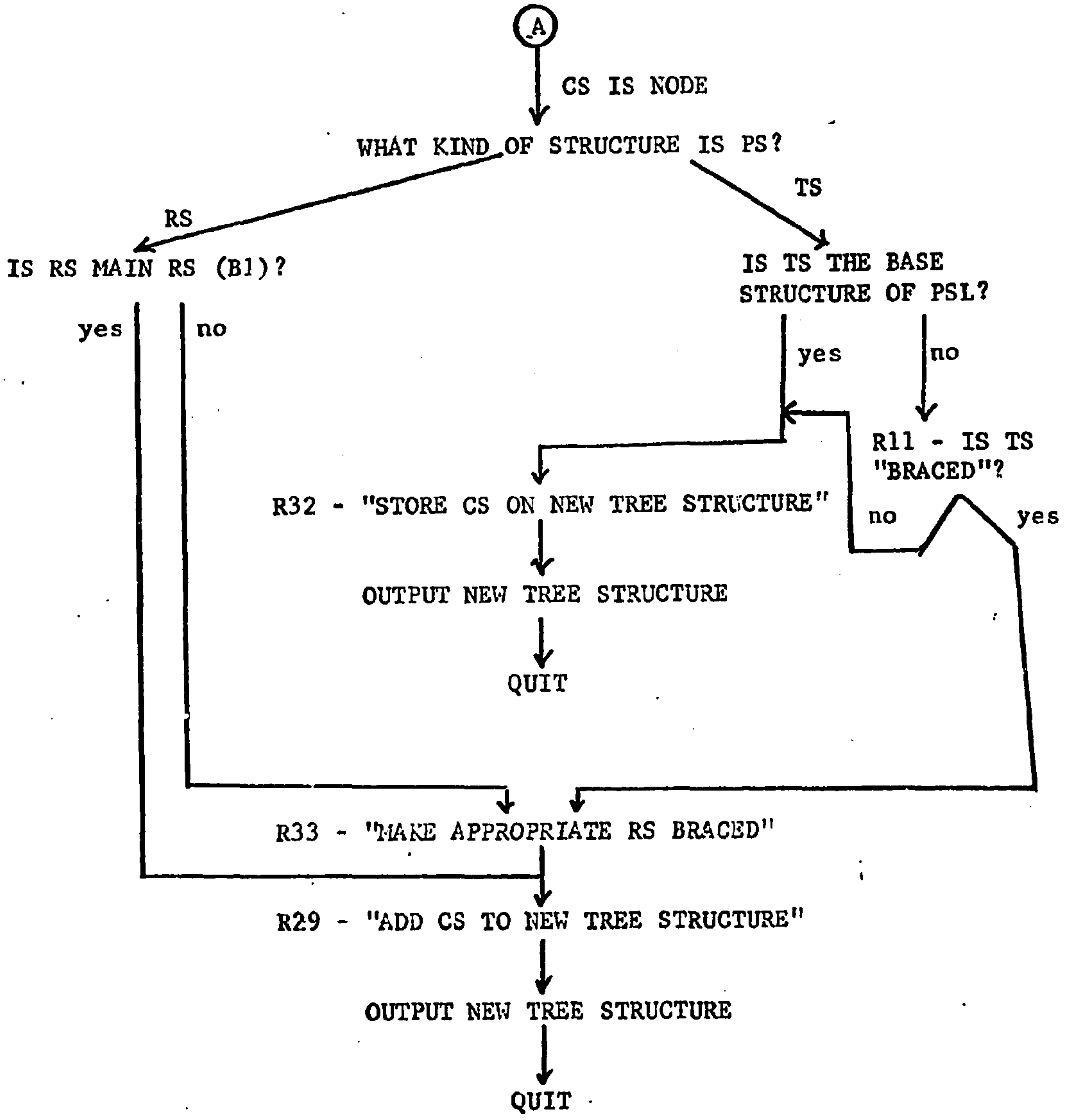

A
CS IS NODE
WHAT KIND OF STRUCTURE IS PS?
RS
TS
IS RS MAIN RS (B1)?
yes
no
IS TS THE BASE STRUCTURE OF PSL?
yes
no
R11 - IS TS "BRACED"?
no
yes
R32 - "STORE CS ON NEW TREE STRUCTURE"
OUTPUT NEW TREE STRUCTURE
QUIT
R33 - "MAKE APPROPRIATE RS BRACED"
R29 - "ADD CS TO NEW TREE STRUCTURE"
OUTPUT NEW TREE STRUCTURE
QUIT

CHART F14B

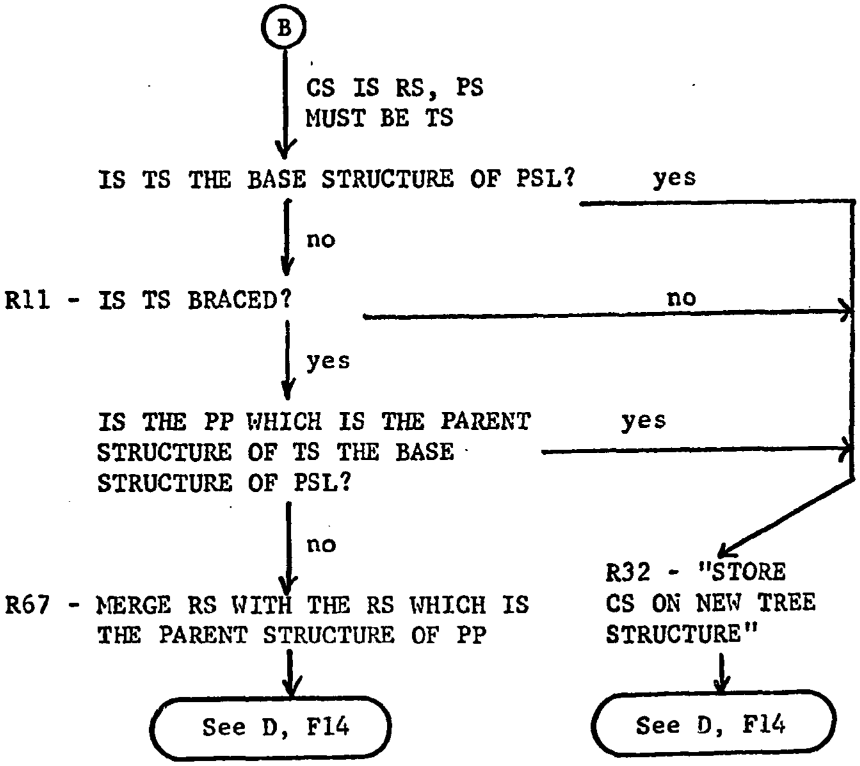

B
CS IS RS, PS MUST BE TS
IS TS THE BASE STRUCTURE OF PSL?
yes
no
R11 - IS TS BRACED?
no
yes
IS THE PP WHICH IS THE PARENT STRUCTURE OF TS THE BASE STRUCTURE OF PSL?
yes
no
R67 - MERGE RS WITH THE RS WHICH IS THE PARENT STRUCTURE OF PP
R32 - "STORE CS ON NEW TREE STRUCTURE"
See D, F14
See D, F14

CHART F14C

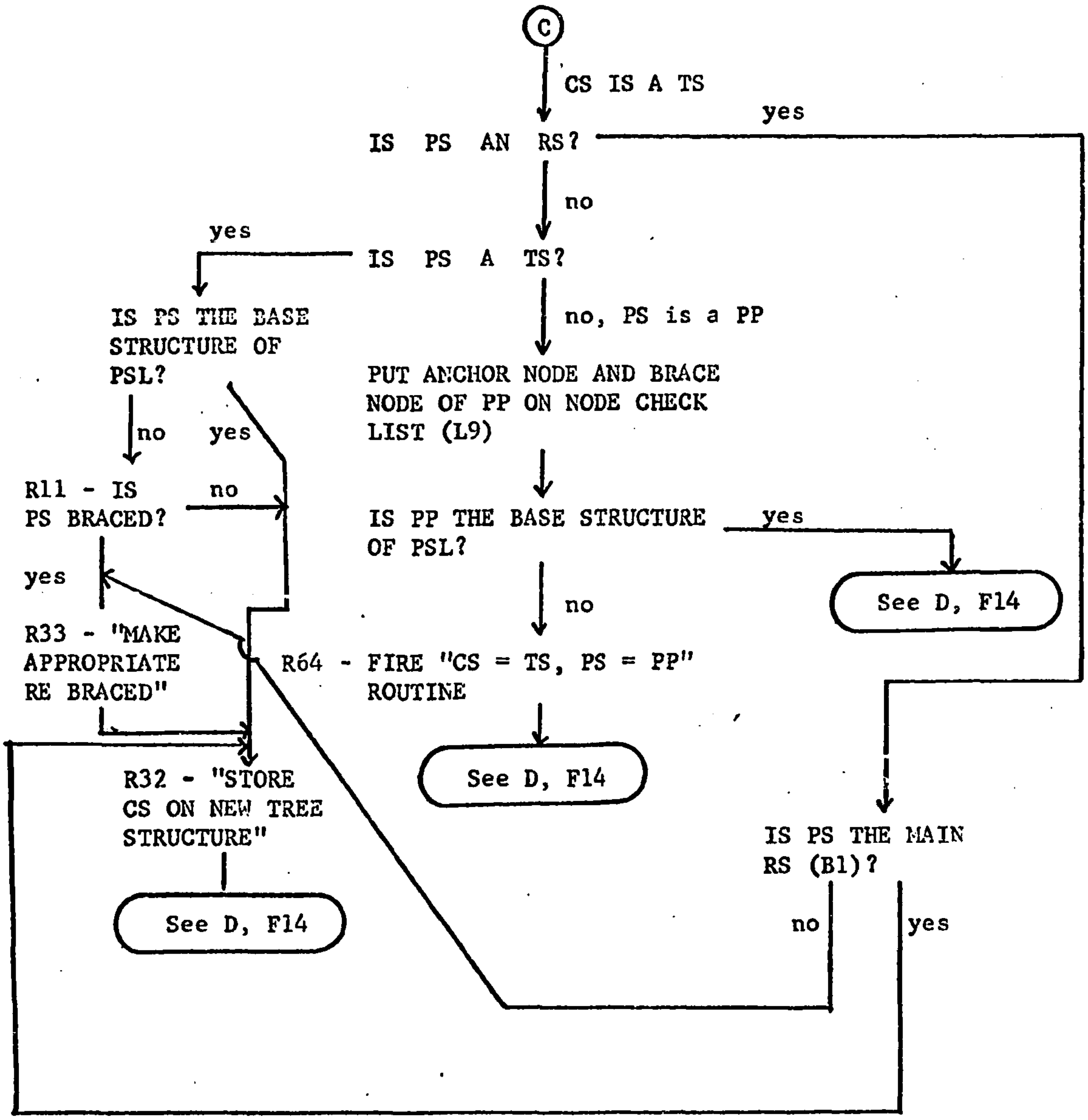

C
CS IS A TS
IS PS AN RS?
yes
no
IS PS A TS?
yes
no, PS is a PP
IS PS THE BASE STRUCTURE OF PSL?
no
yes
PUT ANCHOR NODE AND BRACE NODE OF PP ON NODE CHECK LIST (L9)
R11 - IS PS BRACED?
no
IS PP THE BASE STRUCTURE OF PSL?
yes
See D, F14
yes
no
R33 - "MAKE APPROPRIATE RE BRACED"
R64 - FIRE "CS = TS, PS = PP" ROUTINE
R32 - "STORE CS ON NEW TREE STRUCTURE"
See D, F14
IS PS THE MAIN RS (B1)?
See D, F14
no
yes

CHART F15

R29 - ADD STRUCTURE S TO NEW TREE STRUCTURE NTS

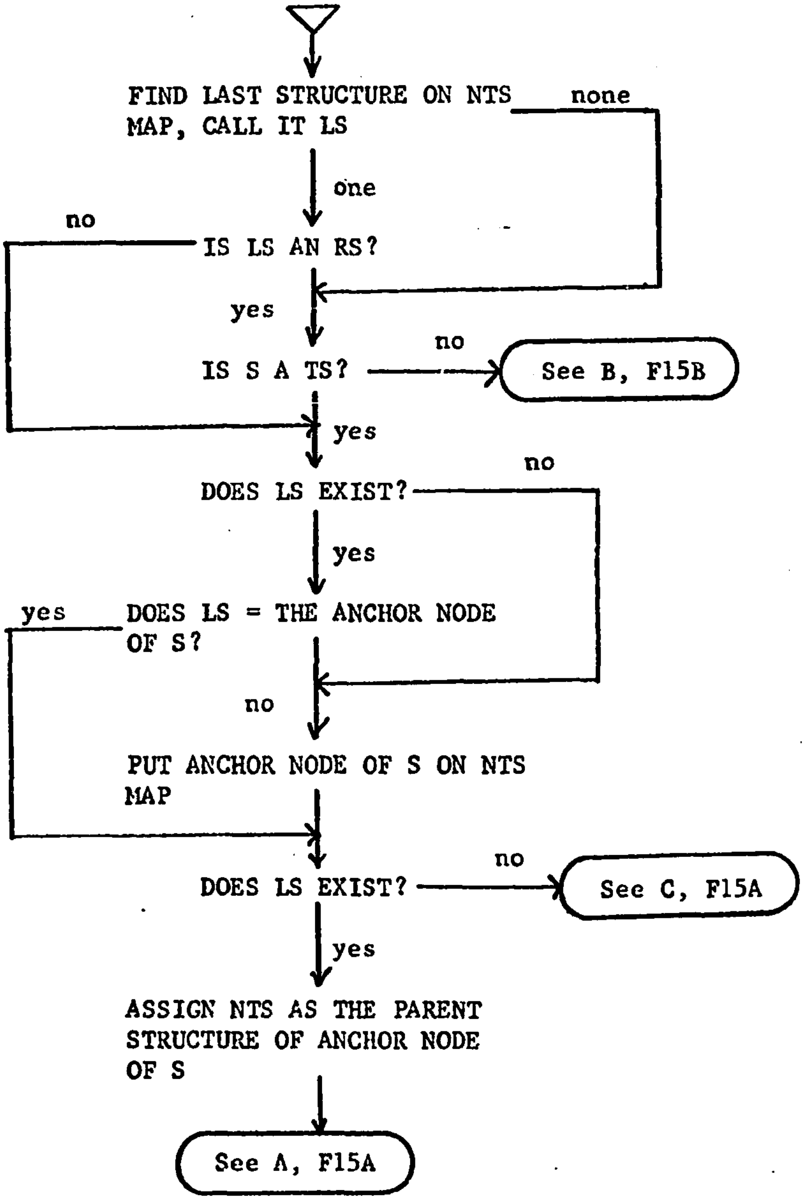

FIND LAST STRUCTURE ON NTS MAP, CALL IT LS
none
one
no
IS LS AN RS?
yes
no
IS S A TS?
See B, F15B
yes
no
DOES LS EXIST?
yes
yes
DOES LS = THE ANCHOR NODE OF S?
no
PUT ANCHOR NODE OF S ON NTS MAP
no
DOES LS EXIST?
See C, F15A
yes
ASSIGN NTS AS THE PARENT STRUCTURE OF ANCHOR NODE OF S
See A, F15A

CHART F15A

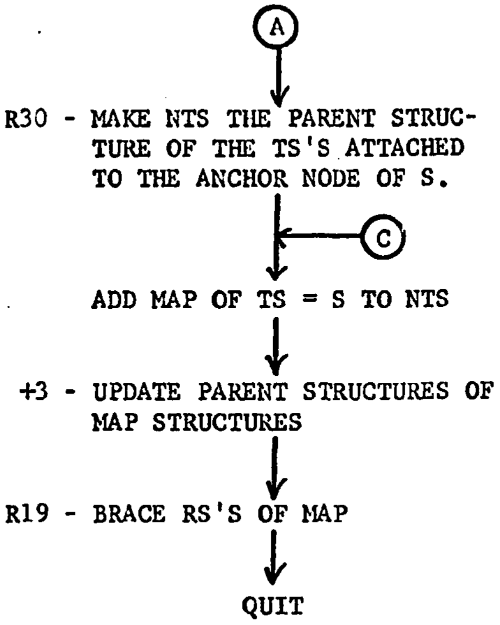

A
R30 - MAKE NTS THE PARENT STRUCTURE OF THE TS'S ATTACHED TO THE ANCHOR NODE OF S.
C
ADD MAP OF TS = S TO NTS
+3 - UPDATE PARENT STRUCTURES OF MAP STRUCTURES
R19 - BRACE RS'S OF MAP
QUIT

CHART F15B

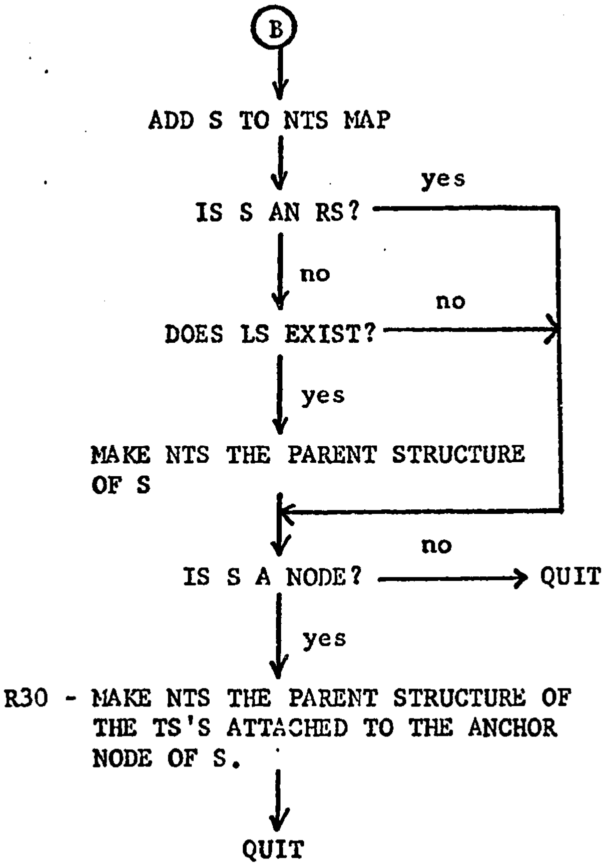

B
ADD S TO NTS MAP
IS S AN RS?
yes
no
DOES LS EXIST?
no
yes
MAKE NTS THE PARENT STRUCTURE OF S
IS S A NODE?
no
QUIT
yes
R30 - MAKE NTS THE PARENT STRUCTURE OF THE TS'S ATTACHED TO THE ANCHOR NODE OF S.
QUIT

CHART F16

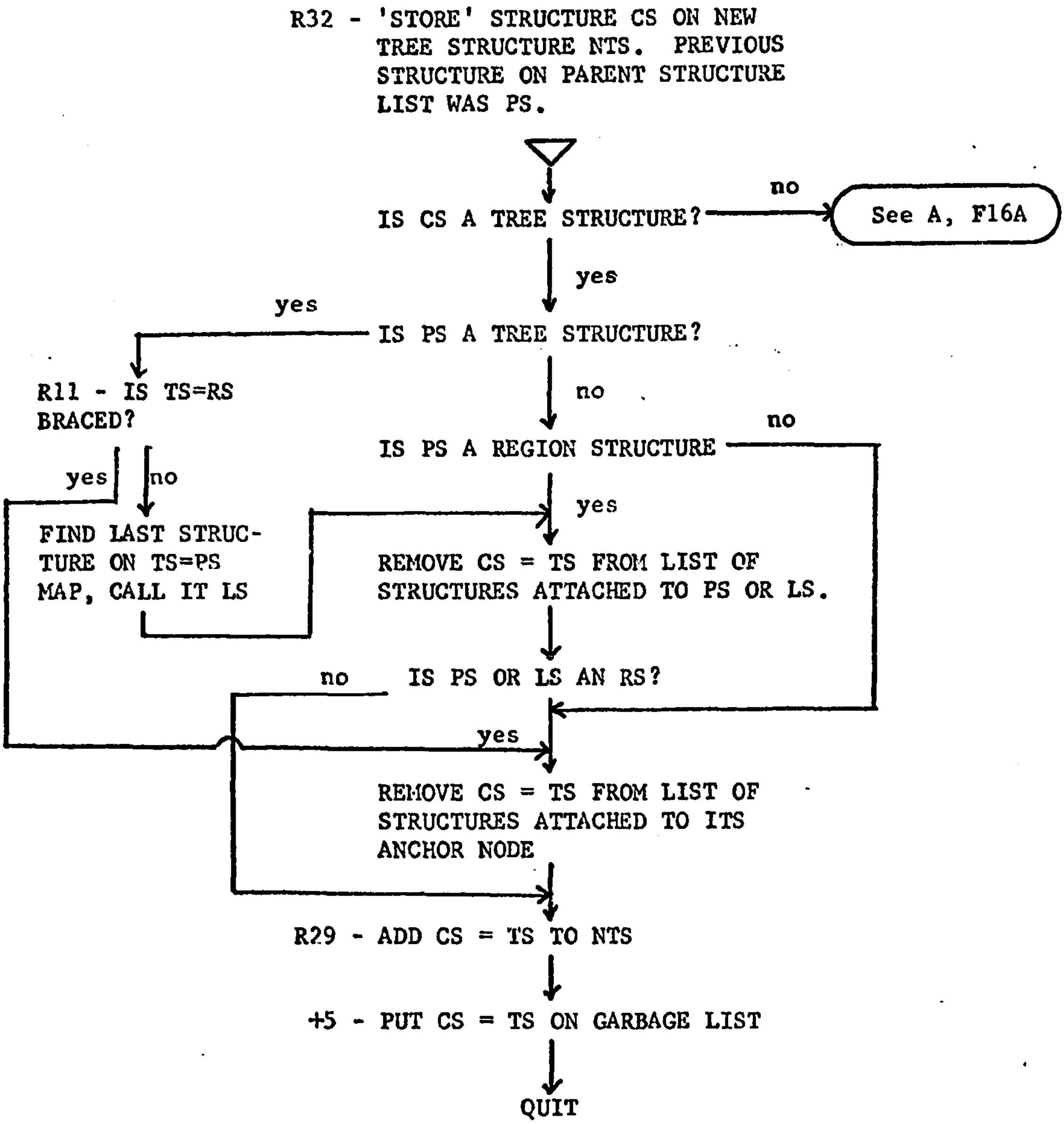

R32 - 'STORE' STRUCTURE CS ON NEW TREE STRUCTURE NTS. PREVIOUS STRUCTURE ON PARENT STRUCTURE LIST WAS PS.
IS CS A TREE STRUCTURE?
no
See A, F16A
yes
IS PS A TREE STRUCTURE?
yes
R11 - IS TS=RS BRACED?
yes
no
FIND LAST STRUCTURE ON TS=PS MAP, CALL IT LS
no
IS PS A REGION STRUCTURE
no
yes
REMOVE CS = TS FROM LIST OF STRUCTURES ATTACHED TO PS OR LS.
no
IS PS OR LS AN RS?
yes
REMOVE CS = TS FROM LIST OF STRUCTURES ATTACHED TO ITS ANCHOR NODE
R29 - ADD CS = TS TO NTS
+5 - PUT CS = TS ON GARBAGE LIST
QUIT

CHART F16A

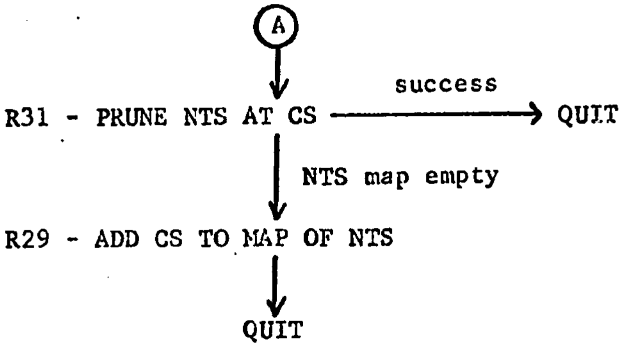
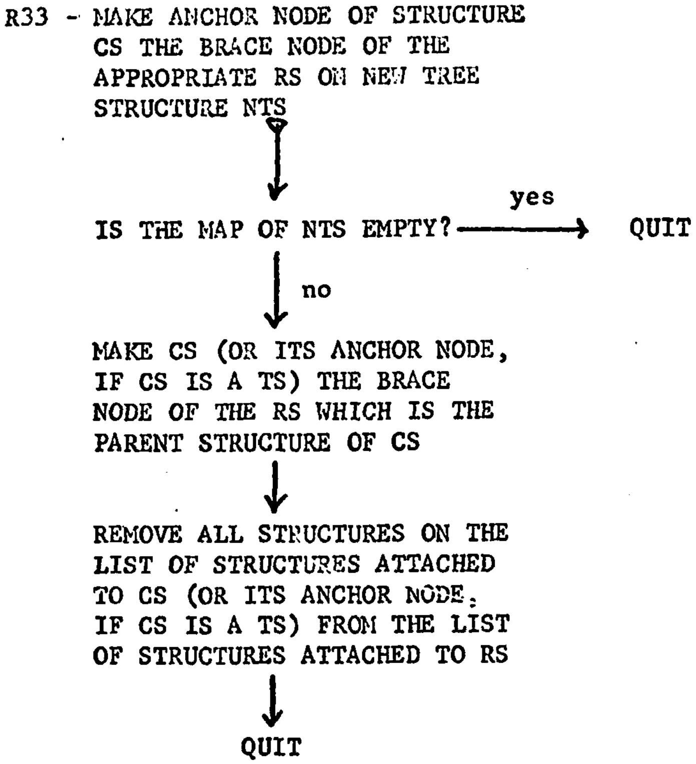

A
R31 - PRUNE NTS AT CS
success
QUIT
NTS map empty
R29 - ADD CS TO MAP OF NTS
QUIT

CHART F17



R33 - MAKE ANCHOR NODE OF STRUCTURE CS THE BRACE NODE OF THE APPROPRIATE RS ON NEW TREE STRUCTURE NTS

IS THE MAP OF NTS EMPTY?

MAKE CS (OR ITS ANCHOR NODE, IF CS IS A TS) THE BRACE NODE OF THE RS WHICH IS THE PARENT STRUCTURE OF CS

REMOVE ALL STRUCTURES ON THE LIST OF STRUCTURES ATTACHED TO CS (OR ITS ANCHOR NODE, IF CS IS A TS) FROM THE LIST OF STRUCTURES ATTACHED TO RS

CHART F18

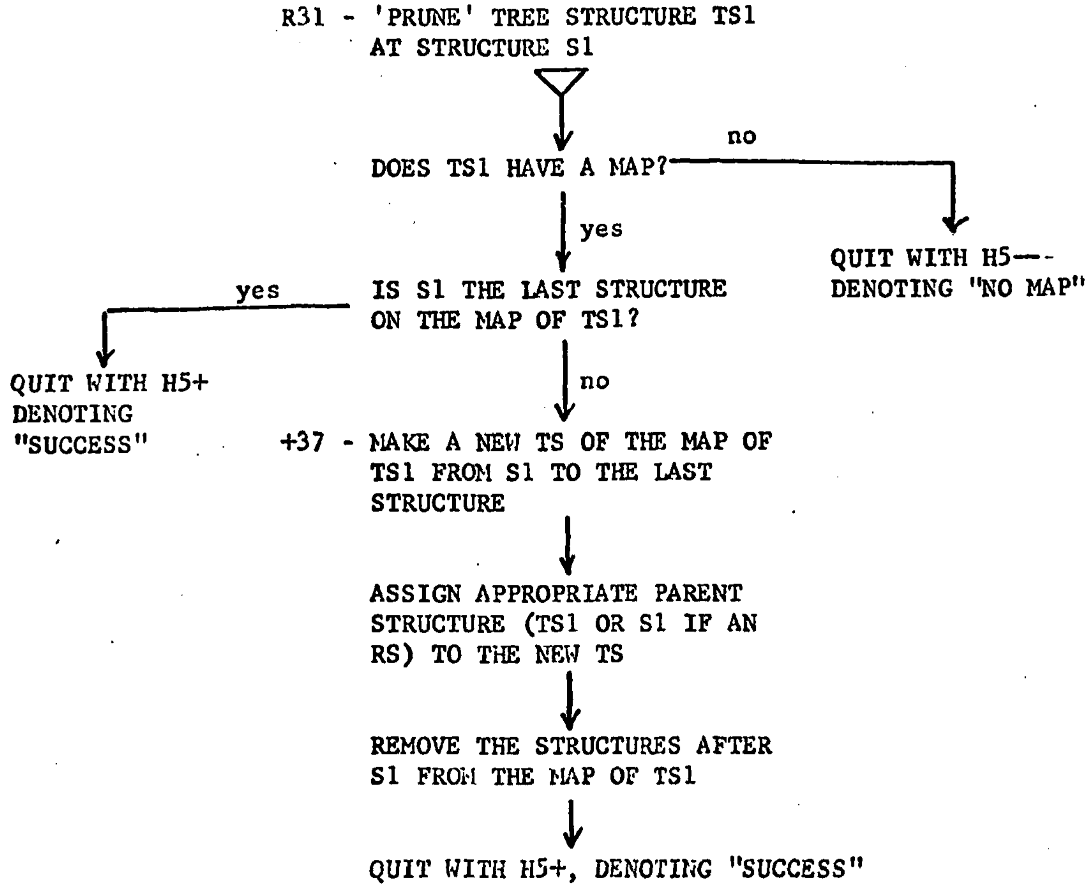

R31 - 'PRUNE' TREE STRUCTURE TS1
AT STRUCTURE S1
DOES TS1 HAVE A MAP?
no
QUIT WITH H5—-
DENOTING "NO MAP"
yes
IS S1 THE LAST STRUCTURE
ON THE MAP OF TS1?
yes
QUIT WITH H5+
DENOTING
"SUCCESS"
no
+37 - MAKE A NEW TS OF THE MAP OF
TS1 FROM S1 TO THE LAST
STRUCTURE
ASSIGN APPROPRIATE PARENT
STRUCTURE (TS1 OR S1 IF AN
RS) TO THE NEW TS
REMOVE THE STRUCTURES AFTER
S1 FROM THE MAP OF TS1
QUIT WITH H5+, DENOTING "SUCCESS"

CHART F19

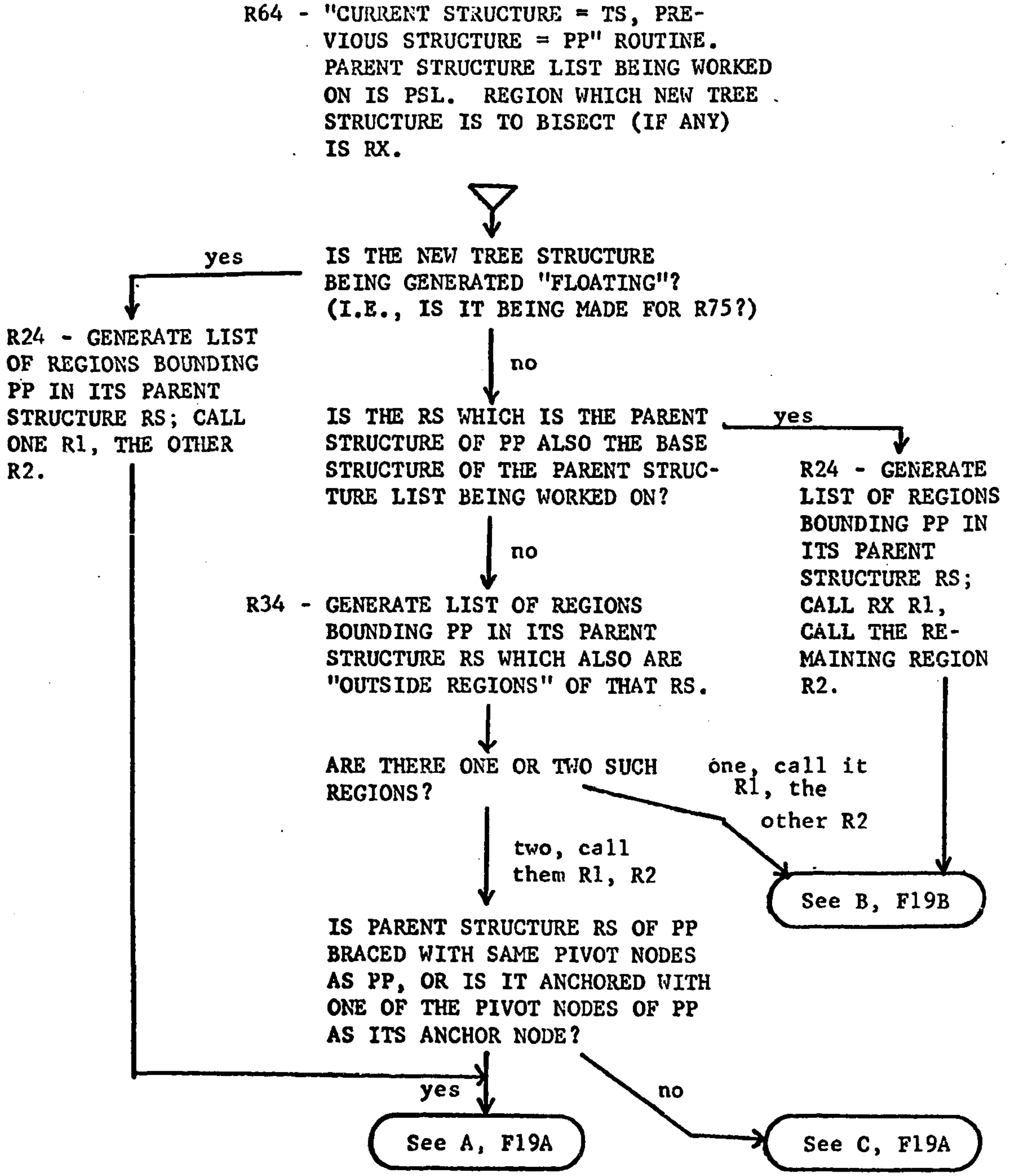

R64 - "CURRENT STRUCTURE = TS, PREVIOUS STRUCTURE = PP" ROUTINE. PARENT STRUCTURE LIST BEING WORKED ON IS PSL. REGION WHICH NEW TREE STRUCTURE IS TO BISECT (IF ANY) IS RX.
IS THE NEW TREE STRUCTURE BEING GENERATED "FLOATING"? (I.E., IS IT BEING MADE FOR R75?)
yes
R24 - GENERATE LIST OF REGIONS BOUNDING PP IN ITS PARENT STRUCTURE RS; CALL ONE R1, THE OTHER R2.
no
IS THE RS WHICH IS THE PARENT STRUCTURE OF PP ALSO THE BASE STRUCTURE OF THE PARENT STRUCTURE LIST BEING WORKED ON?
yes
R24 - GENERATE LIST OF REGIONS BOUNDING PP IN ITS PARENT STRUCTURE RS; CALL RX R1, CALL THE REMAINING REGION R2.
no
R34 - GENERATE LIST OF REGIONS BOUNDING PP IN ITS PARENT STRUCTURE RS WHICH ALSO ARE "OUTSIDE REGIONS" OF THAT RS.
ARE THERE ONE OR TWO SUCH REGIONS?
one, call it R1, the other R2
See B, F19B
two, call them R1, R2
IS PARENT STRUCTURE RS OF PP BRACED WITH SAME PIVOT NODES AS PP, OR IS IT ANCHORED WITH ONE OF THE PIVOT NODES OF PP AS ITS ANCHOR NODE?
yes
See A, F19A
no
See C, F19A

CHART F19A

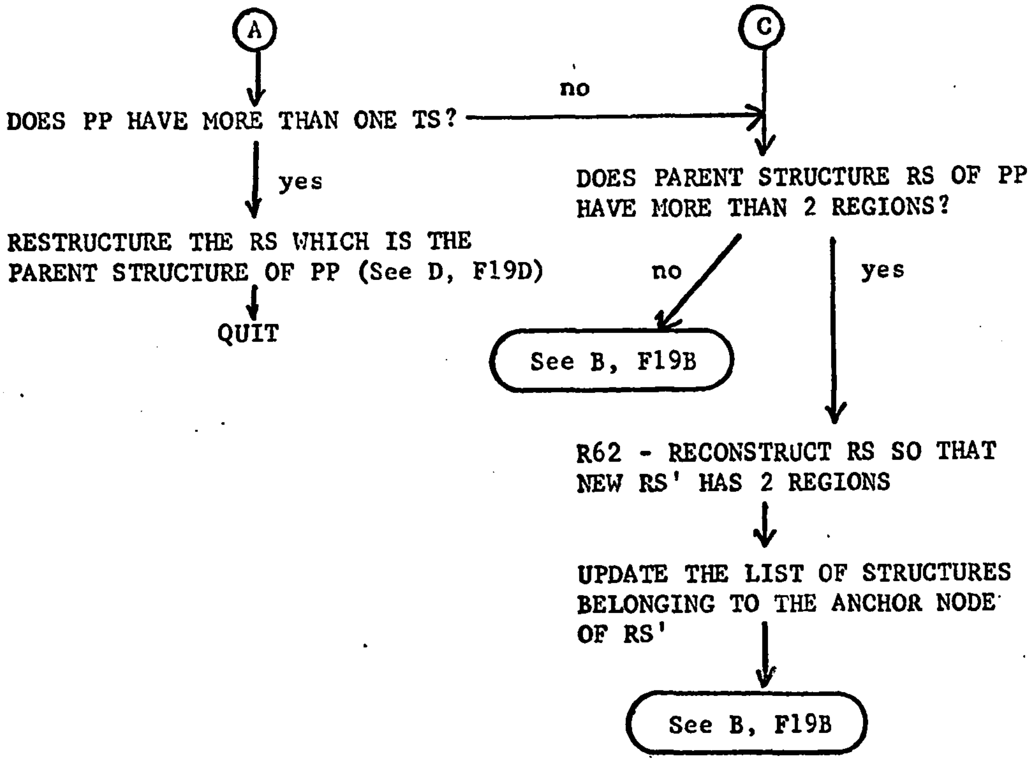

A
C
DOES PP HAVE MORE THAN ONE TS?
no
yes
RESTRUCTURE THE RS WHICH IS THE PARENT STRUCTURE OF PP (See D, F19D)
QUIT
DOES PARENT STRUCTURE RS OF PP HAVE MORE THAN 2 REGIONS?
no
yes
See B, F19B
R62 - RECONSTRUCT RS SO THAT NEW RS' HAS 2 REGIONS
UPDATE THE LIST OF STRUCTURES BELONGING TO THE ANCHOR NODE OF RS'
See B, F19B

CHART F19B

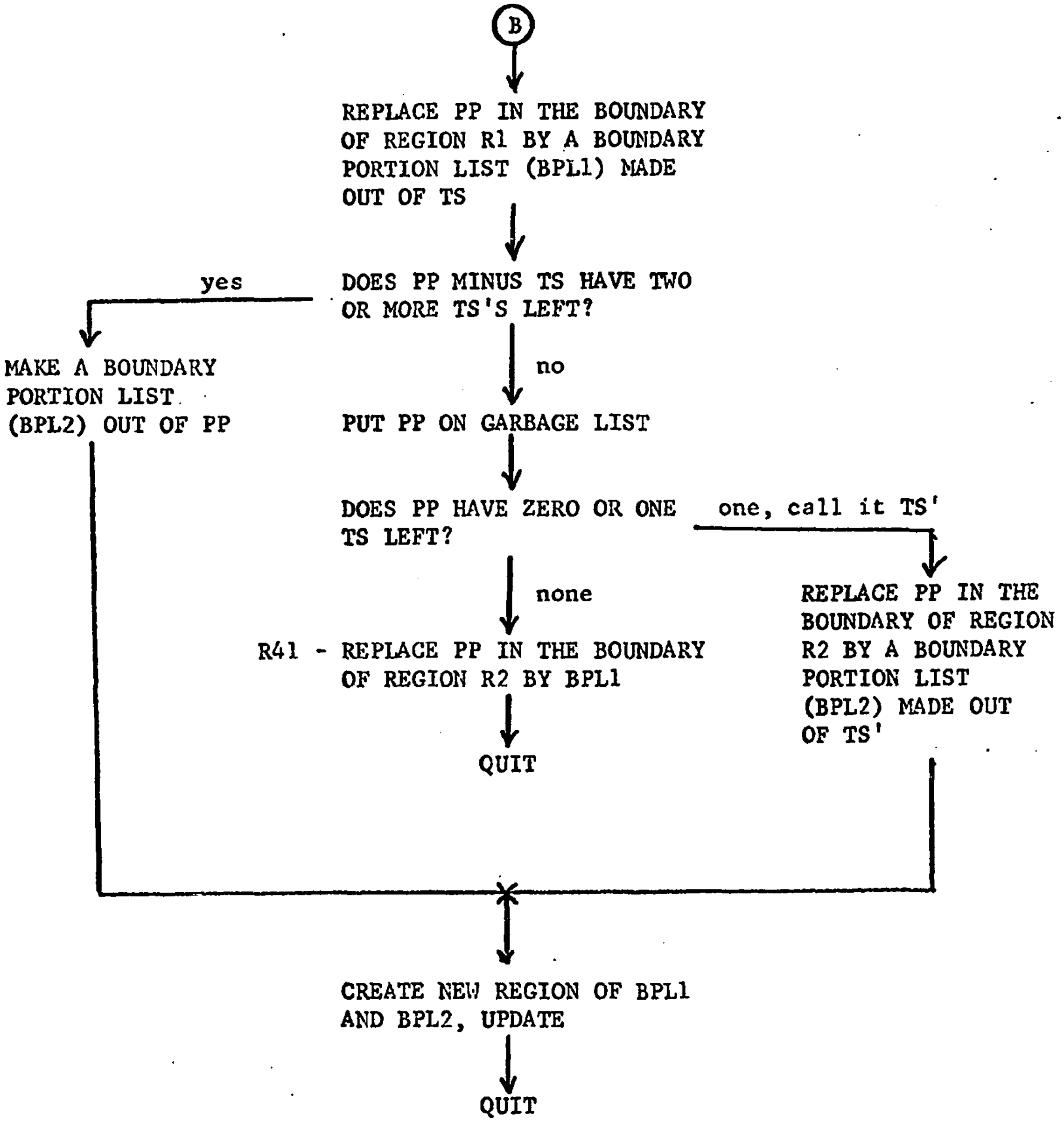
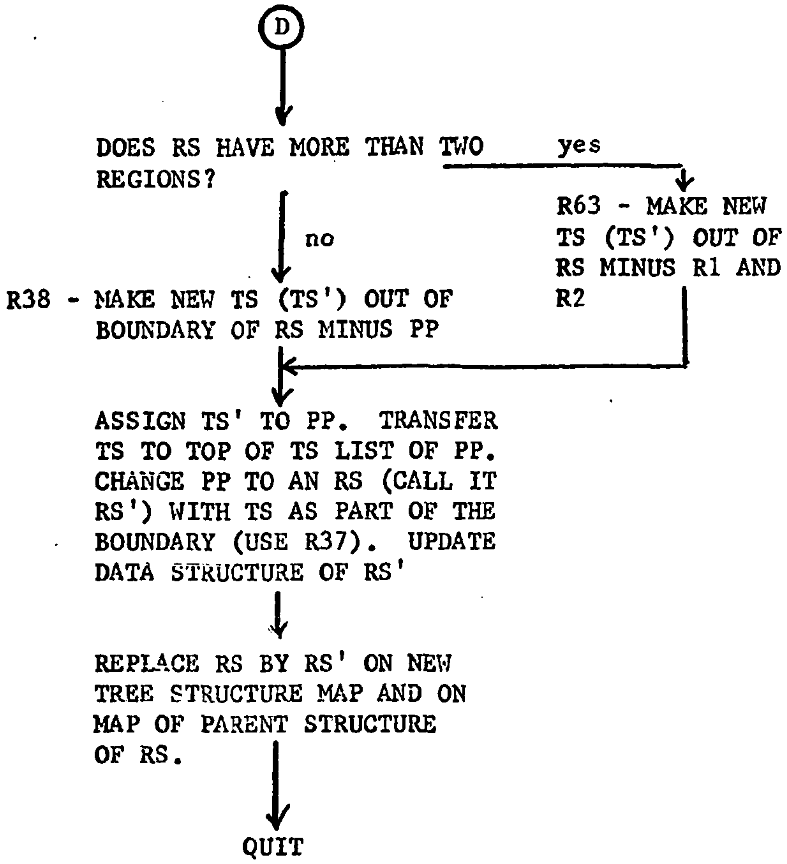

B
REPLACE PP IN THE BOUNDARY OF REGION R1 BY A BOUNDARY PORTION LIST (BPL1) MADE OUT OF TS
DOES PP MINUS TS HAVE TWO OR MORE TS'S LEFT?
yes
MAKE A BOUNDARY PORTION LIST (BPL2) OUT OF PP
no
PUT PP ON GARBAGE LIST
DOES PP HAVE ZERO OR ONE TS LEFT?
one, call it TS'
REPLACE PP IN THE BOUNDARY OF REGION R2 BY A BOUNDARY PORTION LIST (BPL2) MADE OUT OF TS'
none
R41 - REPLACE PP IN THE BOUNDARY OF REGION R2 BY BPL1
QUIT
CREATE NEW REGION OF BPL1 AND BPL2, UPDATE
QUIT

CHART F19D



D

DOES RS HAVE MORE THAN TWO REGIONS?

yes

R63 - MAKE NEW TS (TS') OUT OF RS MINUS R1 AND R2

no

R38 - MAKE NEW TS (TS') OUT OF BOUNDARY OF RS MINUS PP

ASSIGN TS' TO PP. TRANSFER TS TO TOP OF TS LIST OF PP. CHANGE PP TO AN RS (CALL IT RS') WITH TS AS PART OF THE BOUNDARY (USE R37). UPDATE DATA STRUCTURE OF RS'

REPLACE RS BY RS' ON NEW TREE STRUCTURE MAP AND ON MAP OF PARENT STRUCTURE OF RS.

QUIT

CHART F20

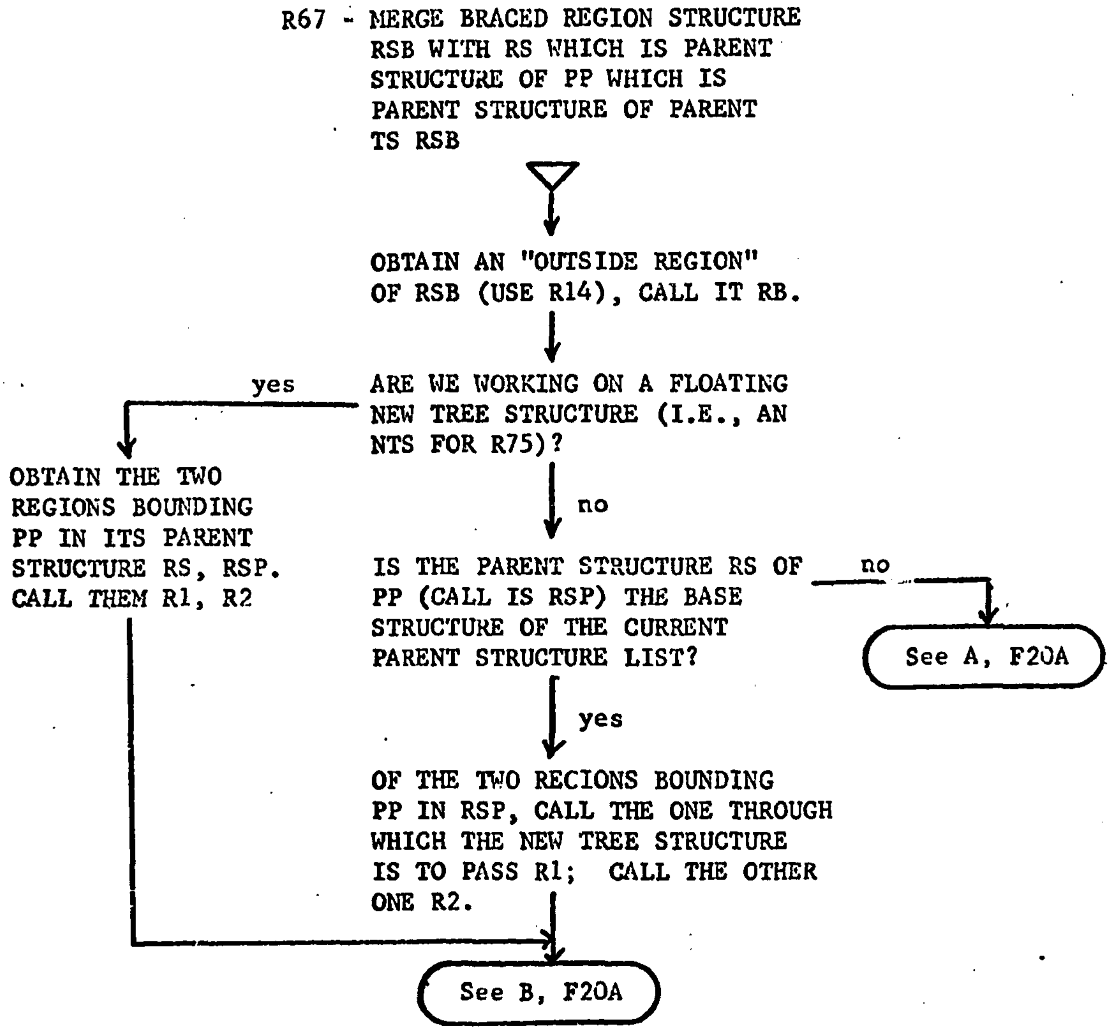
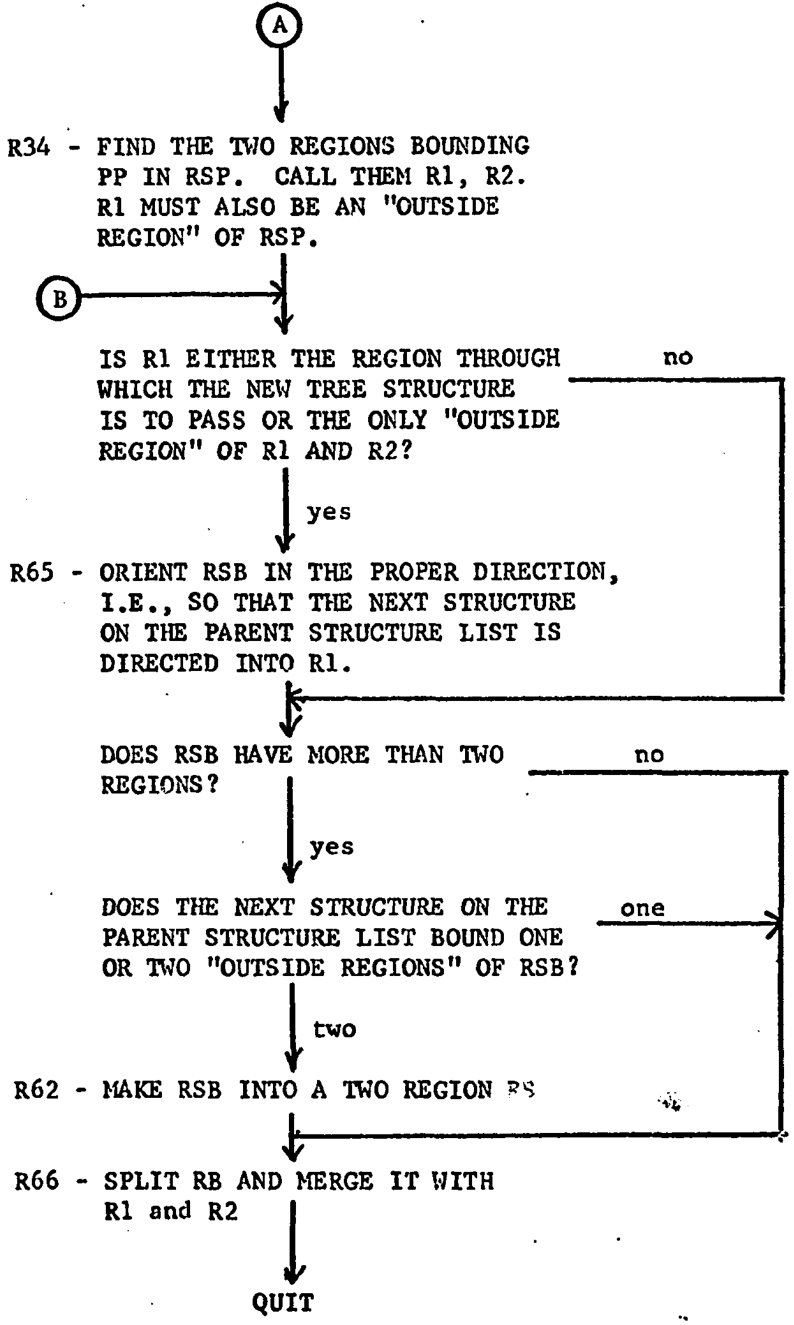

R67 - MERGE BRACED REGION STRUCTURE RSB WITH RS WHICH IS PARENT STRUCTURE OF PP WHICH IS PARENT STRUCTURE OF PARENT TS RSB
OBTAIN AN "OUTSIDE REGION" OF RSB (USE R14), CALL IT RB.
ARE WE WORKING ON A FLOATING NEW TREE STRUCTURE (I.E., AN NTS FOR R75)?
yes
OBTAIN THE TWO REGIONS BOUNDING PP IN ITS PARENT STRUCTURE RS, RSP. CALL THEM R1, R2
no
IS THE PARENT STRUCTURE RS OF PP (CALL IS RSP) THE BASE STRUCTURE OF THE CURRENT PARENT STRUCTURE LIST?
no
See A, F20A
yes
OF THE TWO RECIONS BOUNDING PP IN RSP, CALL THE ONE THROUGH WHICH THE NEW TREE STRUCTURE IS TO PASS R1; CALL THE OTHER ONE R2.
See B, F20A

CHART F20A



R34 - FIND THE TWO REGIONS BOUNDING PP IN RSP. CALL THEM R1, R2. R1 MUST ALSO BE AN "OUTSIDE REGION" OF RSP.

IS R1 EITHER THE REGION THROUGH WHICH THE NEW TREE STRUCTURE IS TO PASS OR THE ONLY "OUTSIDE REGION" OF R1 AND R2?

R65 - ORIENT RSB IN THE PROPER DIRECTION, I.E., SO THAT THE NEXT STRUCTURE ON THE PARENT STRUCTURE LIST IS DIRECTED INTO R1.

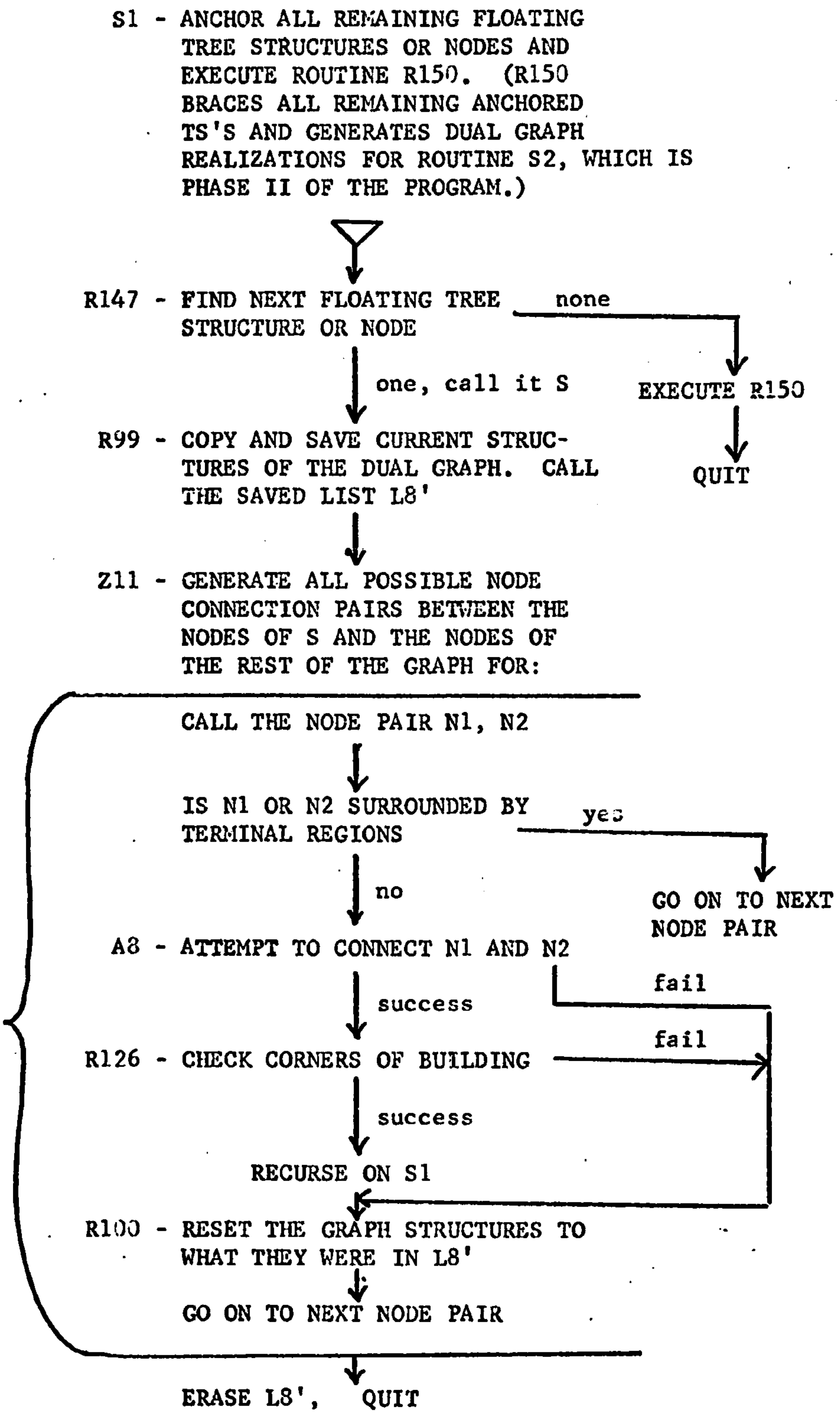
DOES RSB HAVE MORE THAN TWO REGIONS?

DOES THE NEXT STRUCTURE ON THE PARENT STRUCTURE LIST BOUND ONE OR TWO "OUTSIDE REGIONS" OF RSB?

R62 - MAKE RSB INTO A TWO REGION RS

R66 - SPLIT RB AND MERGE IT WITH R1 and R2

CHART F21



S1 - ANCHOR ALL REMAINING FLOATING TREE STRUCTURES OR NODES AND EXECUTE ROUTINE R150. (R150 BRACES ALL REMAINING ANCHORED TS'S AND GENERATES DUAL GRAPH REALIZATIONS FOR ROUTINE S2, WHICH IS PHASE II OF THE PROGRAM.)

R147 - FIND NEXT FLOATING TREE STRUCTURE OR NODE

EXECUTE R150

QUIT

R99 - COPY AND SAVE CURRENT STRUCTURES OF THE DUAL GRAPH. CALL THE SAVED LIST L8'

Z11 - GENERATE ALL POSSIBLE NODE CONNECTION PAIRS BETWEEN THE NODES OF S AND THE NODES OF THE REST OF THE GRAPH FOR:

CALL THE NODE PAIR N1, N2

IS N1 OR N2 SURROUNDED BY TERMINAL REGIONS

GO ON TO NEXT NODE PAIR

A8 - ATTEMPT TO CONNECT N1 AND N2

R126 - CHECK CORNERS OF BUILDING

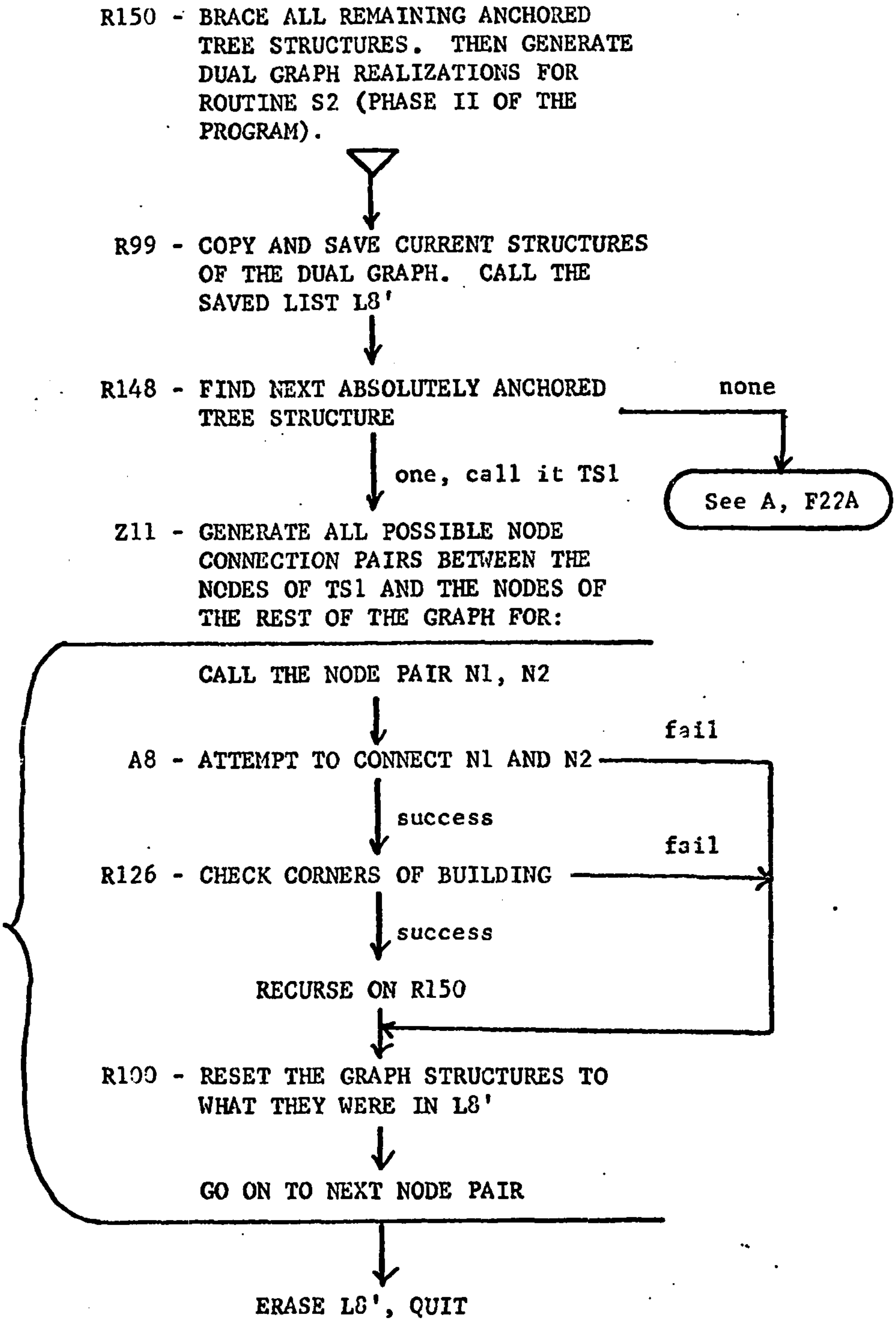
RECURSE ON S1

R100 - RESET THE GRAPH STRUCTURES TO WHAT THEY WERE IN L8'

GO ON TO NEXT NODE PAIR

ERASE L8', QUIT

CHART F22



R150 - BRACE ALL REMAINING ANCHORED TREE STRUCTURES. THEN GENERATE DUAL GRAPH REALIZATIONS FOR ROUTINE S2 (PHASE II OF THE PROGRAM).

R99 - COPY AND SAVE CURRENT STRUCTURES OF THE DUAL GRAPH. CALL THE SAVED LIST L8'

R148 - FIND NEXT ABSOLUTELY ANCHORED TREE STRUCTURE

Z11 - GENERATE ALL POSSIBLE NODE CONNECTION PAIRS BETWEEN THE NODES OF TS1 AND THE NODES OF THE REST OF THE GRAPH FOR:

CALL THE NODE PAIR N1, N2

A8 - ATTEMPT TO CONNECT N1 AND N2

R126 - CHECK CORNERS OF BUILDING

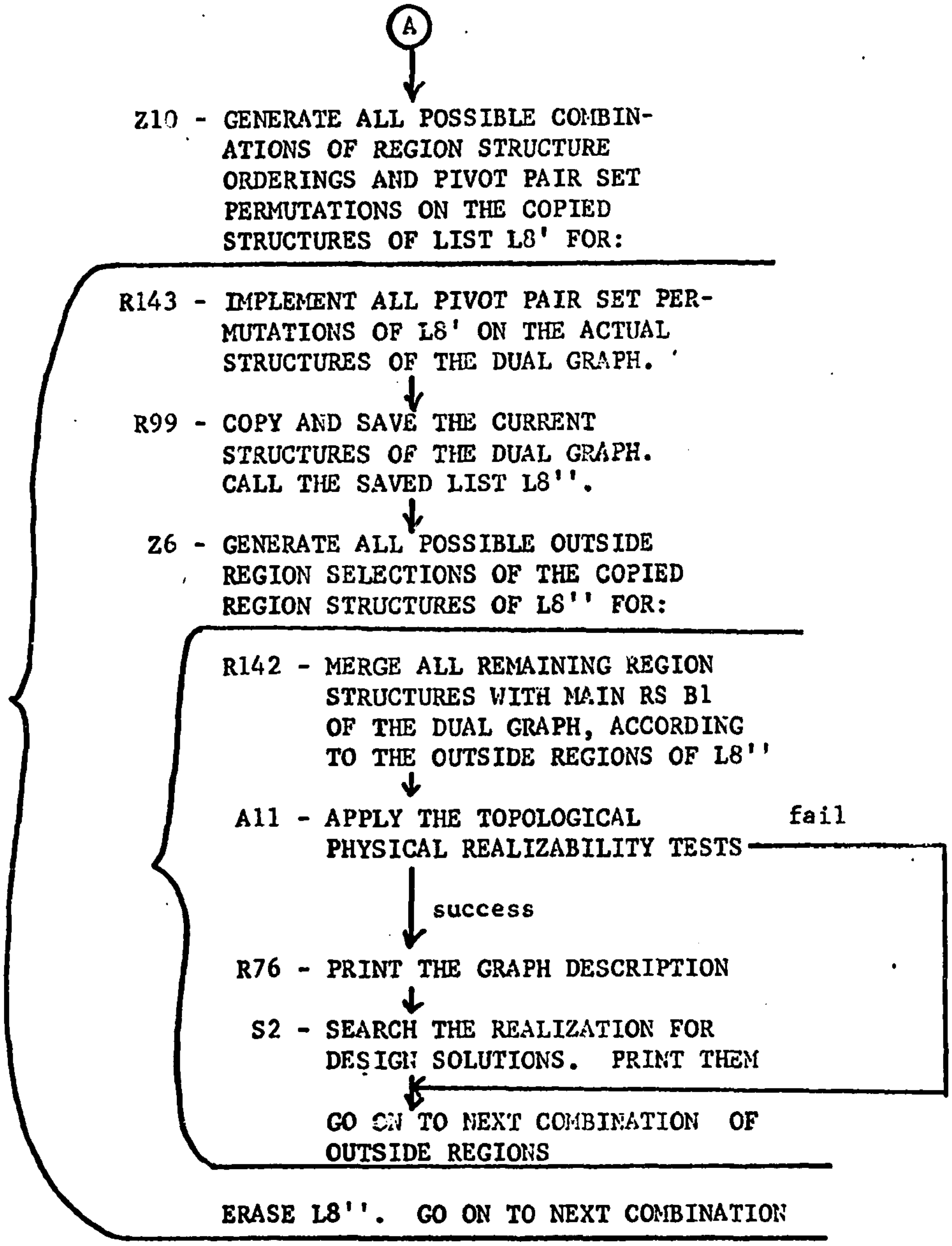
RECURSE ON R150

R100 - RESET THE GRAPH STRUCTURES TO WHAT THEY WERE IN L8'

GO ON TO NEXT NODE PAIR

ERASE L8', QUIT

CHART F22A



Z10 - GENERATE ALL POSSIBLE COMBINATIONS OF REGION STRUCTURE ORDERINGS AND PIVOT PAIR SET PERMUTATIONS ON THE COPIED STRUCTURES OF LIST L8' FOR:

R143 - IMPLEMENT ALL PIVOT PAIR SET PERMUTATIONS OF L8' ON THE ACTUAL STRUCTURES OF THE DUAL GRAPH.

R99 - COPY AND SAVE THE CURRENT STRUCTURES OF THE DUAL GRAPH. CALL THE SAVED LIST L8''.

Z6 - GENERATE ALL POSSIBLE OUTSIDE REGION SELECTIONS OF THE COPIED REGION STRUCTURES OF L8'' FOR:

R142 - MERGE ALL REMAINING REGION STRUCTURES WITH MAIN RS B1 OF THE DUAL GRAPH, ACCORDING TO THE OUTSIDE REGIONS OF L8''

A11 - APPLY THE TOPOLOGICAL PHYSICAL REALIZABILITY TESTS

R76 - PRINT THE GRAPH DESCRIPTION

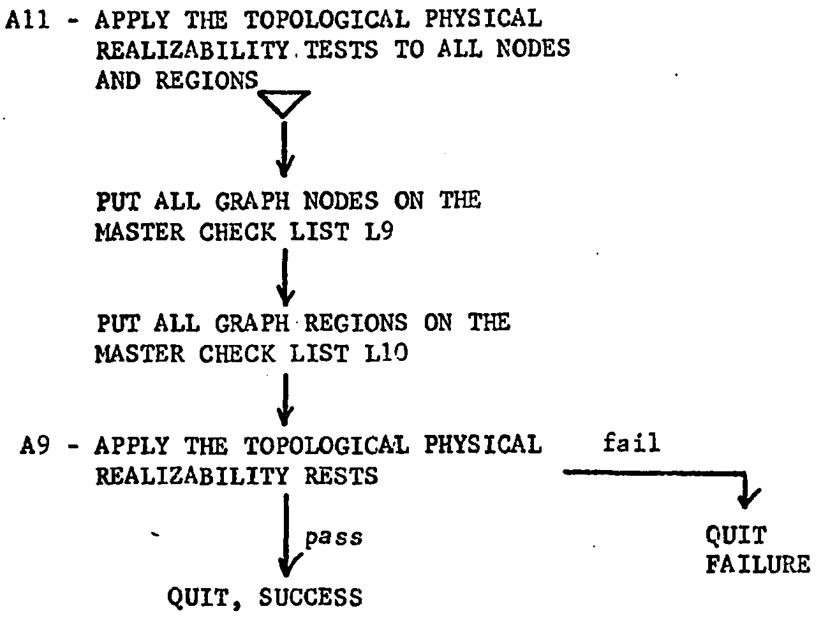
S2 - SEARCH THE REALIZATION FOR DESIGN SOLUTIONS. PRINT THEM

GO ON TO NEXT COMBINATION OF OUTSIDE REGIONS

ERASE L8''. GO ON TO NEXT COMBINATION

ERASE L8' QUIT

CHART F23



A11 - APPLY THE TOPOLOGICAL PHYSICAL REALIZABILITY TESTS TO ALL NODES AND REGIONS

PUT ALL GRAPH NODES ON THE MASTER CHECK LIST L9

PUT ALL GRAPH REGIONS ON THE MASTER CHECK LIST L10

A9 - APPLY THE TOPOLOGICAL PHYSICAL REALIZABILITY RESTS

QUIT, SUCCESS

QUIT FAILURE

CHART F24

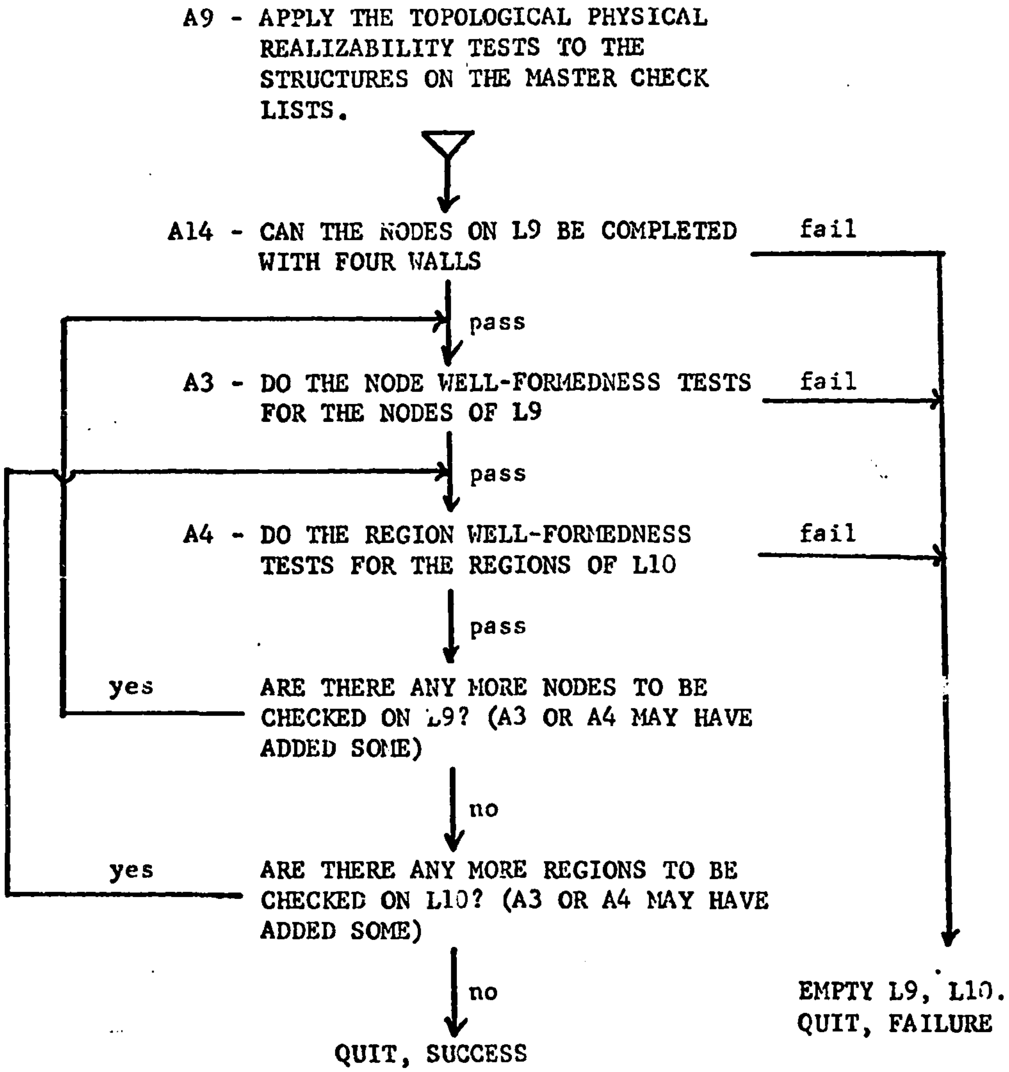
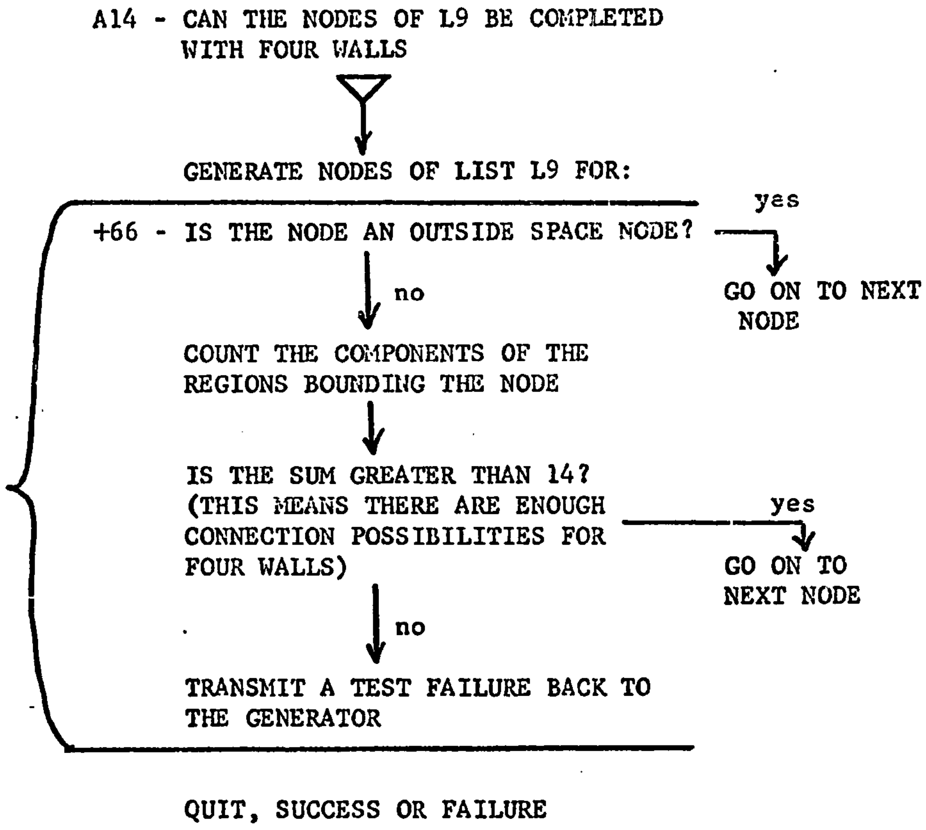

A9 - APPLY THE TOPOLOGICAL PHYSICAL REALIZABILITY TESTS TO THE STRUCTURES ON THE MASTER CHECK LISTS.
A14 - CAN THE NODES ON L9 BE COMPLETED WITH FOUR WALLS
fail
pass
A3 - DO THE NODE WELL-FORMEDNESS TESTS FOR THE NODES OF L9
fail
pass
A4 - DO THE REGION WELL-FORMEDNESS TESTS FOR THE REGIONS OF L10
fail
pass
yes
ARE THERE ANY MORE NODES TO BE CHECKED ON L9? (A3 OR A4 MAY HAVE ADDED SOME)
no
yes
ARE THERE ANY MORE REGIONS TO BE CHECKED ON L10? (A3 OR A4 MAY HAVE ADDED SOME)
no
QUIT, SUCCESS
EMPTY L9, L10.
QUIT, FAILURE

CHART F25



A14 - CAN THE NODES OF L9 BE COMPLETED WITH FOUR WALLS

GENERATE NODES OF LIST L9 FOR:

+66 - IS THE NODE AN OUTSIDE SPACE NODE?

GO ON TO NEXT NODE

COUNT THE COMPONENTS OF THE REGIONS BOUNDING THE NODE

IS THE SUM GREATER THAN 14? (THIS MEANS THERE ARE ENOUGH CONNECTION POSSIBILITIES FOR FOUR WALLS)

GO ON TO NEXT NODE

TRANSMIT A TEST FAILURE BACK TO THE GENERATOR

QUIT, SUCCESS OR FAILURE

CHART F26

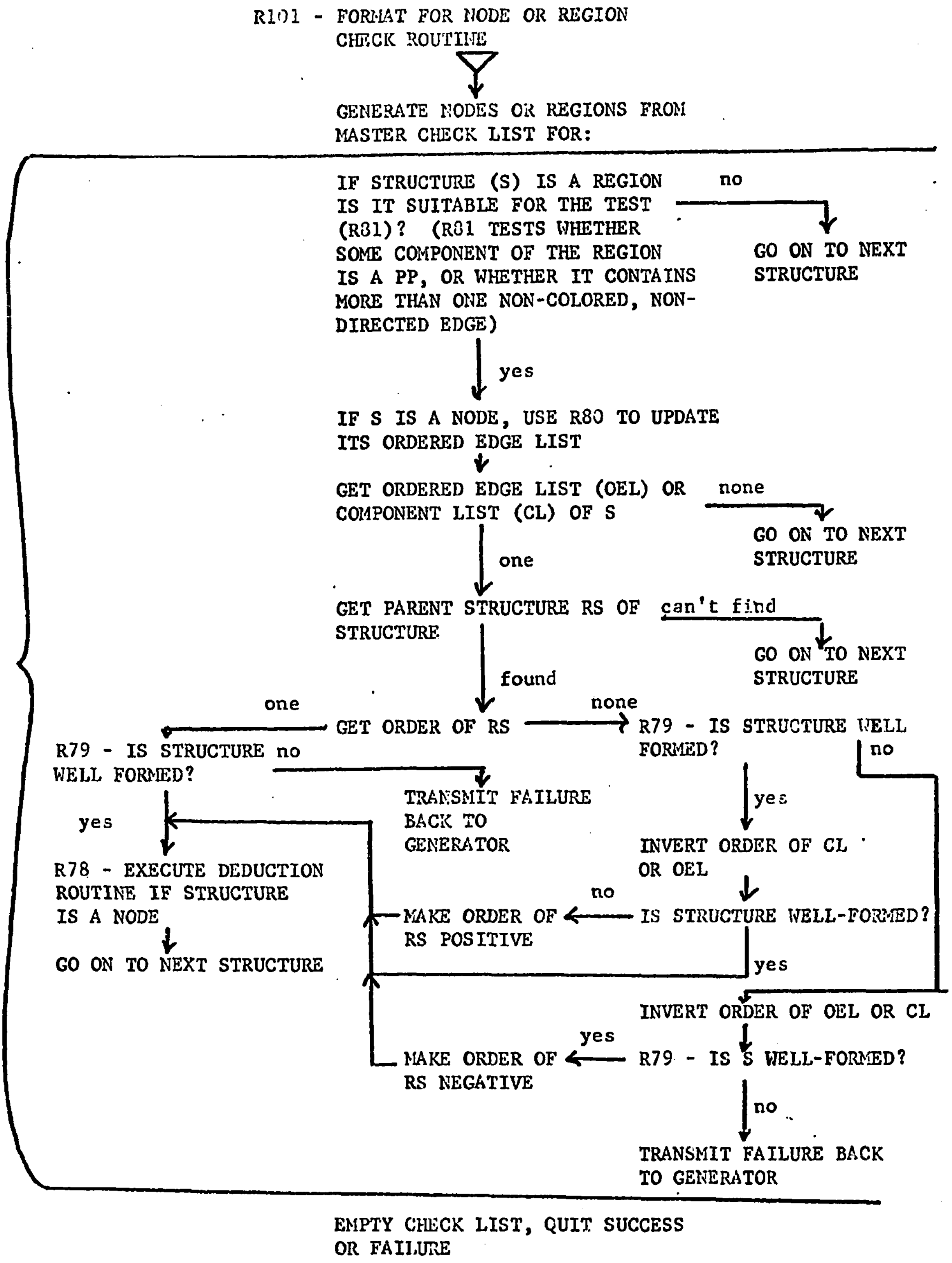

R101 - FORMAT FOR NODE OR REGION CHECK ROUTINE
GENERATE NODES OR REGIONS FROM MASTER CHECK LIST FOR:
IF STRUCTURE (S) IS A REGION IS IT SUITABLE FOR THE TEST (R81)? (R81 TESTS WHETHER SOME COMPONENT OF THE REGION IS A PP, OR WHETHER IT CONTAINS MORE THAN ONE NON-COLORED, NON-DIRECTED EDGE)
no
GO ON TO NEXT STRUCTURE
yes
IF S IS A NODE, USE R80 TO UPDATE ITS ORDERED EDGE LIST
GET ORDERED EDGE LIST (OEL) OR COMPONENT LIST (CL) OF S
none
GO ON TO NEXT STRUCTURE
one
GET PARENT STRUCTURE RS OF STRUCTURE
can't find
GO ON TO NEXT STRUCTURE
found
one
GET ORDER OF RS
none
R79 - IS STRUCTURE WELL FORMED?
no
R79 - IS STRUCTURE WELL FORMED?
no
TRANSMIT FAILURE BACK TO GENERATOR
yes
yes
INVERT ORDER OF CL OR OEL
R78 - EXECUTE DEDUCTION ROUTINE IF STRUCTURE IS A NODE
no
MAKE ORDER OF RS POSITIVE
IS STRUCTURE WELL-FORMED?
GO ON TO NEXT STRUCTURE
yes
INVERT ORDER OF OEL OR CL
yes
MAKE ORDER OF RS NEGATIVE
R79 - IS S WELL-FORMED?
no
TRANSMIT FAILURE BACK TO GENERATOR
EMPTY CHECK LIST, QUIT SUCCESS OR FAILURE

CHART F27

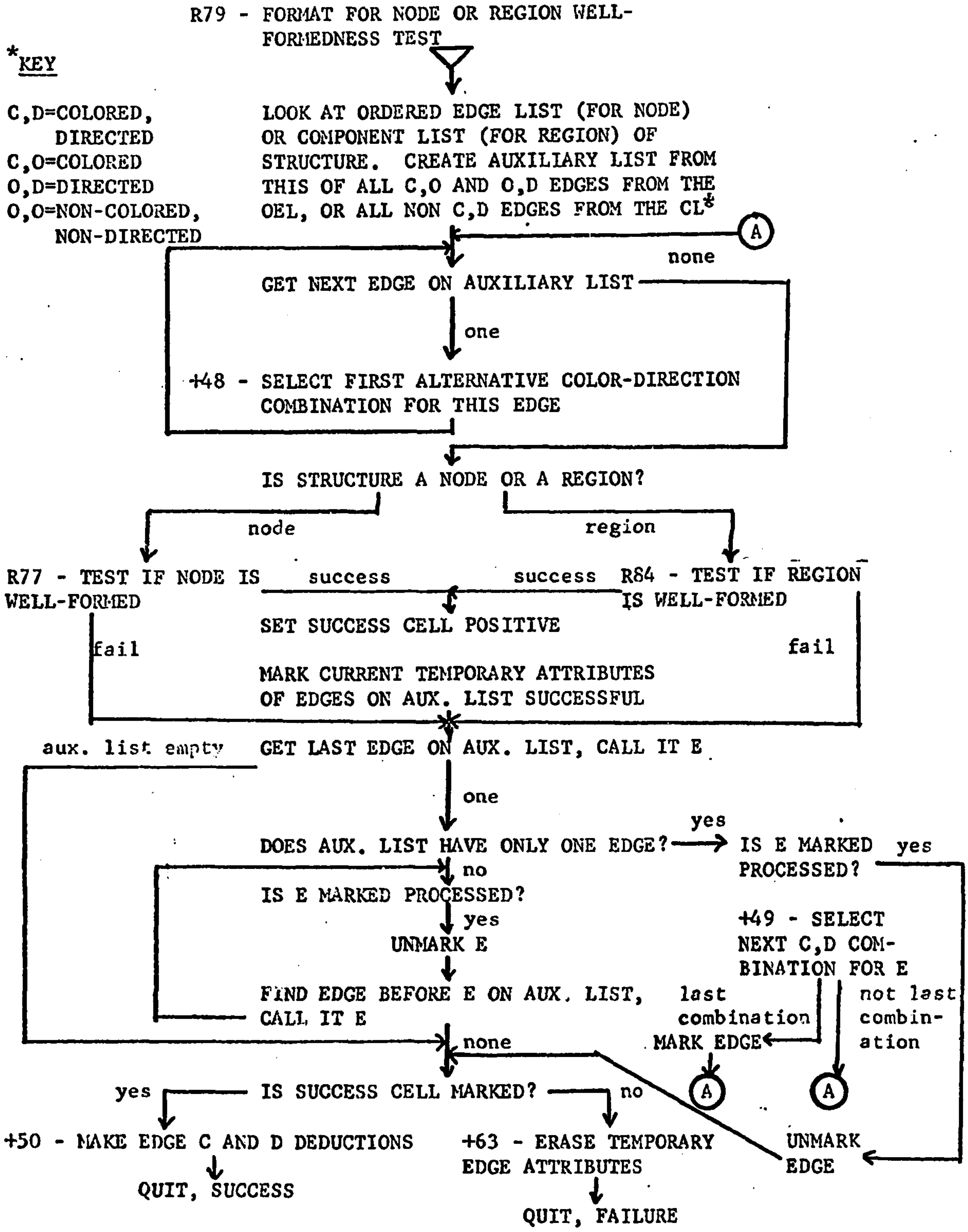
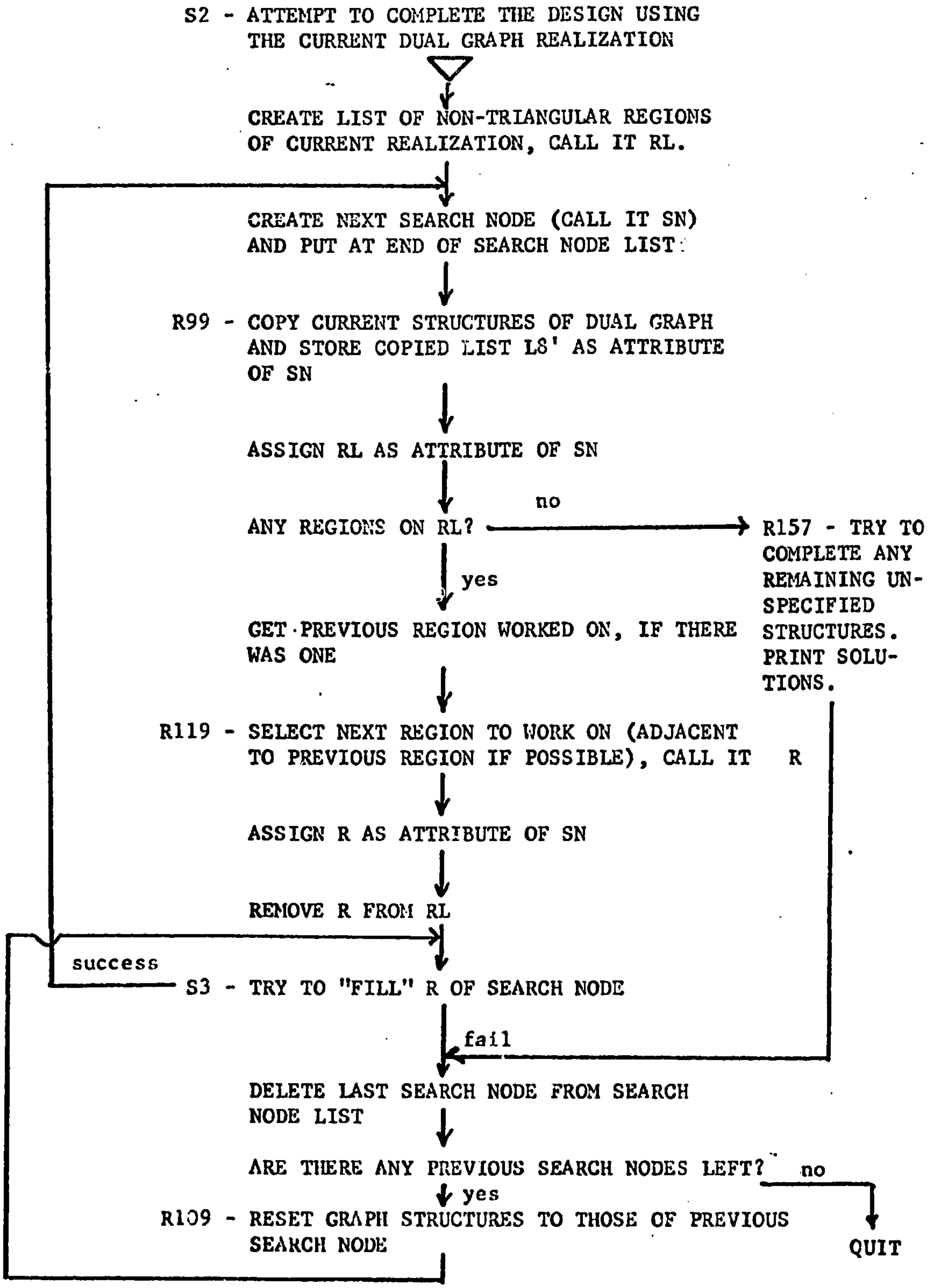

R79 - FORMAT FOR NODE OR REGION WELL-FORMEDNESS TEST
*KEY
C,D=COLORED, DIRECTED
C,O=COLORED
O,D=DIRECTED
O,O=NON-COLORED, NON-DIRECTED
LOOK AT ORDERED EDGE LIST (FOR NODE) OR COMPONENT LIST (FOR REGION) OF STRUCTURE. CREATE AUXILIARY LIST FROM THIS OF ALL C,O AND O,D EDGES FROM THE OEL, OR ALL NON C,D EDGES FROM THE CL*
A
none
GET NEXT EDGE ON AUXILIARY LIST
one
+48 - SELECT FIRST ALTERNATIVE COLOR-DIRECTION COMBINATION FOR THIS EDGE
IS STRUCTURE A NODE OR A REGION?
node
region
R77 - TEST IF NODE IS WELL-FORMED
success
success
R84 - TEST IF REGION IS WELL-FORMED
SET SUCCESS CELL POSITIVE
fail
fail
MARK CURRENT TEMPORARY ATTRIBUTES OF EDGES ON AUX. LIST SUCCESSFUL
aux. list empty
GET LAST EDGE ON AUX. LIST, CALL IT E
one
yes
DOES AUX. LIST HAVE ONLY ONE EDGE?
IS E MARKED PROCESSED?
yes
no
IS E MARKED PROCESSED?
yes
UNMARK E
+49 - SELECT NEXT C,D COMBINATION FOR E
FIND EDGE BEFORE E ON AUX. LIST, CALL IT E
last combination
not last combination
none
MARK EDGE
A
A
yes
IS SUCCESS CELL MARKED?
no
+50 - MAKE EDGE C AND D DEDUCTIONS
QUIT, SUCCESS
+63 - ERASE TEMPORARY EDGE ATTRIBUTES
QUIT, FAILURE
UNMARK EDGE

CHART F28



S2 - ATTEMPT TO COMPLETE THE DESIGN USING THE CURRENT DUAL GRAPH REALIZATION

CREATE LIST OF NON-TRIANGULAR REGIONS OF CURRENT REALIZATION, CALL IT RL.

CREATE NEXT SEARCH NODE (CALL IT SN) AND PUT AT END OF SEARCH NODE LIST.

R99 - COPY CURRENT STRUCTURES OF DUAL GRAPH AND STORE COPIED LIST L8' AS ATTRIBUTE OF SN

ASSIGN RL AS ATTRIBUTE OF SN

ANY REGIONS ON RL?

R157 - TRY TO COMPLETE ANY REMAINING UNSPECIFIED STRUCTURES. PRINT SOLUTIONS.

GET PREVIOUS REGION WORKED ON, IF THERE WAS ONE

R119 - SELECT NEXT REGION TO WORK ON (ADJACENT TO PREVIOUS REGION IF POSSIBLE), CALL IT R

ASSIGN R AS ATTRIBUTE OF SN

REMOVE R FROM RL

S3 - TRY TO "FILL" R OF SEARCH NODE

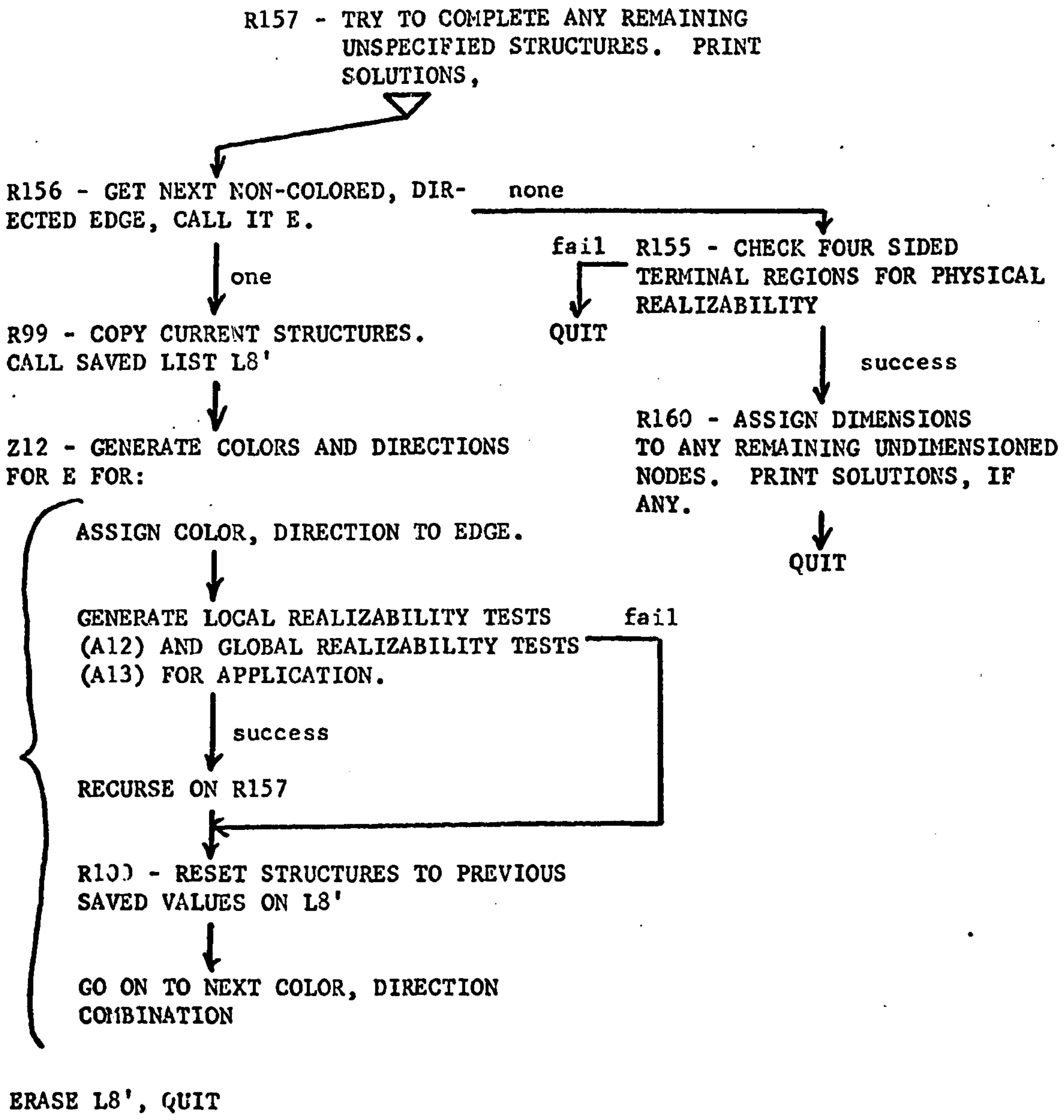
DELETE LAST SEARCH NODE FROM SEARCH NODE LIST

ARE THERE ANY PREVIOUS SEARCH NODES LEFT?

QUIT

R109 - RESET GRAPH STRUCTURES TO THOSE OF PREVIOUS SEARCH NODE

CHART F29



R157 - TRY TO COMPLETE ANY REMAINING UNSPECIFIED STRUCTURES. PRINT SOLUTIONS,

R156 - GET NEXT NON-COLORED, DIRECTED EDGE, CALL IT E.

R99 - COPY CURRENT STRUCTURES. CALL SAVED LIST L8'

Z12 - GENERATE COLORS AND DIRECTIONS FOR E FOR:

ASSIGN COLOR, DIRECTION TO EDGE.

GENERATE LOCAL REALIZABILITY TESTS (A12) AND GLOBAL REALIZABILITY TESTS (A13) FOR APPLICATION.

RECURSE ON R157

R100 - RESET STRUCTURES TO PREVIOUS SAVED VALUES ON L8'

GO ON TO NEXT COLOR, DIRECTION COMBINATION

ERASE L8', QUIT

R155 - CHECK FOUR SIDED TERMINAL REGIONS FOR PHYSICAL REALIZABILITY

R160 - ASSIGN DIMENSIONS TO ANY REMAINING UNDIMENSIONED NODES. PRINT SOLUTIONS, IF ANY.

CHART F30

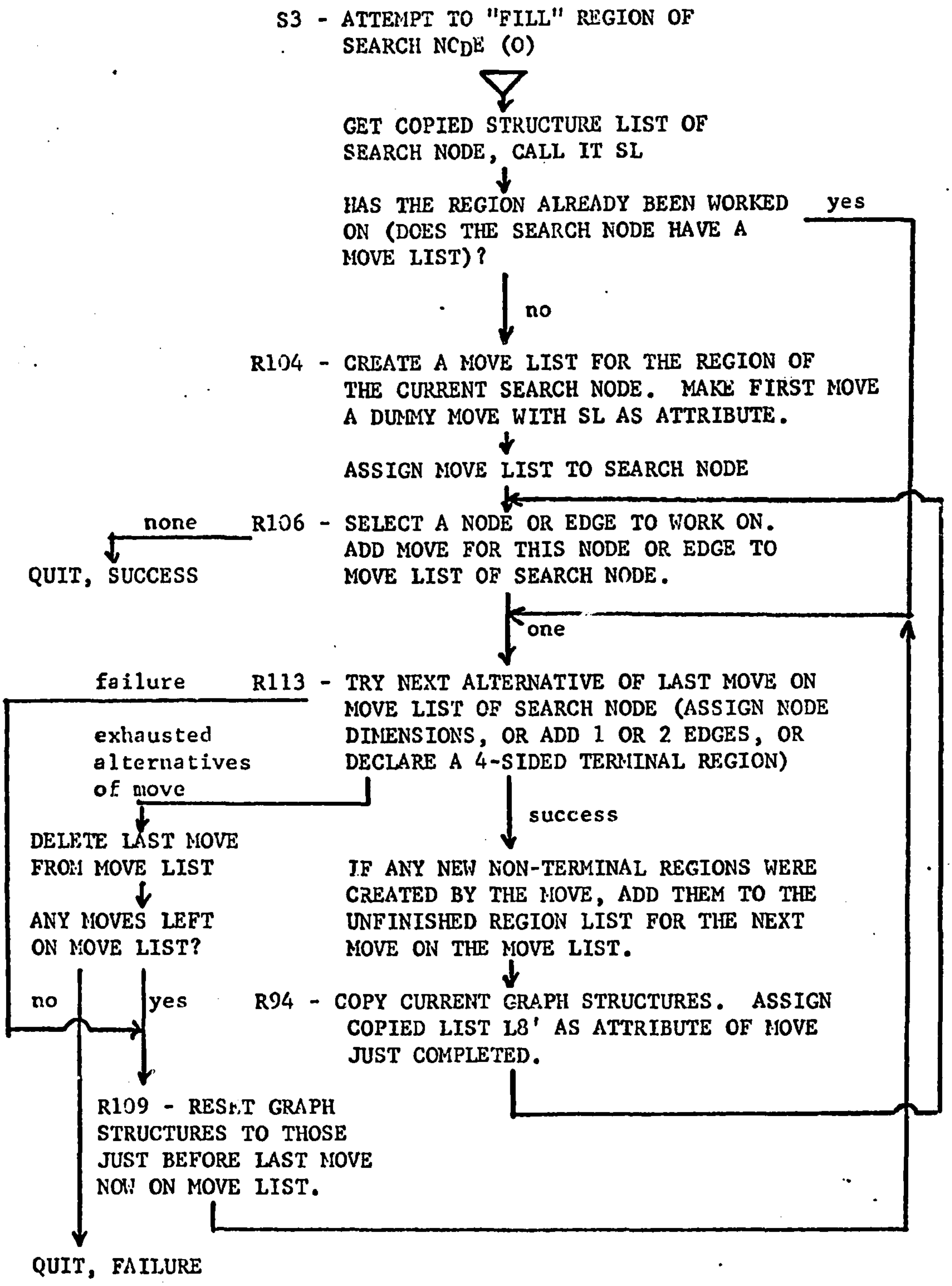

S3 - ATTEMPT TO "FILL" REGION OF SEARCH NCDE (O)
GET COPIED STRUCTURE LIST OF SEARCH NODE, CALL IT SL
HAS THE REGION ALREADY BEEN WORKED ON (DOES THE SEARCH NODE HAVE A MOVE LIST)?
yes
no
R104 - CREATE A MOVE LIST FOR THE REGION OF THE CURRENT SEARCH NODE. MAKE FIRST MOVE A DUMMY MOVE WITH SL AS ATTRIBUTE.
ASSIGN MOVE LIST TO SEARCH NODE
none
R106 - SELECT A NODE OR EDGE TO WORK ON. ADD MOVE FOR THIS NODE OR EDGE TO MOVE LIST OF SEARCH NODE.
QUIT, SUCCESS
one
failure
R113 - TRY NEXT ALTERNATIVE OF LAST MOVE ON MOVE LIST OF SEARCH NODE (ASSIGN NODE DIMENSIONS, OR ADD 1 OR 2 EDGES, OR DECLARE A 4-SIDED TERMINAL REGION)
exhausted alternatives of move
success
DELETE LAST MOVE FROM MOVE LIST
IF ANY NEW NON-TERMINAL REGIONS WERE CREATED BY THE MOVE, ADD THEM TO THE UNFINISHED REGION LIST FOR THE NEXT MOVE ON THE MOVE LIST.
ANY MOVES LEFT ON MOVE LIST?
no
yes
R94 - COPY CURRENT GRAPH STRUCTURES. ASSIGN COPIED LIST L8' AS ATTRIBUTE OF MOVE JUST COMPLETED.
R109 - RESET GRAPH STRUCTURES TO THOSE JUST BEFORE LAST MOVE NOW ON MOVE LIST.
QUIT, FAILURE

CHART F31

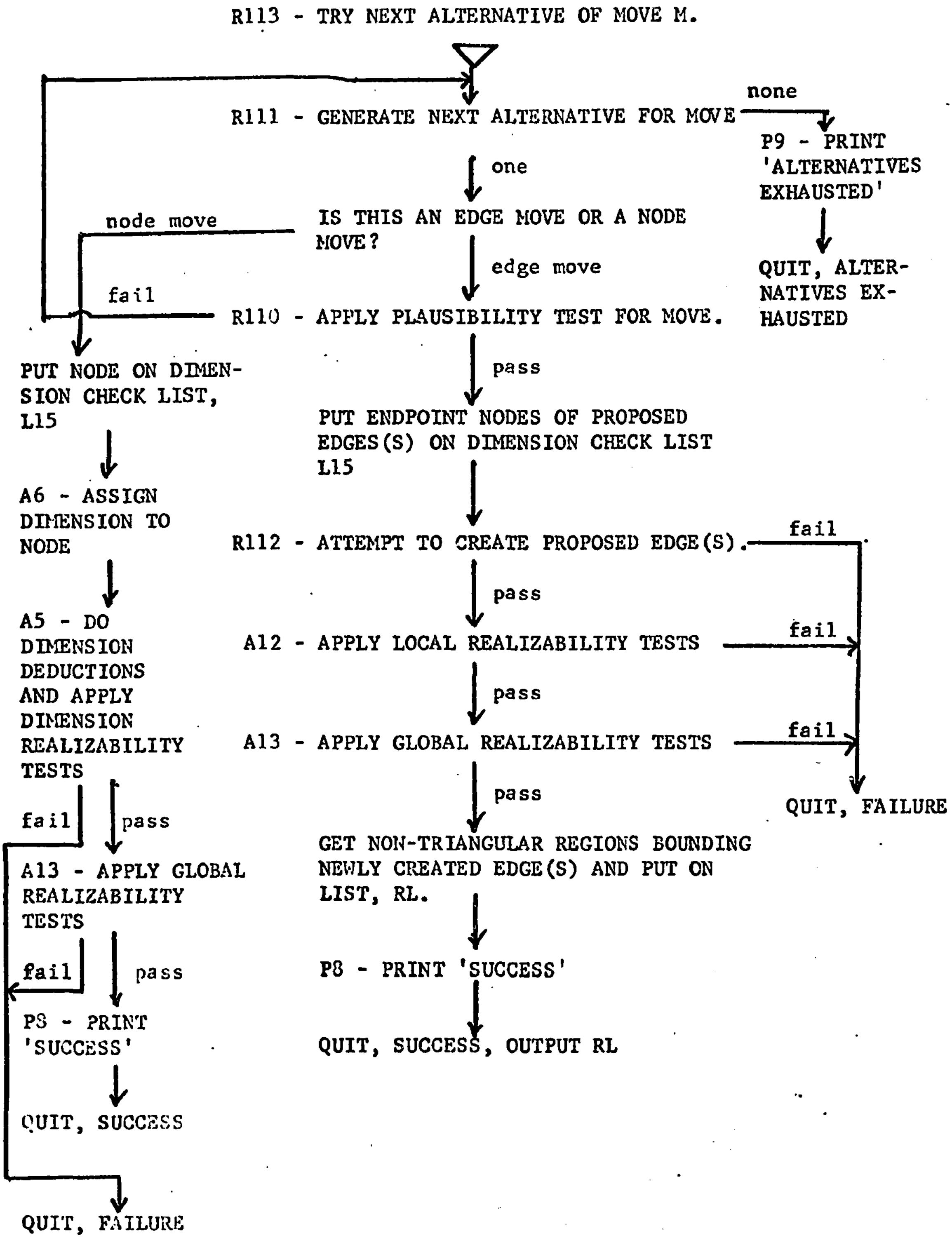
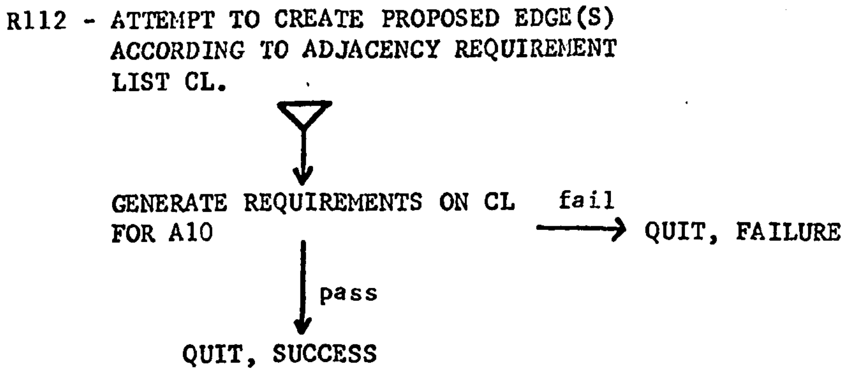
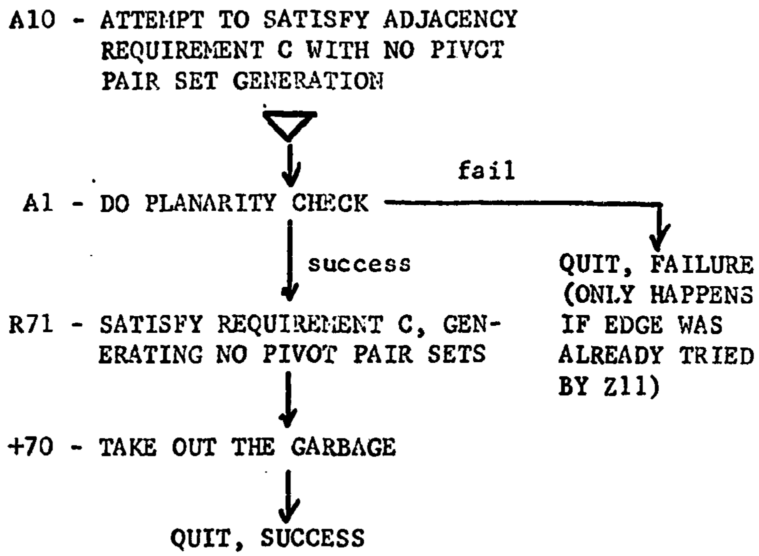

R113 - TRY NEXT ALTERNATIVE OF MOVE M.
R111 - GENERATE NEXT ALTERNATIVE FOR MOVE
none
P9 - PRINT 'ALTERNATIVES EXHAUSTED'
QUIT, ALTERNATIVES EXHAUSTED
one
IS THIS AN EDGE MOVE OR A NODE MOVE?
node move
edge move
fail
R110 - APPLY PLAUSIBILITY TEST FOR MOVE.
PUT NODE ON DIMENSION CHECK LIST, L15
pass
PUT ENDPOINT NODES OF PROPOSED EDGES(S) ON DIMENSION CHECK LIST L15
A6 - ASSIGN DIMENSION TO NODE
R112 - ATTEMPT TO CREATE PROPOSED EDGE(S).
fail
pass
A5 - DO DIMENSION DEDUCTIONS AND APPLY DIMENSION REALIZABILITY TESTS
A12 - APPLY LOCAL REALIZABILITY TESTS
fail
pass
A13 - APPLY GLOBAL REALIZABILITY TESTS
fail
pass
QUIT, FAILURE
fail
pass
A13 - APPLY GLOBAL REALIZABILITY TESTS
GET NON-TRIANGULAR REGIONS BOUNDING NEWLY CREATED EDGE(S) AND PUT ON LIST, RL.
fail
pass
P8 - PRINT 'SUCCESS'
P8 - PRINT 'SUCCESS'
QUIT, SUCCESS, OUTPUT RL
QUIT, SUCCESS
QUIT, FAILURE

CHART F32


R112 - ATTEMPT TO CREATE PROPOSED EDGE(S) ACCORDING TO ADJACENCY REQUIREMENT LIST CL.
GENERATE REQUIREMENTS ON CL FOR A10
fail
QUIT, FAILURE
pass
QUIT, SUCCESS


A10 - ATTEMPT TO SATISFY ADJACENCY REQUIREMENT C WITH NO PIVOT PAIR SET GENERATION
A1 - DO PLANARITY CHECK
fail
QUIT, FAILURE (ONLY HAPPENS IF EDGE WAS ALREADY TRIED BY Z11)
success
R71 - SATISFY REQUIREMENT C, GENERATING NO PIVOT PAIR SETS
+70 - TAKE OUT THE GARBAGE
QUIT, SUCCESS

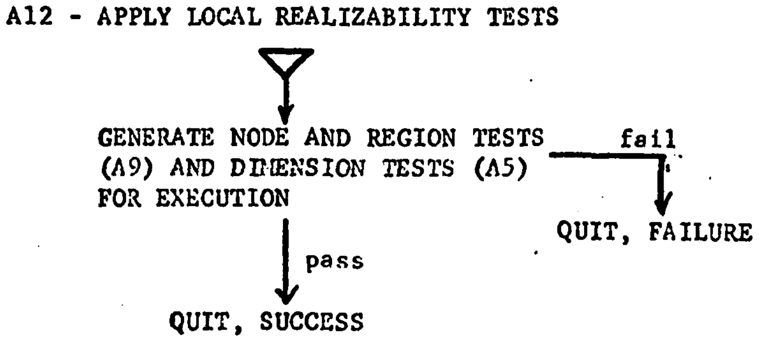

A12 - APPLY LOCAL REALIZABILITY TESTS
GENERATE NODE AND REGION TESTS (A9) AND DIMENSION TESTS (A5) FOR EXECUTION
fail
QUIT, FAILURE
pass
QUIT, SUCCESS

CHART F33

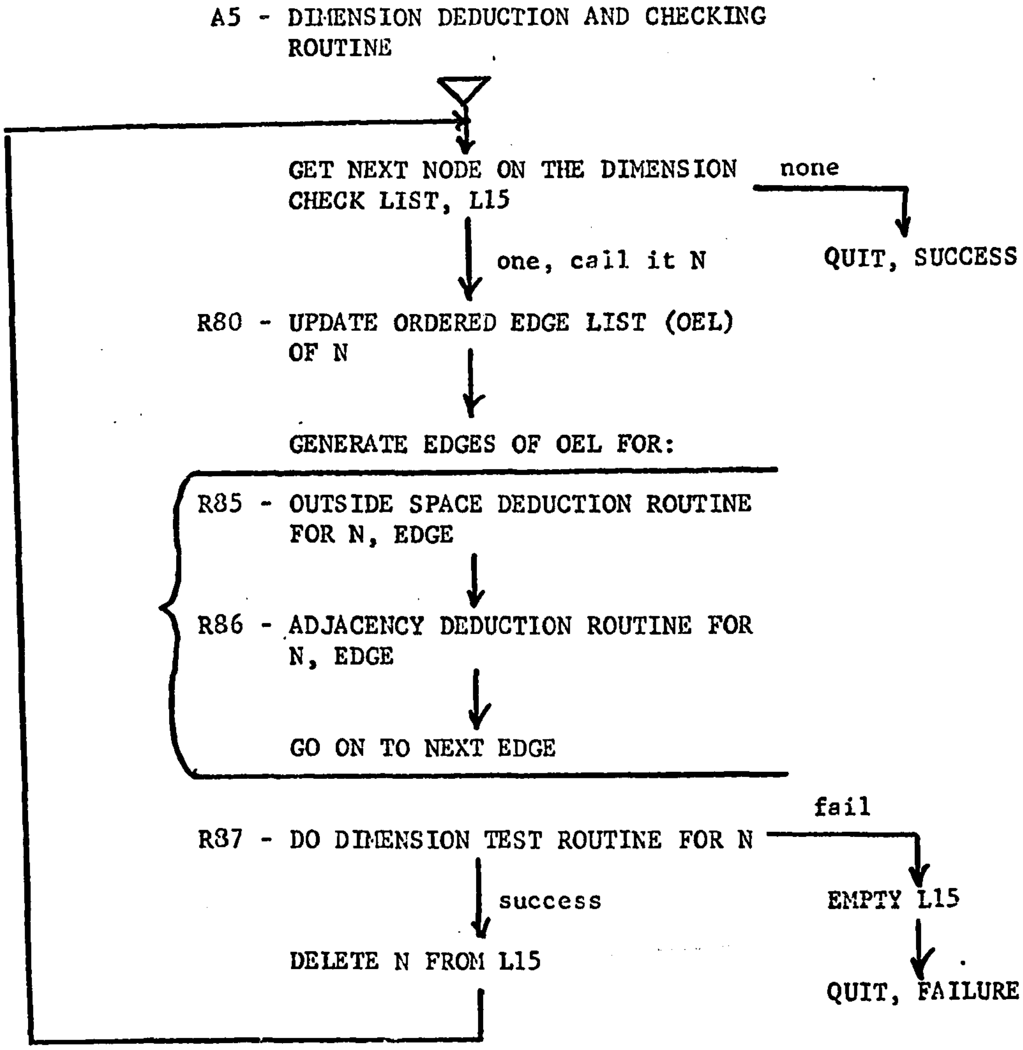
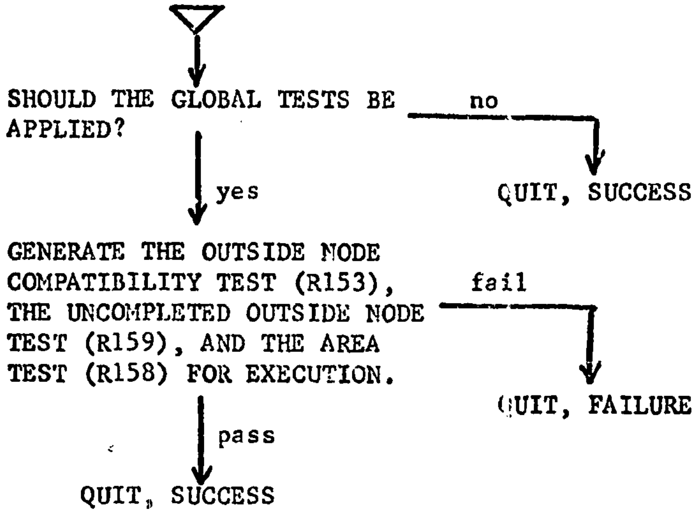

A5 - DIMENSION DEDUCTION AND CHECKING ROUTINE
GET NEXT NODE ON THE DIMENSION CHECK LIST, L15
none
QUIT, SUCCESS
one, call it N
R80 - UPDATE ORDERED EDGE LIST (OEL) OF N
GENERATE EDGES OF OEL FOR:
R85 - OUTSIDE SPACE DEDUCTION ROUTINE FOR N, EDGE
R86 - ADJACENCY DEDUCTION ROUTINE FOR N, EDGE
GO ON TO NEXT EDGE
R87 - DO DIMENSION TEST ROUTINE FOR N
fail
EMPTY L15
QUIT, FAILURE
success
DELETE N FROM L15

CHART F34

A13 - THE GLOBAL REALIZABILITY CHECKS


SHOULD THE GLOBAL TESTS BE APPLIED?
no
QUIT, SUCCESS
yes
GENERATE THE OUTSIDE NODE COMPATIBILITY TEST (R153), THE UNCOMPLETED OUTSIDE NODE TEST (R159), AND THE AREA TEST (R158) FOR EXECUTION.
fail
QUIT, FAILURE
pass
QUIT, SUCCESS

REFERENCES

1. Alexander, Christopher and Poyner, Barry. The Atoms of Environmental Structure. Center for Planning and Development Research, University of California, Berkely, July 1966.

2. Alexander, Christopher. HIDECS 3: Four Computer Programs for the Hierarchical Decomposition of Systems Which Have an Associated Linear Graph. Research Report R63-27, Department of Civil Engineering, Civil Engineering Systems Laboratory, M.I.T., Cambridge, Massachusetts.

3. ____________________. Notes on the Synthesis of Form. Harvard University Press, Cambridge, Massachusetts, 1964.

4. ____________________. "Relational Complexes in Architecture." Architectural Record (September 1966), 185.

5. ____________________. The Coordination of the Urban Rule System. Center for Planning and Development Research, University of California, Berkeley, July, 1966.

6. Armour, Gordon C. and Buffa, Elwood S. "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities. Management Science. Vol. 9, No. 1 (January 1963), 294-309.

7. Auslander, L. and Trent, H.M. "On the Realization of a Linear Graph Given Its Algebraic Specification". The Journal of the Acoustical Society of America, Vol. 33, No. 9, (September 1961), 1183-1192.

8. Berge, Claude. The Theory of Graphs and Its Applications. John Wiley and Sons, Inc., New York, 1962.

9. Buffa, Elwood S., Armour, Gordon C,, and Vollmann, Thomas E. "Allocating Facilities with 'CRAFT'." Harvard Business Review, (March-April 1964).

10. Busacker, R.G. and Saaty, T.L. Finite Graphs and Networks - An Introduction with Applications. McGraw-Hill, 1965.

11. Casalaina, V. and Rittel, H. Morphologies of Floor Plans. Paper for the Conference on Computer-Aided Building Design, 1967.

12. Eastman, Charles M. The Identification of Architectural Design Criteria by Computer: An Exploratory Study. Preliminary draft, Carnegie-Mellon University, Department of Computer Science, August 18, 1967.

13. Grason, John. . Some Ideas Concerning an Automatic Procedure for Component Layout and Wire Routing in Integrated Circuit Design, working paper, September 1967.

14. ____________. The Synthesis Process in Design - A Study via Computer Implementation. Complex Information Processing Paper #99, Carnegie Institute of Technology, April 7, 1967.

15. Hershdorfer, Alan M. Computer Systems for Building Design. Fourth National Conference on Building Design.

16. Krejcirik, M. "Computer-Aided Plant Layout". Computer-Aided Design (Autumn 1969), 7-19.

17. Lee, R.C. and Moore, J.M. "CORELAP - Computerized Relationship Layout Planning". Industrial Engineering. Vol. 18, No. 3 (March 1967).

18. Lee, Kaiman. COMSBUL - Computerized Multi-Story Building Layout. For information on this computer program contact Kaiman Lee, 1 Rosa St.,Hyde Park, Massachusetts 02136.

19. Levin, P.H. "Use of Graphs to Decide the Optimum Layout of Buildings". Architects' Journal, (October 7, 1964).

20. Manheim, Marvin L. Hierarchical Structure: A Model of Design and Planning Processes. M.I.T. Report #7, M.I.T. Press, Cambridge, 1966 - taken from research report No. R64-15, May 1964.

21. Moran, Thomas P. On the Dimensioning of Planar Configurations. Paper done in Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, May 1969.

22. Newell, Allen, and Simon, Herbert A. "GPS, A Program That Simulates Human Thought". Computers and Thought, ed. Edward A. Feigenbaum and Julian Feldman, Part 2, Section 1, 279-293. McGraw-Hill, 1963.

23. Parsons, David. "Planning by the Numbers" in "Performance Design". Progressive Architecture. (August 1967), 111.

24. Simon, Herbert A. The Sciences of the Artificial. M.I.T. Press, 1969.

25. Tutte, William T. "Squaring the Square". The Second Scientific American Book of Mathematical Puzzles and Diversions, Simon and Schuster, New York, 1961, 186.

26. Whitehead, B. and Eldars, M.Z. "An Approach to the Optimum Layout of Single-Storey Buildings". Architects' Journal (June 17, 1964).