"SPACE AND FUNCTION ANALYSIS"

"A Computer System for the generation of functional layouts in the S.A.R. Methodology".

by

Alfonso Govela

B. Arch. Universidad Iberoamericana,

Mexico D.F., 1974

submitted in partial fulfillment
of the requirements for the degree of
Master of Architecture
in
Advanced Studies.

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1977.

Signature of Author ______________________________
Department of Architecture, February 1977

Certified by ______________________________
Prof. Nicholas Negroponte, Associate Professor
Thesis Supervisor.

Accepted by ______________________________
Chairman, Departmental Committee for
Graduate Students.

ACKNOWLEDGEMENTS

I thank my Thesis Advisor, Associate Professor Nicholas Negroponte, for presenting, in a Summer Session back in 1973, the possibilities of computer applications in Architecture, and for providing later on the opportunity and the support to work with him and to develop this project.

I thank John N. Habraken, Head of the Department, for the patient sessions where he explained in simple terms the basic concepts of the S.A.R. Methodology that appear here in a convoluted way.

I thank Mike Gerzso for the discussions we have had on Architecture and Computers, and for starting, through his work, my interest in Production Systems.

I thank Seth Steinberg for his constant help in conceptual issues, programming techniques and practical matters.

The work reported herein has been sponsored by CONACYT, Consejo Nacional de Ciencia y Tecnologia, Mexico, D.F., Mexico.

Alfonso Govela
February 1977.

TABLE OF CONTENTS

Abstract

"SPACE AND FUNCTION ANALYSIS"

"A Computer System for the generation of
functional layouts in the S.A.R. Methodology".

Alfonso Govela

Submitted to the Department of Architecture on February 1977 in partial fulfillment of the requirements for the degree of Master of Architecture in Advanced Studies.

As part of the S.A.R. Methodology, a set of computer programs has been implemented to carry on the systematic generation of all the possible functional layouts for a given design criteria.

They are intended to help the designer analyze and evaluate the relationships between a space and its function, and display the consequences of different design standards on different sizes and layouts of spaces.

The main assumption is that a space can be analyzed functionally by looking at characteristic arrangements of furniture or equipment, that correspond to a certain function.

A function can be defined in terms of the location, dimensions and relations between furniture elements and spaces.

A set of design standards describe a spatial system and constraint a solution space where particular layouts can be effectively, and if necessary, exhaustively explored by a procedure that generates as many arrangements as desired.

This generative capability is aimed to help in the development and evaluation of standards for spatial performance. By studying the different layouts that each set of standards permits, different evaluation techniques can be defined to compare, select and agree on the most adequate criteria for an actual situation or an hypothetical case.

Thesis Supervisor: Nicholas Negroponte

Title: Associate Professor of Architecture.

0. Introduction:

This Thesis is concerned with the formulation of architectural functions and the analysis of their spatial consequences. It has originated through several interests and it reflects this in the different sections. On one hand it comes from an interest in spatial design and design methodologies, on the other it has grown from an interest in computer applications and generative techniques, as analytical tools within the framework of design.

The main problem that it approaches is the first logical operation in the process of designing 'supports' in the S.A.R. Methodology. It attempts to provide a systematic way to define spatial functions and spatial characteristics, and present a procedure that enumerates all the possible relations between a function and a space.

Its conceptual basis come from two different fields, a participatory design discipline originated as an alternative solution to mass housing problems, and the models that have become standard practice in the computer field of Artificial Intelligence. The first provided a structured view of the problems in design on the basis of which functional standards are described, while the second, through 'Problem-Solving' and 'Tree-Searching' techniques, provided a way of generating the different functional layouts that these standards permit.

Although the first represents new ideas in architecture, the second, we might say, is an idea that has been around for sometime. However, its application in this case is different in an important sense. Prcblem-Solving has been oriented to the representation of procedures that find solutions by searching through a range of possibilities according to some given rules. A procedure that 'designs' or finds design solutions in this sense, besides being far away in terms of our knowledge of design, would miss the issue of values implicit in design. What matters is not only the solution, but the definition of the problem and the process to reach this result, as values are defined constantly in each decision taken.

In design, it is more relevant the problem of evaluation. Not how we get solutions but how do we evaluate their consequences. To analyze decisions, we have to evaluate their consequences and we have to enumerate their results, through the enumeration of results we can compare one decision against another and select the one we consider more adequate. Searching and enumeration are in principle equivalents, and it is in the context of enumeration that the techniques of Problem-Solving are used in this Thesis. At the scale of function analysis, it is not another approach to use the computer as designer, but an attempt to extend a small part of a methodology that recognizes the problem of values, to the scale of a room, and provide a generative method that can be used as its main analytical operation.

In the first chapter the problem area is defined with our main assumptions. Following it, there is a general presentation of the S.A.R. principles and an outline of the Problem-Solving representations and search techniques. In the third chapter the principles of both S.A.R. and Artificial Intelligence are applied and extended to the SPACE and FUNCTION ANALYSIS, and in the fourth an outline of the computer system is presented.

1.- PROBLEM DEFINITION AND ASSUMPTIONS.

1.1.- Background:

Spaces in human environments, must frequently have a purpose, they exist as containers for human activities. Within them, actions of different kinds are continuously performed in a multitude of ways. The importance of their purpose, shows in fact, in our identifying them, in everyday language, with names that refer to the action or actions that can be realized within its limits.

This series of activities constitute what we call, or think as, the 'use' that is made of a space, or the 'function' that has beer assigned to it. Using a space or carrying on a series of activities can result from a careful planning process at the time when the space is created, or simply result from a spontaneous adaptation, at a later point in time, to a function that was neither considered in its creation, nor planed to be contained by it. In either case, the success or failure of this 'use' depends in the relations between different characteristics of the space, such as shape, proportions or dimensions, to name a few, and the kind of function that can be assigned to it. Spatial characteristics permit or prevent, sometimes in a definite strong way, the performance of certain functions.

In the process of design, the assignment of spatial types to activities or uses, is one of the initial, if not the first step taken in the generation of spatial solutions. These types are almost always roughly set up at the beginning of the process and their definition fluctuates until the very end, according to other, often more global, circumstances in the design.

Must of the time, assumptions already exist in the form of cultural preferences or in the form of standards that delimit the range of possible spaces corresponding to a function. These norms or personal preferences, set the acceptable characteristics for a space, but very seldom provide a framework for understanding the reasons behind their existence, or the implications when a change in their definition is made. By being unaware of their rationale, we sometimes fail to comprehend the relations between the two, and consequently fail to understand what the impact of different spatial solutions might be on different uses.

The problem of relating spaces to functions might be considered a trivial problem without any need for explanations, or simply a matter of design experience where no further systematization or exploration is needed.

At a practical level, however, there has been an increasing interest during the last years, for 'performance' studies of different kinds. The economic analysis of buildings has being shifted from the more 'solid' actual cost of the building construction, to the more 'softer' interest in the builoing use over its whole life perioc. The proportion of costs between the initial construction expenses (%) and the maintenance bills (%) indicates areas where savings can become more substantial, and has pointed out the importance of understanding how spaces are used.(1)

At a deeper, more significant level, on the other hand, there has been an increasing questioning of values and assumptions behind design solutions, as it becomes aparent that desigr problems are not well defined technical situations with clearcut solutions, but difficult problems which solutions represent implicit set of values, and which values must be agreed upon before attempting any technical implementation.

As Rittel (2), has pointed out quite correctly, the increasing critique on professional work stems from challenging the tests for efficiency, by renewed preocupation with their consequences for equity. The professional's job, "....once seen as solving an assortment of problems that appeared to be definable, understandable and consensual...", is being confronted now with the fact that "...the seeming consensus .... is beirg eroded by .... differentiation of values...". "...There seems to be a growing realization that a weak strut in the professional's support system lies at the juncture where goal-formulation, problem-definition, and equity issues meet...".

How to provide a basis for the first kind of spatial analysis, and how to coordinate the formulation and evaluation of standards or values at this second level, are the main two issues which motivated, although -of course- they were not solved, the work done in this

thesis.

1.2.- Problem definition:

The idea behind SPACE and FUNCTION analysis was to develop, along the lines of the S.A.R. methodology (3), a systematic way of figuring out the relations between a space and its function or its set of possible functions.

Its main objectives were to understand what makes a space adequate for a certain use, when can we realistically assign activities to a given space, or what are the consequences of changing spatial characteristics or functional requirements.

How to formulate functions, describe spaces, and analyze the relations between these two, are the main parts of the thesis work. How to find out, for a given function, one or all the spaces where it can be contained in a satisfactory way; or for a given space, how to find out, one or all the functions that can be contained by it, are the particular questions that we would attempt to answer.

1.3.- Assumptions:

A design problem in itself, the development of this SPACE and FUNCTION analysis has been approached with certain assumptions in mind:

- Independently of defending, rationalizing or explaining where personal values for a function come from, it was considered more relevant to find out how can functions be expressed in a way that helps us understand their spatial implications.

- It was assumed that a function can be defined in terms of the pieces of furniture or equipment that are needed to carry on a certain activity. To talk about the use of a space, we can define then the series of elements that should be contained by it, together with their positions in space and the possible relations beween different pieces.

- Functions formulated in these terms, represent a set of standards or value systems which implicitly define a range of 'acceptable' uses that a space can have. An arrangement of elements that correspond to this functional definition is called a 'functional layout', and it stands for one of the possible valid distributions of elements in space.

- To evaluate the consequences of different standards, we have to look at all the possible arrangements that are implicitly permited by them. Only by knowing what variations in functional layouts are possible, we can analyze, understand or refutate, in terms of their implications, design values in design solutions.

- Functional definitions are, therefore, not attempted as definite laws or patterns, but rather as statements open to refutation, as implications are evaluated through the enumeration of their possible consequences.

The formulation of functional standards and the enumeration of their possible layouts are the main assumptions behind SPACE and FUNCTION analysis. The methodological basis for these two parts are explored in the next chapter, its final form in chapter three, and the implementation of a generative computer system in chapter four.

Chapter One,

Notes:

(1) "Life-Cycle costing in the Public Buildings Service",
G.S.A. General Services Administration,
Public Building Service,
Study made by Booz-Allen & Hamilton Inc. for G.S.A.,P.B.S.

(2) RITTEL HORST, "Dilemmas in a General Theory of Planning",
Working Paper 194, University of California,
Berkeley, Cal. Nov. 1972.

(3) BOEKHOLT J.TH., THIJSSEN A.P., DINJENS P.J.M., HABRAKEN N.J.
"Variations: The Systematic Design of Supports",
forthcomming M.I.T. Press. Cambridge Mass. 1976.

2.- METHODOLOGICAL BASIS AND ENUMERATION TECHNIQUES

Standaros or norms must be formulated in a way that permits the evaluation of their conseauences. They should be described, at least, in a way that allows testing particular layouts; but, more important, and not so obvious, they should be structured in a way that permits the systematic enumeration of these layouts.

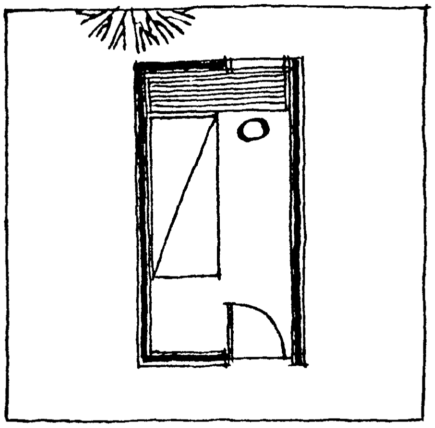
That standards must be sufficiently defined to permit the evaluation of layouts, is a clear point and can be solved in different ways. That their formulation must permit the exaustive enumeration of all possible layouts if necessary, might be clear from the standpoint of understanding their implications, but it is not so transparent how it can be implemented.

In this chapter two main ideas are revised. First the general principles of the S.A.R. <Stitching Architecten Research> Group (1), are presented as a methodological basis that, having solved this formulation problem at the scale of housing, can be applied -or rather, reduced- to the scale of space and function analysis; and second, the general principles of Problem-Solving and Enumeration techniques are presented as a framework that can be used in the actual generation of layouts.

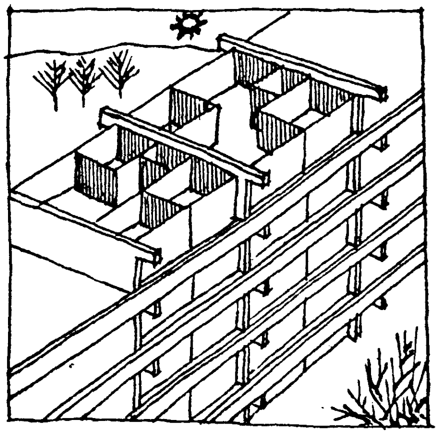
2.1.- S.A.R. formulation of Design Problems:

2.1.1.- Parts and Relations:

In the S.A.R. methoaology, design problems can be formulated in terms of "...an environment and elements that have to be placed in that environment..."(2). There is always a site, environment or context where different elements can be positioned, and the stanoards that define a set of values in design solution can always be expressed in terms of that site, its elements and their positions.

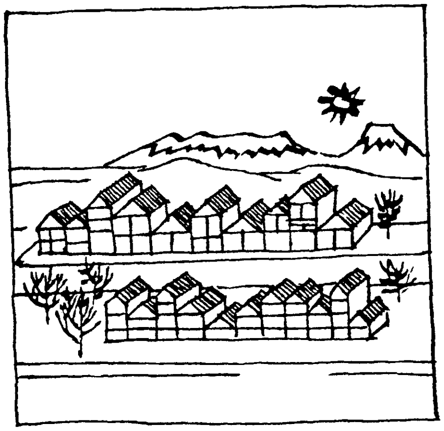


Depending on the scale we work, the site can vary from the spaces in city block, to one area in room space, and accordingly, the elements that are positioned in them can range from the pieces of furniture needed to perform a function to the support structures where people dwell in an urban environment.

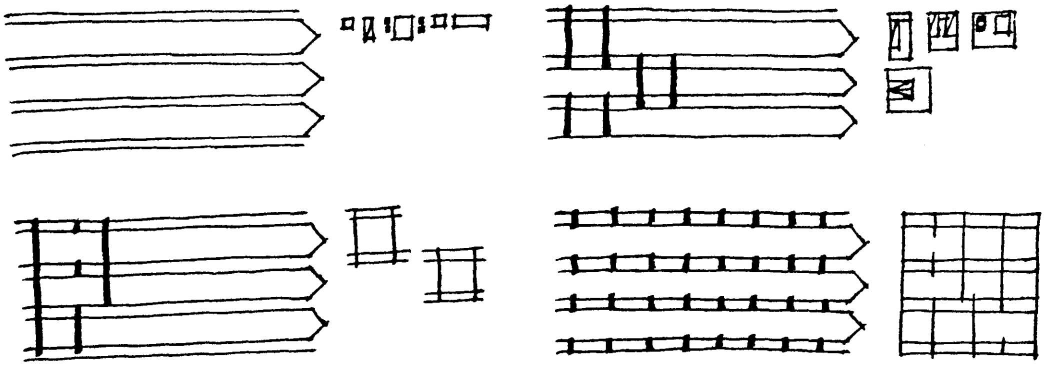


Sites can appear in design problems as given situations, that is contexts or constrains that exist already and that are external to the designer actions, or they can be defined during the design process. They can represent one specific situation, as might be the case of a site as an urban block, where infrastructure, size, dimensions, and surrounding buildings are established and well defined; or they can be used to describe multiple situations, and stand then as general schemes or models for several instances of similar sites.

The elements that are positioned in a site can



be defined by the designer or selected from some range of possible options. In either case, to formulate standards that relate elements and site, elements have to be described with sufficient exactitude. Type and number of elements, shape and dimensions, are the basic information



that would be required for any description of them.

Among these elements there are certain relations that should have to be defined. Elements can relate to each other in different ways, that is in terms of their being adjacent, near to each other, separated from, contained by, etc.. Their relations result from program requirements that should be present in the design solution, and as part of these requirements they should also be described precisely.

A "well defined" designed problem in the methodology, is formed then by the following parts and relations:

1. A description of the environment.

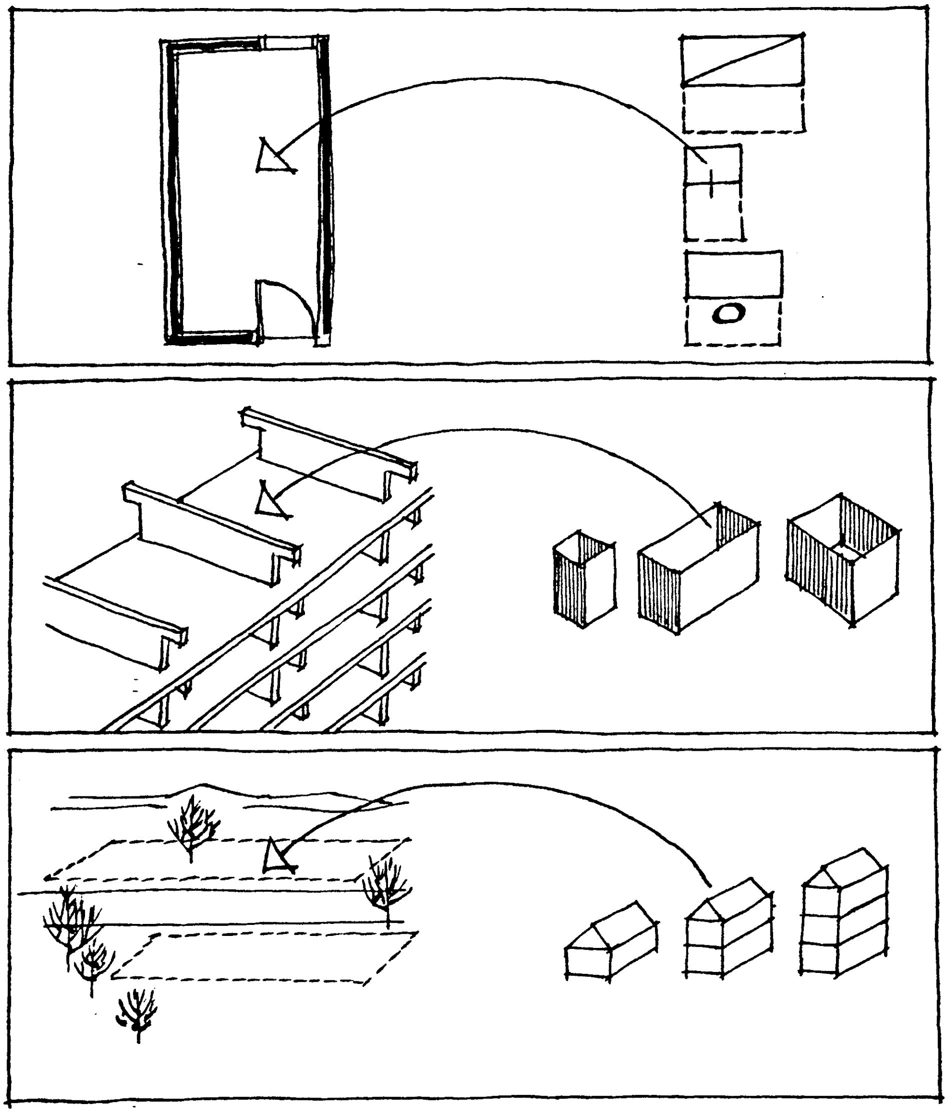
2. A defined set of elements that could be used in the environment.

3. Data about the location of elements relative to one another.

4. Data about the location ot the elements in the environment.

2.1.2.- Levels of definition:

The formulation of Site, Elements, Relations and Positions, can be applied at different levels and at different scales of the design problem. As said before,



the Site can be an urban block, or a space in a room, and the elements can be housing structures, or furniture pieces.

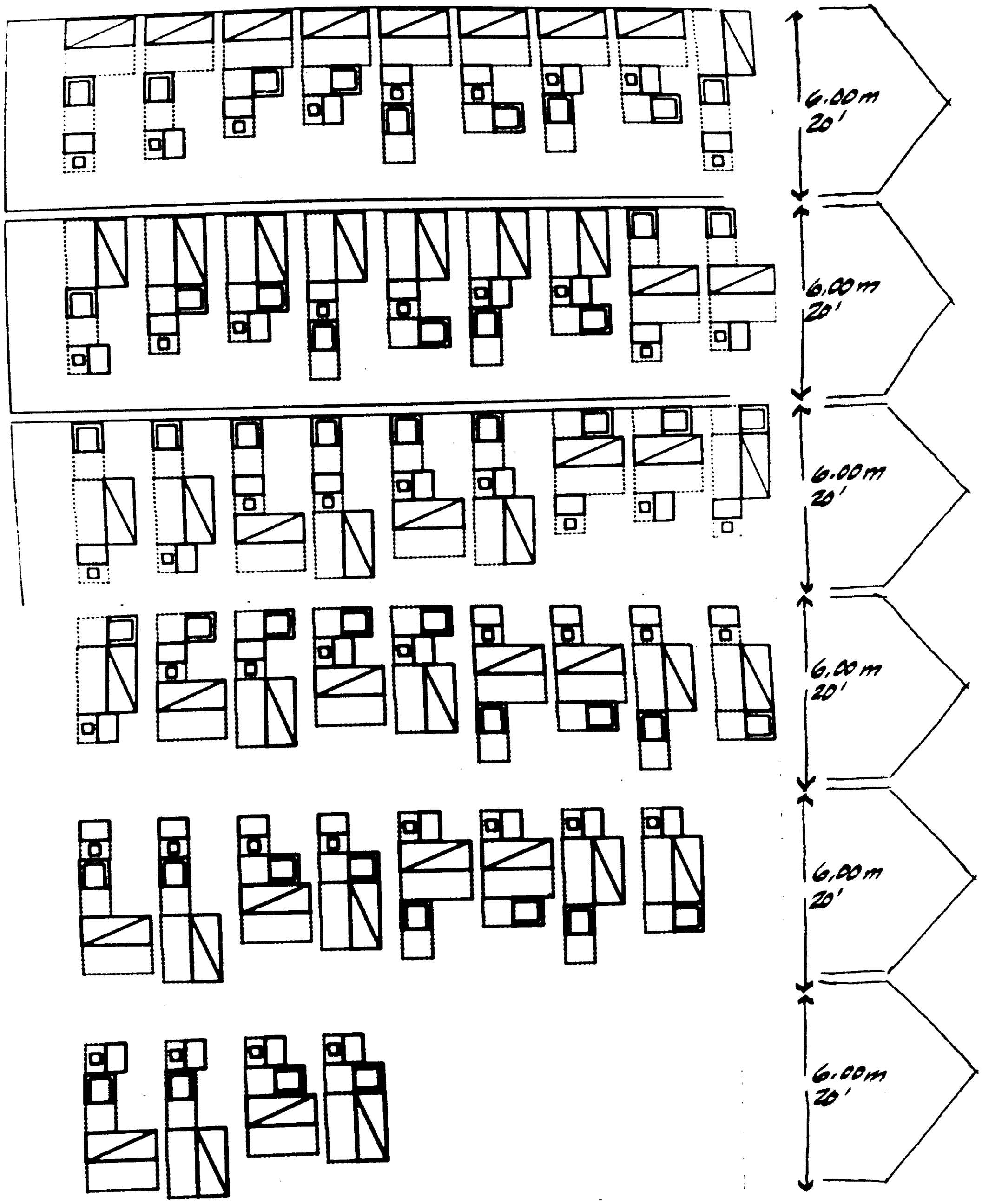
It is characteristic in this formulation, that the Site at one scale, becomes an Element at the next level up, and its elements consequently, stand as Sites for the next formulation one level down. The variations of element layouts at one scale become then, together with their positions in their site, one of the elements that can be used in the definition of standards one level up. A room can be the Site for furniture layouts, and together with one possible functional layout be an element for a dwelling site. A dwelling with all its rooms becomes then an element in the building site, and buildings become elements in the urban block.

2.1.3.- Operations of Analysis:

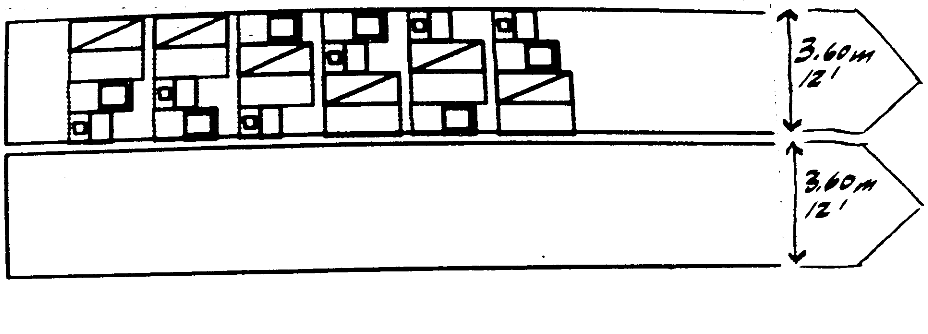
The evaluation of standards at any level of this hierarchy, is performed by generating all the possible layouts for a given site at a given level.

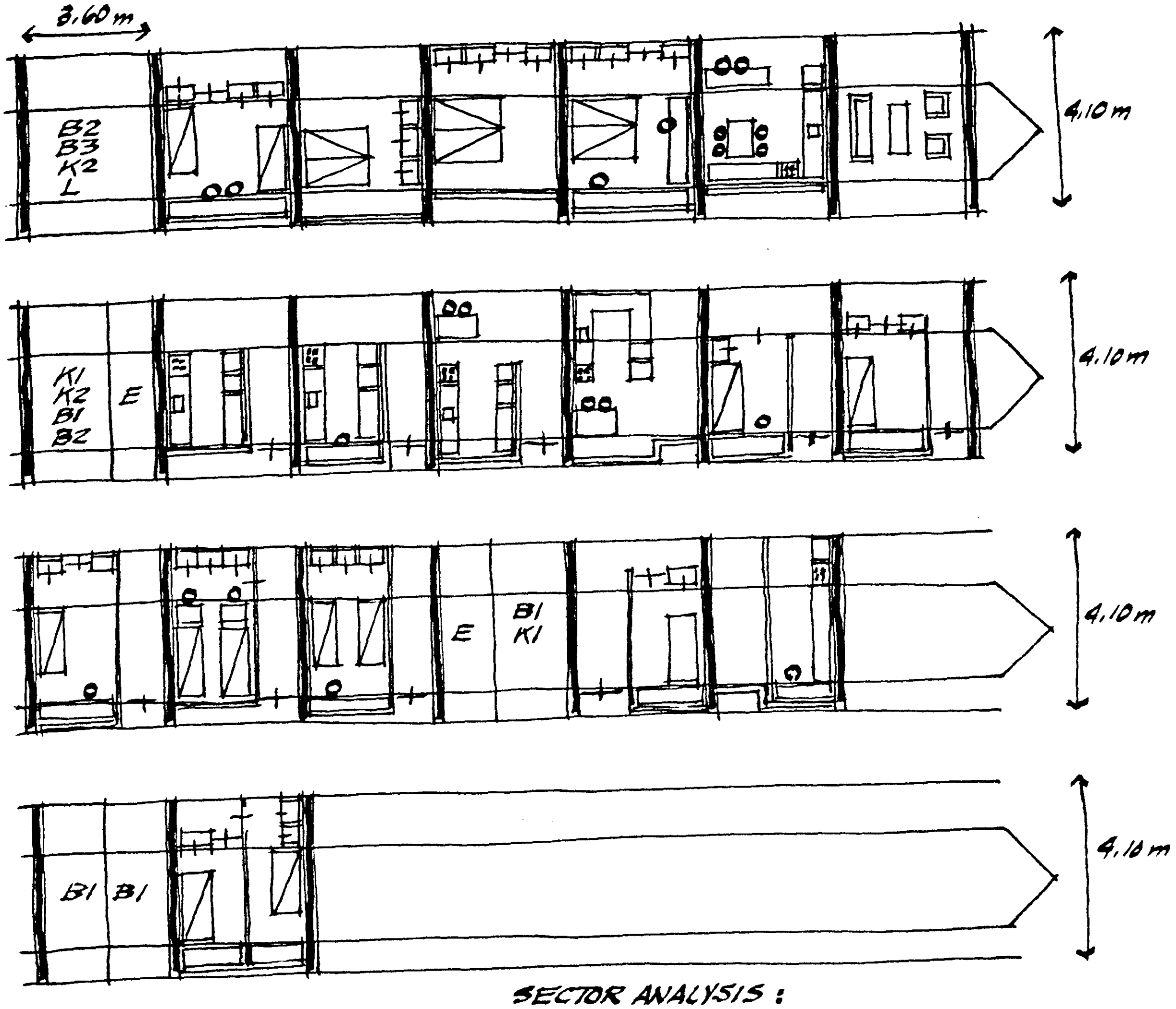
We start with the smaller layer in the structure, explore its alternatives and if satisfied, agree to select some basic layout variations as elements for the next level up. In there we generate again all the possible variations among which, in turn, some are selected to pop up as elements to the next element in the levels, until we reach the layouts of the final layer.

To continue the example in the illustrations above the possible room layouts, or the uses that can be

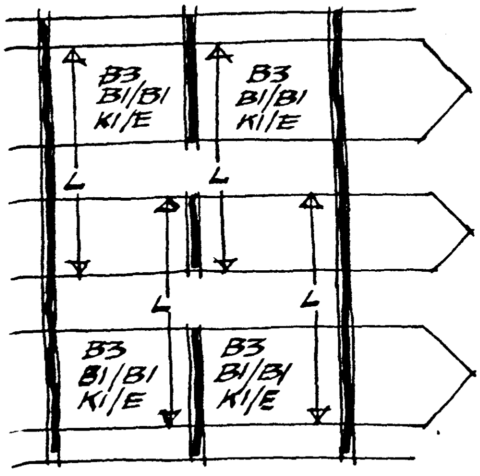


SPACE AND FUNCTION ANALYSIS :

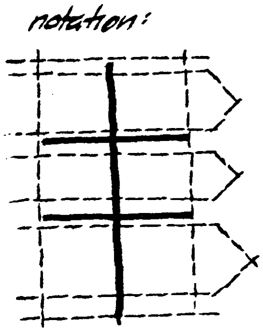


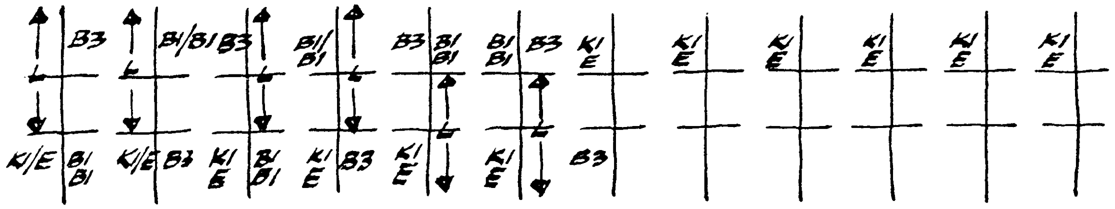


SECTOR ANALYSIS :

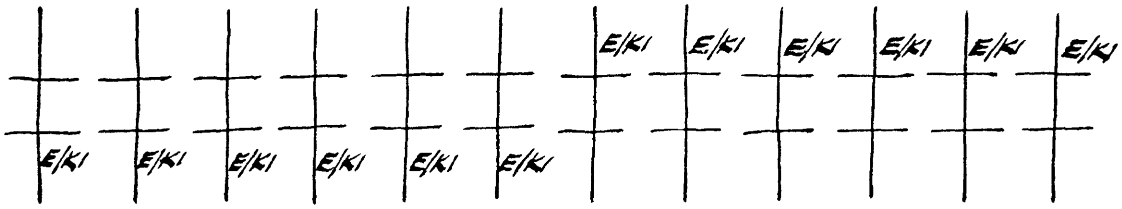


BASIC VARIATIONS :





BASIC VARIATIONS CONT.:



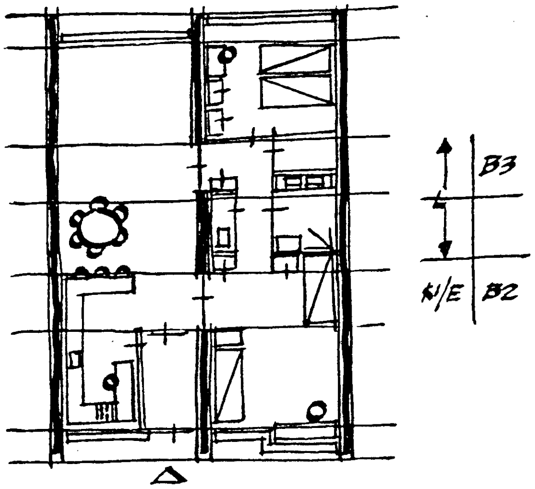
assigned to a room of 3.60 x 4.10 m would be: (SECTOR ANALYSIS)

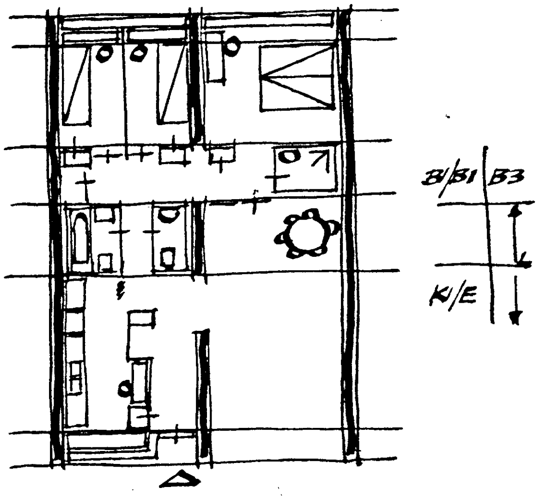
And the variations of room arrangements in an apartment floor, would be generically: (BASIC VARIATIONS)

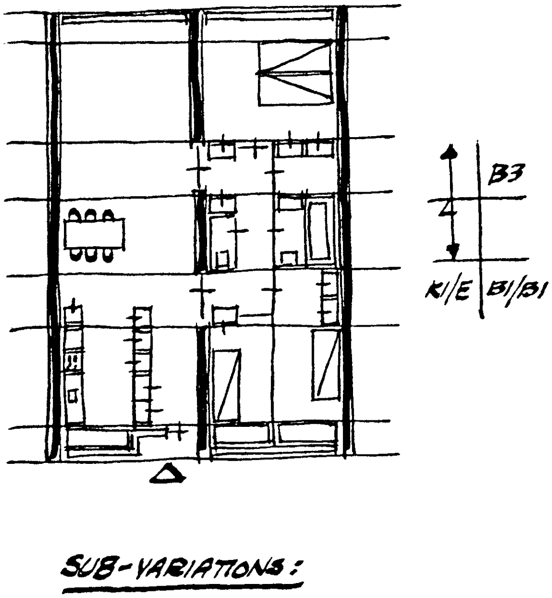
some of which could result in the possible floor plans, (SUB-VARIATIONS)

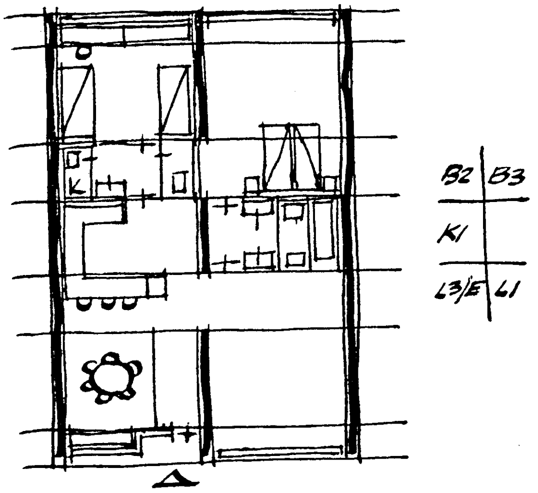
and which could be positioned, similarly in an urban block as: (URBAN TISSUE)

This formulation structures a design problem in a way that permits the systematic enumeration of layouts. In chapter 3 these general principles will be applied to the scale of SPACE and FUNCTION ANALYSIS, and it will be shown through an example how is the enumeration carried

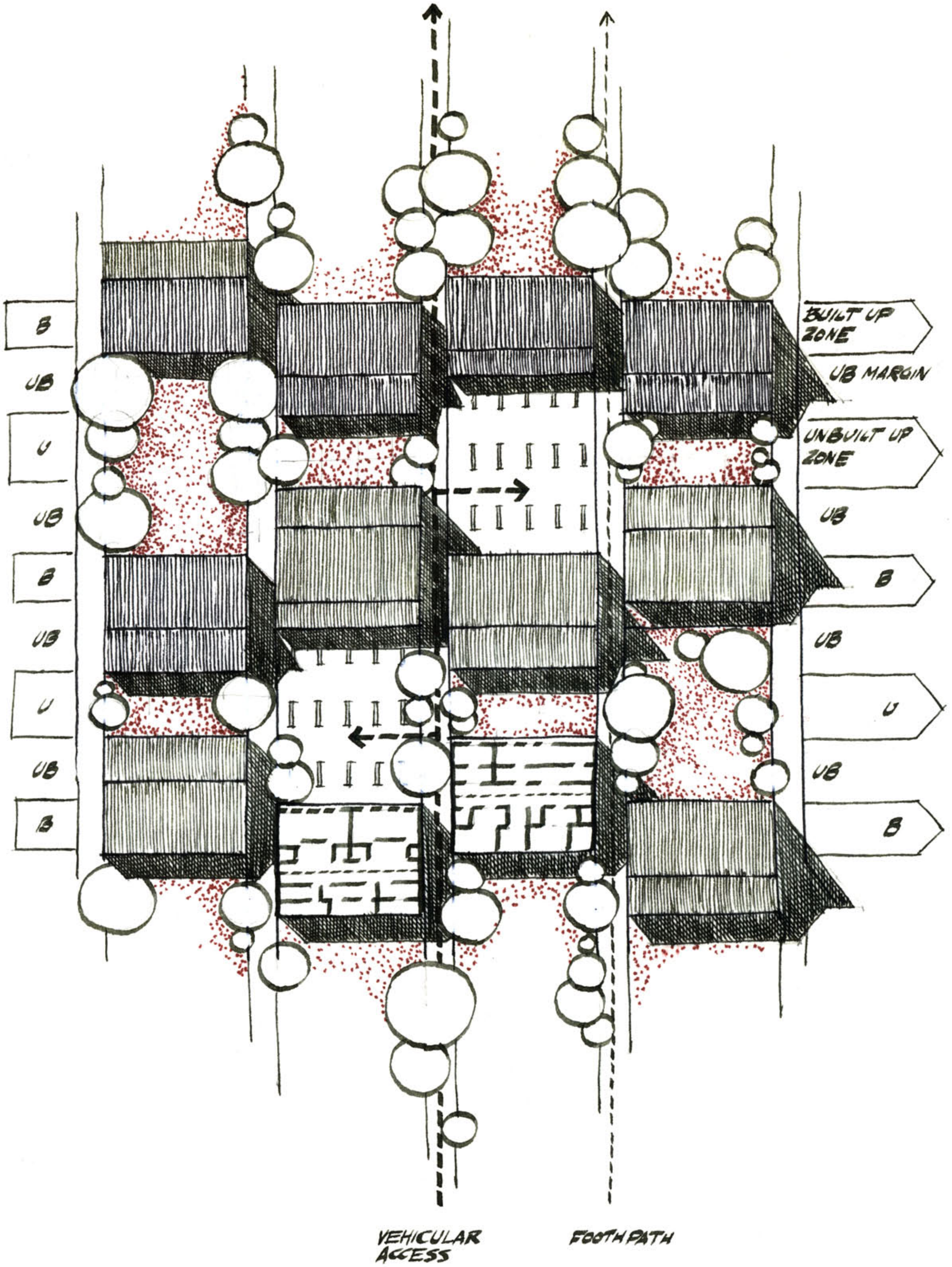




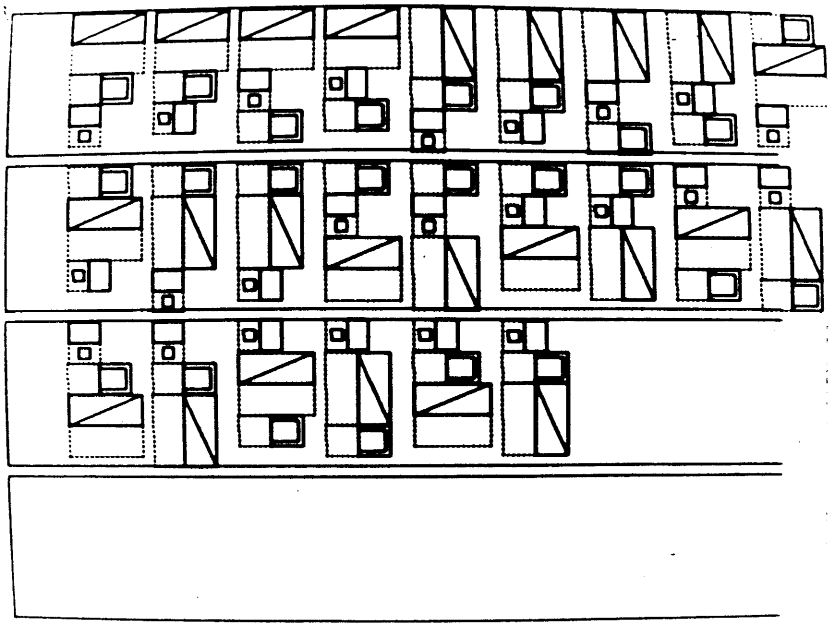




SUB-VARIATIONS:



URBAN TISSUE :



on.

In the following sections, some relevant techniques from other fields will be described as we have drawn from then the basic approach for our method of enumeration.

2.2.- Enumeration Techniques:

At the beginning of the Thesis, techniques from Combinatorial Theory were explored as it was thought that furniture arrangements constitute a problem of assigning furniture pieces to parts in the site, and therefore the enumeration of layouts and the existence of configurations are both problems of a strong combinatorial nature.

These techniques presented the atractive of a whole body of theory that could be brought to use in our particular case. A furniture layout for example, could be represented as a SYSTEM of DISTINCT REPRESENTATIVES problem, where we define for each space in the site a set of elements that can be positioned in it, and we see each furniture configuration as a selection of pieces, one among each set, that are positioned respectively in each of the spaces. Considering then each positioned piece a 'representative' of the spatial set, and considering the collection of positioned pieces as a 'system of distinct (not repeated) representatives' for all the sets we have.

Unfortunately, in our case, this assignment depends on several factors, like size, fitting in site, relations to other elements, that cannot be defined in the formulation of our problem, as our problem consist in fact in finding out first, what of these assignments can be done at all, and then enumerating the layouts that correspond to them.

As an alternative, 'problem-solving' representations and 'tree-searching' methods from Artificial Intelligence, were explored as models of our problem and as techniques that we can use for the evaluation of functional standards.

Generated from an interest in expanding the areas of application for computers, Artificial Intelligence has evolved during the last 20 years to explore, among other things, the formulation of general frameworks for 'problem-solving'. From psychological studies of how people proceed in solving particular tasks, to the development of techniques that permit a procedural definition of these approaches, it has produced, besides quite heated polemics (3), a series of principles and techniques that are relevant to our problem.

Whether the existance of these techniques show any degree of intelligence in the person or system that uses them, or whether there can be such a thing as a general problem solver, are questions not only beyond the interests of this thesis, but questions that tend to distract us from the relevance that these methods have in themselves.

The basic outline of problem-solving representations and tree-searching methods follows on.

2.2.1.- Problem-Solving in general, Representation and Search:

We can say that there is a problem to be solved when we perceive a discrepancy between a situation as it is and a situation as we think that it should be. Confronted with this discrepance, we are forced to find the action, or the collection of actions, that can reduce this difference and bring the actual characteristics of the present situation as near as possible to the desired characteristics of our ideal situation.

A problem, in these terms, consists in the recognition of discrepancies between different situations and the proposal of a plan of actions that can take us from the conditions that we have, to the conditions that we think we ought to have. (4)

There is enough room for discussion on how it is that we actually go about producing plans of action, but for our case, it might be sufficient to say that effective actions, must of the time, result from previous thought and evaluation, and such thought and evaluation arise from an understanding, through an internal model, of the problem structure that we are confronted with.

A model of this structure consists in an internalization of the main characteristics of the problem and the set of possible operations that we can perform to bridge the gap between present and desired conditions.

To build models or representations of a problem, we engage in a process of understanding, and to do so we have to demand certain conditions in the descriptions that we build. Descriptions should not contradict aspects of reality, since if we work with a representation instead of dealing directly with the actual situation, we want to correlate our results with results that the 'real' situation might produce, or otherwise our solutions can not be of any use. Descriptions should lend themselves to practical expression of the problem and permit the expression of the processes that can be used in our attempts to reach its solutions. We want to describe the structure of the problem in a consistent way, with a practical formulation of its information and a relevant representation of the processes involved in changing old conditions into new, more desirable ones.(5)

How to build such descriptions for the problem and how to manipulate them in looking for solutions, are the two main conceptual issues in 'problem-solving'

methods, corresponding to REPRESENTATION and SEARCH.

2.2.2.- Representation:

'Problem-solving' representations describe problem conditions together with laws of transformation that specify how to change one condition into another.

Problem conditions describe the actual or initial situation, an intermediate or partial situation, and the desired or 'goal' situation. The legal set of actions that can be used in solving the problem are defined by the transformation laws, and the combination of both conditions and transformations, specify the extent of a set of situations among which there might exist the solution that we are looking for.

As Newell and Simon present it: "...To state a problem is to designate (1) a TEST for a class of symbol structures (solutions of the problem), and (2) a GENERATOR of symbol structures (potential solutions). To solve a problem is to generate a structure, using (2) that satisfies the test of (1)..." (6)

2.2.2.1.- Basic Model, Post Production Systems:

The basic principle behind these representations goes back to a general computation mechanism presented by Emil Post in 1943. Post proposed to analyze expressions in logical systems as strings of symbols written in some finite alphabet, and analyze logical systems as "sets of rules that tell how some string of symbols may be transformed into other string of symbols". (7)

A simple model that represents, for example, the structure of 'palindromes', words that read identically forwards and backwards, in a Post's Production System, would be:

Alphabet: a,b,c

Axioms or initial situations: a,b,c,aa,bb,cc

Productions:

```
$ -->  a$a     (P1)

$ -->  b$b     (P2)

$ -->  c$c     (P3)
```

where, the alphabet represents the symbols we can use in constructing new strings. The axioms are our initial situations, or the strings that we take as given, not derived from any other source. '$' stands for a string, any string, that is either an axiom or a string that has been generated from the successive application of productions to axioms. The productions represent an ordered pair of symbol strings with a left side, such as '$' and a right side such as 'a$', which indicate possible transformations of the string on the left into the string on the right. That is '$' into 'a$a'.

As can be seen from the production rules, the system will only generate 'palindromes' and all the possible 'palindromes' composed from the letters 'a','b', or 'c', given the fact that the axioms are 'palindromes' already, and each production rule mirrors the same elements in both sides of a previous word.

The generation of the word 'bacacab' would be:

1.- $\boxed{a}$ ----- *initial situation*

2.- $c\boxed{a}c$ ------ *production (P3) a → cac*

3.- $a\boxed{c}\boxed{a}\boxed{c}a$ ----- *production (P1) cac → acaca*

4.- $b\boxed{a}\boxed{c}\boxed{a}\boxed{c}\boxed{a}b$ --- *production (P2) acaca → bacacab*

and the 'problem' of generating a goal 'bacacab' out of an initial situation 'a' would be equivalent to the problem of finding out the sequence of production rules that will take us from 'a' into 'bacacab'.

Our problem description has built within itself then, both the capability for generation and the test for solutions. The structure of palindromes is understood and expressed as a 'system of transformations'. "...In as much as it is a system and not a mere collection of elements and their properties, these transformations involve laws: the structure is preserved or enriched by the interplay of its transformation laws, which never yield results external to the system nor employ elements that are external to it..."

"...(its) notion of structure is comprised (in short) of three key ideas: the idea of wholeness, the idea of transformation, and the idea of self-regulation..."

(8).

2.2.2.2.- Graph Notation:

Before going into the description of variations of this basic model, we should look first at some notational principles used in the description of problem-solving methods that are relevant to this thesis.

Problem-solving representations share, besides being systems of transformations, the use of the mathematical notion of a GRAPH as a common notation. (9)

A "GRAPH": G = (N,E), consists of a finite, nonempty set of 'nodes' "N", and a set of 'edges' "E", used to represent a set of elements and relations that exist between them. The set of nodes "N" stands for the elements we want to talk about, and the set of edges "E" corresponds to the relations that exist between pairs of these elements in the set. Graphically, nodes are represented by dots or circles, and edges are shown as lines that link related nodes. A "GRAPH" would be, for instance:

G = (N,E)

N = (l,m,n)

E = ((l,m),(l,n),(m,n))

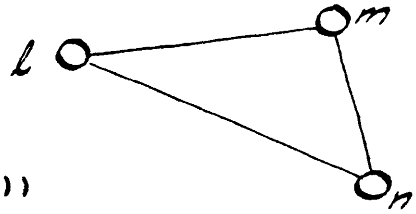


FIGURE 2.1

If we think, as we did before, of problem representations being general descriptions of problem situations related by transformations, we can begin to see the correspondance between the notion of a graph and the notion of problem representation.

At a first level, nodes can correspond to elements, conditions or characteristics in a problem, edges can correspond to desired relations among them, and a graph would stand then for an initial, a partial or a final problem situation. At a second level, nodes can become now problem situations in themselves, and edges represent transformation laws, production rules, or changes from one situation into a new different one; the graph standing then for the set of situations among which we search for a solution. Being the graph an abstract description of a structure, that is a set of parts and their relations, it can be used to represent both the structure of particular problem situations, and the structure of general problem transformations.

According on how we define the nodes and edges, a graph can be described explicitly, as in the example above in figure(2.1), or it can be described implicitly, as in the generation of the 'palindrome' words, where nodes were not listed one by one, but defined as initial situations or simply as valid combinations; and where edges were defined as production rules, or as relations between general schemes of strings, denoted by '$', and new configurations that contain the previous scheme plus an addition of the letters 'a', 'b' or 'c'. It is said that the graph is defined implicitly, because by means of initial situations and production rules, we can always have a way of finding out whether a string is a member of the set of nodes, or whether a transformation is a member of the set of links, instead of having to define explicitly each and everyone of the members of both sets.

A graphical representation of one portion of the implicit 'palindrome' graph, would be:

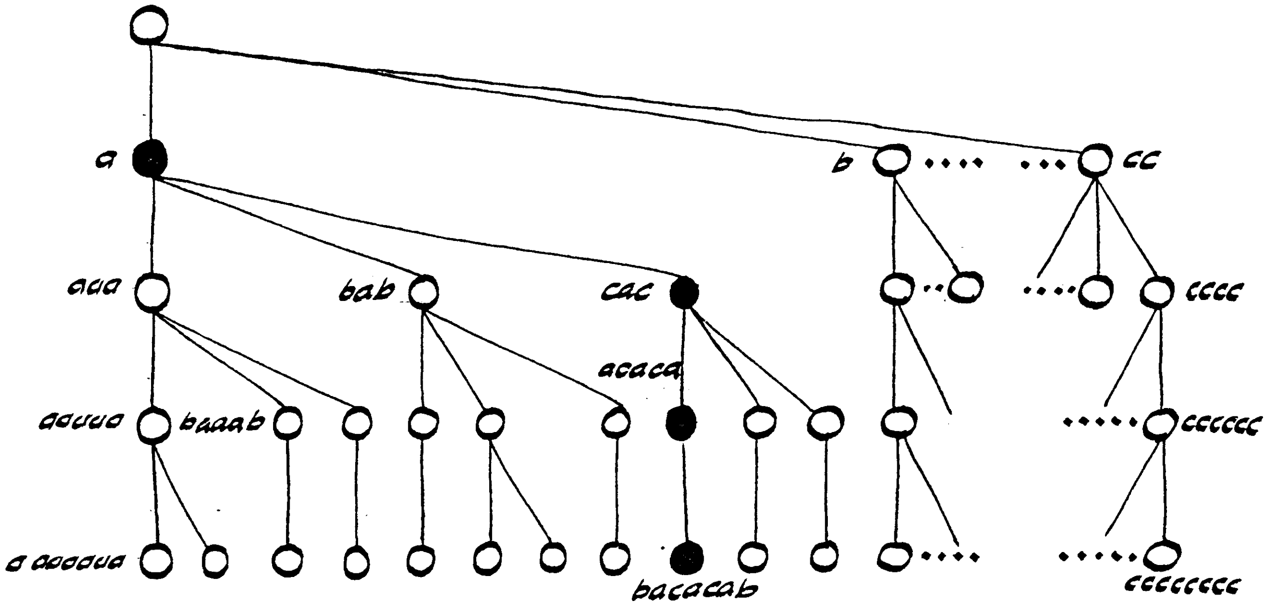


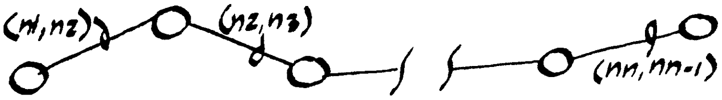
FIGURE 2.2.

where at the too we have an empty string, from where we can select each of the possible axioms, to which we can apply each of the possible productions, and continue doing so as long as we want, generating new 'palindromes' everytime the graph grows further and further down.

If this graph notation is going to be used as a convention for problem-solving representations, then it is important to define several concepts that are relevant for this purpose, besides what we have already said about nodes and eages.

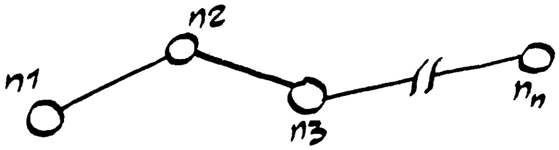
When we have a seouence of edges of the form:

(n1,n2),(n2,n3),(n3,n4),.....,(nn,nn-1)



where the node at the end of each edge corresponds to the node at the beginning of the next edge, this sequence is called a 'path'. A 'path' goes along a sequence of linearly connected nodes and for this reason it can also be represented as:

n1,n2,n3,......,nn



and be said to have a 'path length' of n-1, that is a length equal to the number of edges included in the sequence.

When all the nodes in a 'path' are distinct, with exception possible of the first and the last node, then we say that the 'path' is 'simple'. When the first and last nodes of a 'simple path' are equal to one another, then we call this path a 'cycle'.

The graph in figure(2.1) has one cycle, from 'I' to 'I', and the graph in figure(2.2) has no cycle at all, but several paths. When a graph, like the second one, has a first node which no edge enters, each of the other nodes have only one entering edge, and from the first node, called the 'root', there is a 'path' to every node in the graph, then the graph is called a 'tree'.

'Trees' are important graphs to us, because in their paths they show distinct sequences of nodes from an initial node, the root, to some final nodes, down at the bottom of the graph. The initial situation plus the rules of transformation, can then 'grow' a 'tree' whose branches are all the possible paths from that given situation, to the set of situations that might constitute a solution; and finding a solution becomes then equivalent to finding the sequence of transformations through a certain path that can take us from the tree root to a desired node down its branches.

Everytime we grow a series of edges out of a node in the tree, we say we expand the node one level down. The set of nodes at the end of these expanded edges are called the 'successors' of the expanded node, and these in turn becomes the 'predecessors' to the newly created nodes.

Nodes and edges, trees and paths, are basic concepts of Graph Theory which are used as notation for problem-solving methods not only because of their expressive possibilities, but because they give us access to other theoretical notions that will be explained later on (ref. chapter(3)).

2.2.3.- Main types of Representations:

There have been three general kinds of problem representations in problem-solving methods: (10)

1.- STATE-SPACE REPRESENTATIONS.

2.- PROBLEM-REDUCTION REPRESENTATIONS.

3.- THEOREM PROVING.

2.2.3.1.- State-Space:

In the case of STATE-SPACE representations, problem conditions or problem situations are described by "STATE DESCRIPTORS" which represent certain characteristics of the problem solution at a certain point in time.

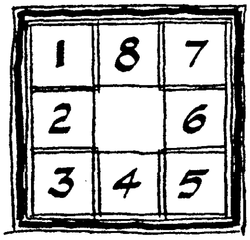
Initial and final situations are expressed respectively as existing and desired characteristics which not necessarely have to be restricted to the format of strings, but that can take any form of description more appropiate for the problem in hand. In the case of the 'palindromes', for instance, strings would have been such a form and the words 'a' and 'bacacab' would have been the initial state descriptor and the final state descriptors for the problem of generating the expression 'bacacab'.

Legal transformations in this representation appear as "OPERATORS" or rules that specify, in very much the same fashion as in Post Production Systems, how to change a "STATE DESCRIPTOR" into a new "STATE DESCRIPTOR".

The set of all situations that can be reached by the application of these operators to the initial state constitute what is called the "STATE-SPACE", that is, still back in our 'palindrome' example, all the combinations of words that contain the letter 'a' in the middle and whose letters repeat alternatively 'a' or 'b' or 'c' on each side of it.

An example, taken from Nilsson(1971), where a STATE-SPACE representation has neither state-descriptors, nor operators described as strings, is a model for a sliding-block 8-puzzle.

In this puzzle there are 8 numbered block located in a 9, 3 by 3, cell space, which can be slide agains the empty cell to form certain configurations such as:



1	8	7
2		6
3	4	5

Operators in this case correspond to the valid and possible movements of the empty cell from one location to another, as blocks are slided to occupy its previous place, and an example of the operators 'rules' would be figure(2.3).

Supposing that the initial situation is:

and the final, desired configuration is:

then one sequence of transformations that can

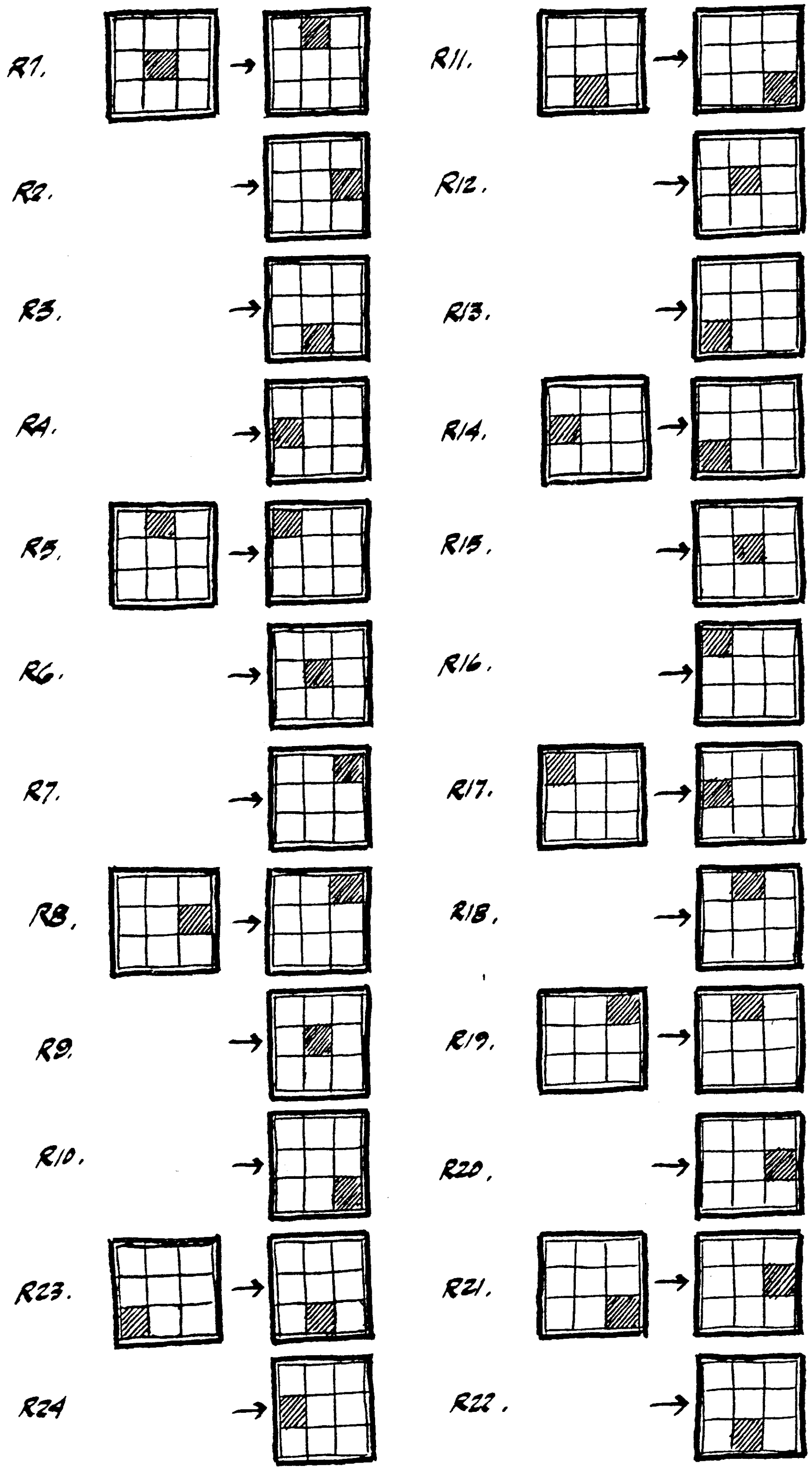
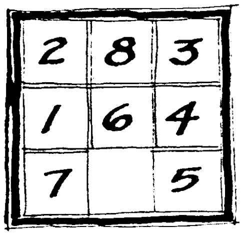
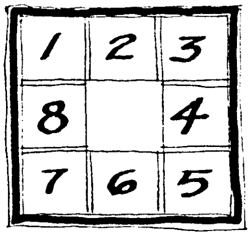

R1.
R2.
R3.
R4.
R5.
R6.
R7.
R8.
R9.
R10.
R23.
R24
R11.
R12.
R13.
R14.
R15.
R16.
R17.
R18.
R19.
R20.
R21.
R22.

FIGURE 2.3:



2	8	3
1	6	4
7		5

FIGURE 2.4



1	2	3
8		4
7	6	5

FIGURE 2.5

produced the final configuration would be:

STATE-SPACE representations lend themselves to practical expressions of problems with structures that have a sequential characteristic. Different situations can be explored from previous situations. At any point in time

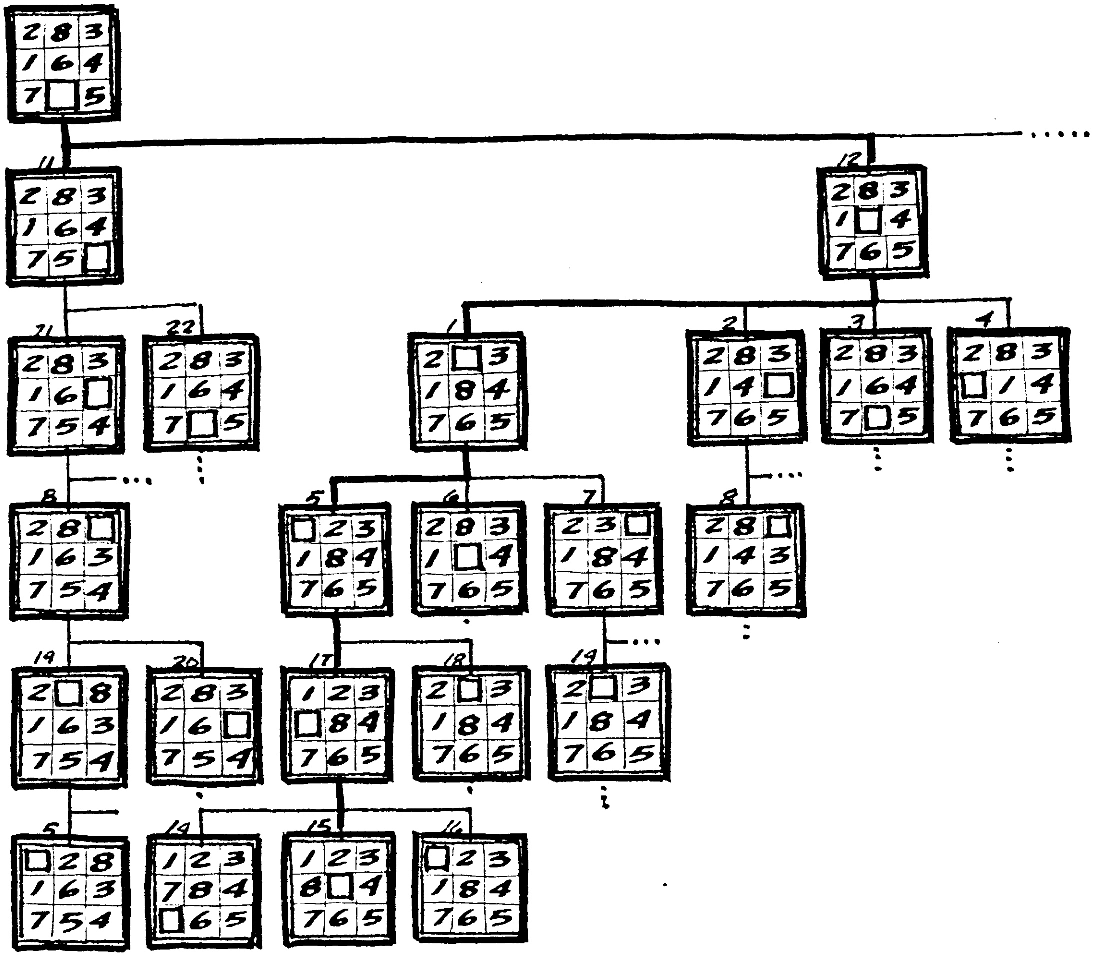


FIGURE 2.6.

we can analyze the state we are in, to find what is the existing difference to our final goal, and the process of reaching a solution can be composeo of a concatenation of operations, one after the other, continuously modifying a state into a new state.

2.2.3.2.- Prcblem-Reduction:

In PROBLEM-REDUCTION representations, we deal instead with the structure of the problem itself. Rather than working with descriptions of the different steps taken to solve a problem, we explore how an original problem can be reduced or decomposed into simpler "primitive problems" which solutions imply the solution of the larger one.

Problem reduction methods are concerned with strategies that can be pursued to reach a solution. These strategies are oriented to decompose an original problem into a set of smaller components which solutions might be easier to obtain.

The elements that we deal with in this mode of representation are therefore descriptions of problems, called "PROBLEM DESCRIPTORS", consisting of initial situations, goal situations and operators that transform one into the other, as was presented in the previous 'state-space' representation. We might, in fact, think of this representation as being one level higher from the previous one, and contain 'state-space' representations as problem descriptors of components in a hierarchy of problem situations.

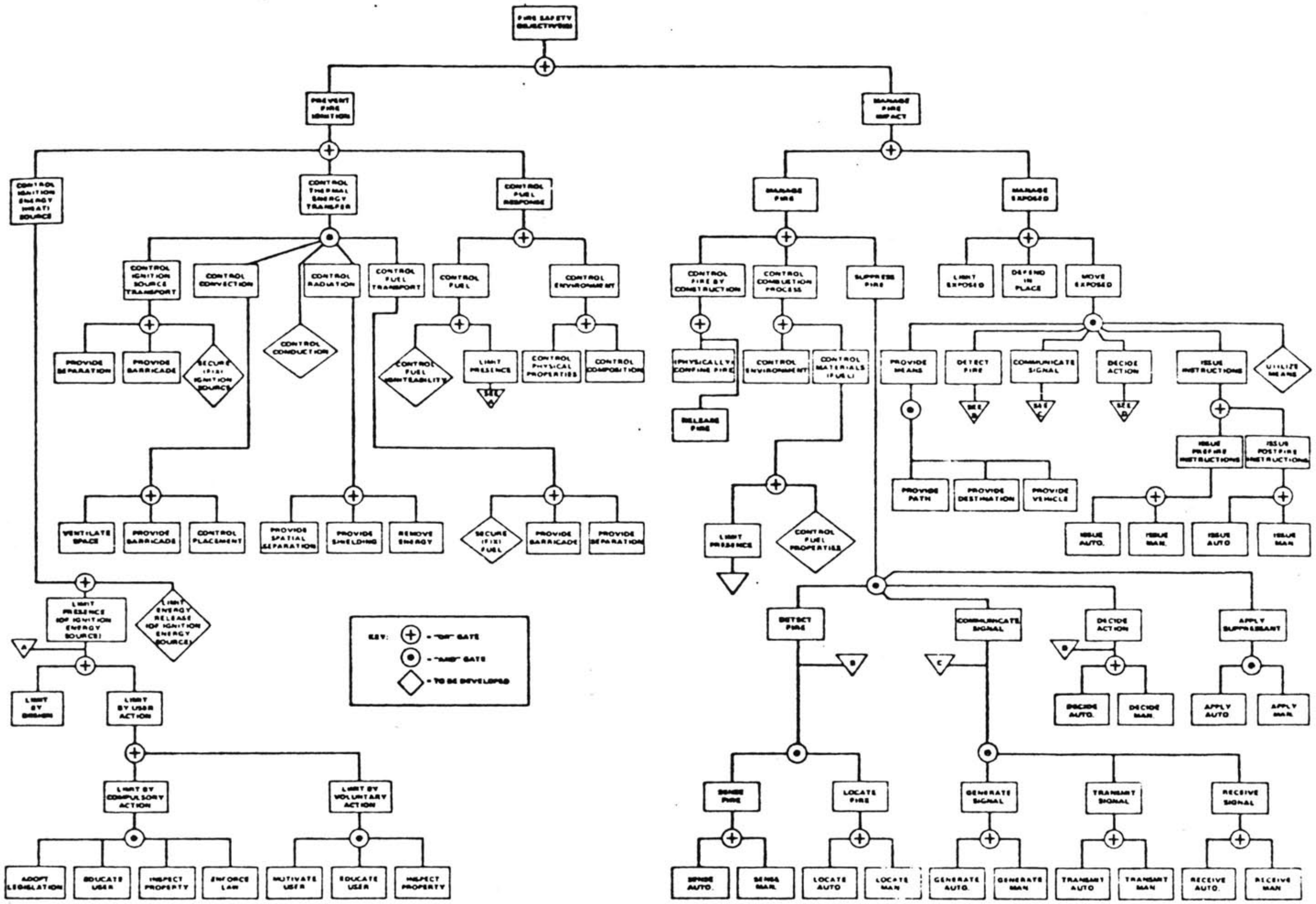
The legal transformations in this case, are decompositions of one problem into its possible components. They are accomplished through "OPERATORS" that specify how one problem descriptor might be transformed into a set of possible subproblem descriptors. Through the application of these operators, we can generate a set of related subproblems and among its combinations look for those descriptors that we can more easily satisfy.

If the 'state-space' of a problem represents the set of all possible situations that can be reached by applications of state-space operators, the application of problem-reduction operators generates the set of all possible strategies we can chose from, before exploring any particular state-space descriptor.

One example of this reduction method can be the following 'decision tree', proposed by the National Fire Protection Association for the analysis of fire protectionsystems for buildings.(11) In here, the most general statement of the problem might be simply 'fire protection', but the design of fire protective measures implies the selection among different valid strategies, the one that provides the desired protection and the desired cost or performance. figure(2.7).

The elements are different actions that can be taken as protective measures, and their implementation implies a subset of problems that have to be solved. Subproblems which, in their turn, represent different actions with smaller subset of problems and so on until we get some basic "primitives" for every kind of action.

The operators break a protection measure into a collection of alternatives that can solve its implementation. For each collection of alternatives different combinations constitute a solution. There might be alternatives which by themselves can solve the problem, or alternatives which can be used only in combination with other different measures. The first alternatives are described, in our graph rotation, as "OR" successors of problem descriptor nodes. Either of them can solve by itself our problem, and we say then that alternative one, OR alternative two, OR whathever other alternative is available, can be selected as a winning strategy. The second alternatives are "AND" successors of problem descriptor nodes. All of them have to be satisfied in



Decision Tree

FIGURE 2.7

order to to have our problem solved, and therefore we say that alternative three, AND alternative four, AND alternative five, must be selected and satisfied to proceed in our solution.

As can be seen in the picture (figure(2.7)), a strategy consists then of the combination of paths, through 'AND'/'OR' nodes, that reaches a set of 'primitive problems' which solution can be found.

Problem-reduction representations can be attempted when the problems that we want to model can be decomposed in a similar fashion, when its solution structure has this hierarchical order among its different parts, and when the process of reaching a solution can be stated as a synthesis of related "primitive" solutions.

2.2.3.3.- Theorem-Proving:

In THEOREM-PROVING representations, situations are described using a logical formalism, for example "first-order predicate calculus", as a language in which initial and final conditions of a problem can be expressed as valid sentences, and where logical analysis can be performed in order to find out implications, proofs or deductions about statements of our problem.

The elements in this representation, belong, as in the case with Post Production Systems, to a given alphabet. Their combination result from operations that dictate how symbols can be assembled into legitimate strings or expressions, called "well formed formulas"; which relevance, besides their legal formulation, can always be decided by interpreting them as assertions on some domain of interest.

The formalism represents the set of all the valid and meaningful statements that can be made about an area in particular, as new statements can be deduced or manipulated by the application of 'rules of inference' to previous statements. Its two main parts include first, the 'syntax' or the part that regulates how "well formed formulas" can be constructed out of other "well formed formulas" or out of symbols in the alphabet; and second, the 'semantics' or the part that relates "well formed formulas" to the domain of interest, by assigning them a 'true' or 'false' value.

Although this representation offers the

aavantages of generality, uniformity of representation and the logical power of techniques for making deductions, it always remains difficult to reach the level of formalization that is demanded, and difficult also to express our knowledge of specific problems in logical

formalisms as the predicate calculus.

2.2.4.- Search:

For all the previous representations, once we have formulated our problem in their terms, the second issue that remains to be solved is how to find the sequence of operators or inference rules, that can transform a state descriptor into another state descriptor, break a problem into its components, or deduce a new statement out of an old one.(12)

Which alternatives to select when there are several transformations that can be applied, and how to control the growth of branches in our tree to a number of paths that still can be explored within reasonable time

bounds, are the main problems of search.

2.2.4.1.- Basic Techniques:

For the first problem, that is which

alternatives to select next, two conventions can be established on how to explore systematically all the paths in the solution space of a given problem. Depending on how we proceed exploring nodes, or in what orded we decide to generate alternatives at each level of the graph, we can move along the breadth or along the depth of the paths that extend out of the tree root.

If we decide to explore all the successor nodes at a given point, before continuing to expand them into other levels further down, we say we conduct the search in a BREADTH-FIRST manner as shown

in figure(2.8). If we decide to explore only one node at each level of the tree, and proceed doing so, for each successive nodes until we reach a terminal branch, or until we have explored all of the possible paths, we say that we conduct the search then in a DEPTH-FIRST way, see figure(2.9).

BREADTH-FIRST or DEPTH-FIRST searches are conventions on how to visit each of the nodes in a solution space, and depending on the structure of the problem, each of them has particular advantages or disadvantages. When solutions are unevenly distributed through the levels of the tree, as in figure(2.8), a

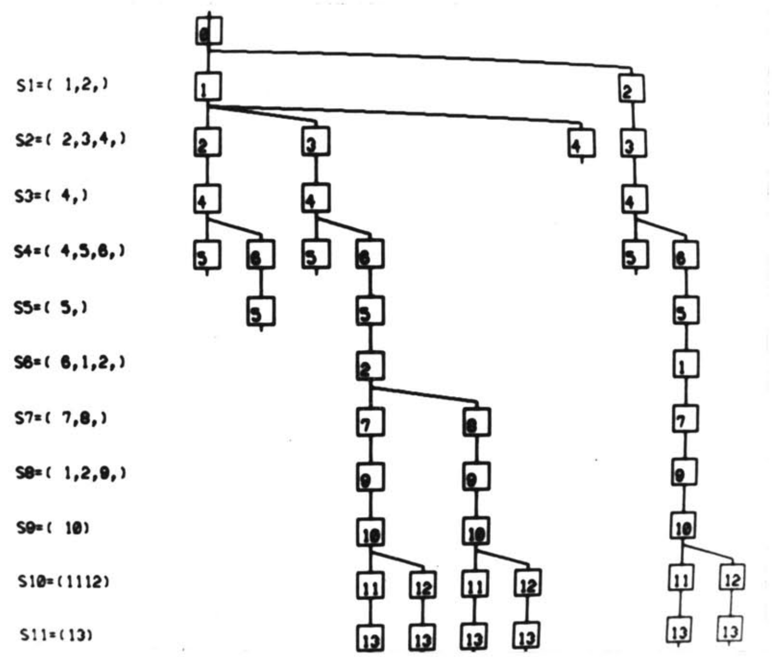


FIGURE 2.8

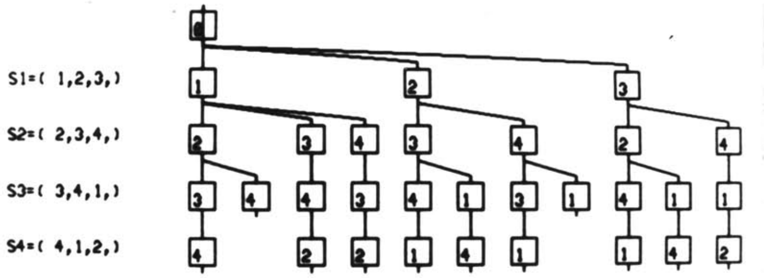


FIGURE 2.9

depth-first search may spend longer time exploring alternatives beyond the level where a shorter path might have been found if we had use breadth-first search instead. But, on the other hand, when solutions exist at similar levels of the tree,(figure 2.9) a lot of unnecessary work would be done by breadth-first searches when depth-first would find the solution much more faster.

Independent of these conventions, trees for problem-solving situations tend to increase their size quite rapidly. Even small alternative generators, like the oprators in the 8-puzzle presented in the description of state-space representations, combine with each other into large number of possibilities and paths. For example, in this case, three kind of operators, one -at the center position- that produces four alternatives when applied, four -at the different sides- that produce three alternatives each, and four -at each of the corners- that generate only two alternatives; can combine, in sequences that contain 14 moves, into a state-space of 1,497,792 possible paths, extending out of our initial configuration in figure(2.4). Problems with larger number of operators, and with operators that expand larger number of possibilities can quite easy reduce a representation in these terms, to a non-operative alternative.

The control of this 'ever present threat of exponential explosion of search' (6), demands a knowledce of the problem structure. We can, in fact, measure our understanding of a problem in terms of representations which processes reduce search to a minimal operation, and say that the less we know about a problem, the more we have to search for its solution.

A 'pruning' of branches, or a reduction of the combinations that have to be explored in a problem space, can only come from embedded knowledge in the process of branch selection and generation of alternatives.

A policy for branch selection can take advantage of particular characteristics of the problem, and decide which is the next expansion to proceed, by ranking successive nodes against their 'promise' (10) to succeed; or suppress altogether the exploration of branches, by certain rules of thumb, called 'heuristic information'(10).

Simple heuristic information can reduce

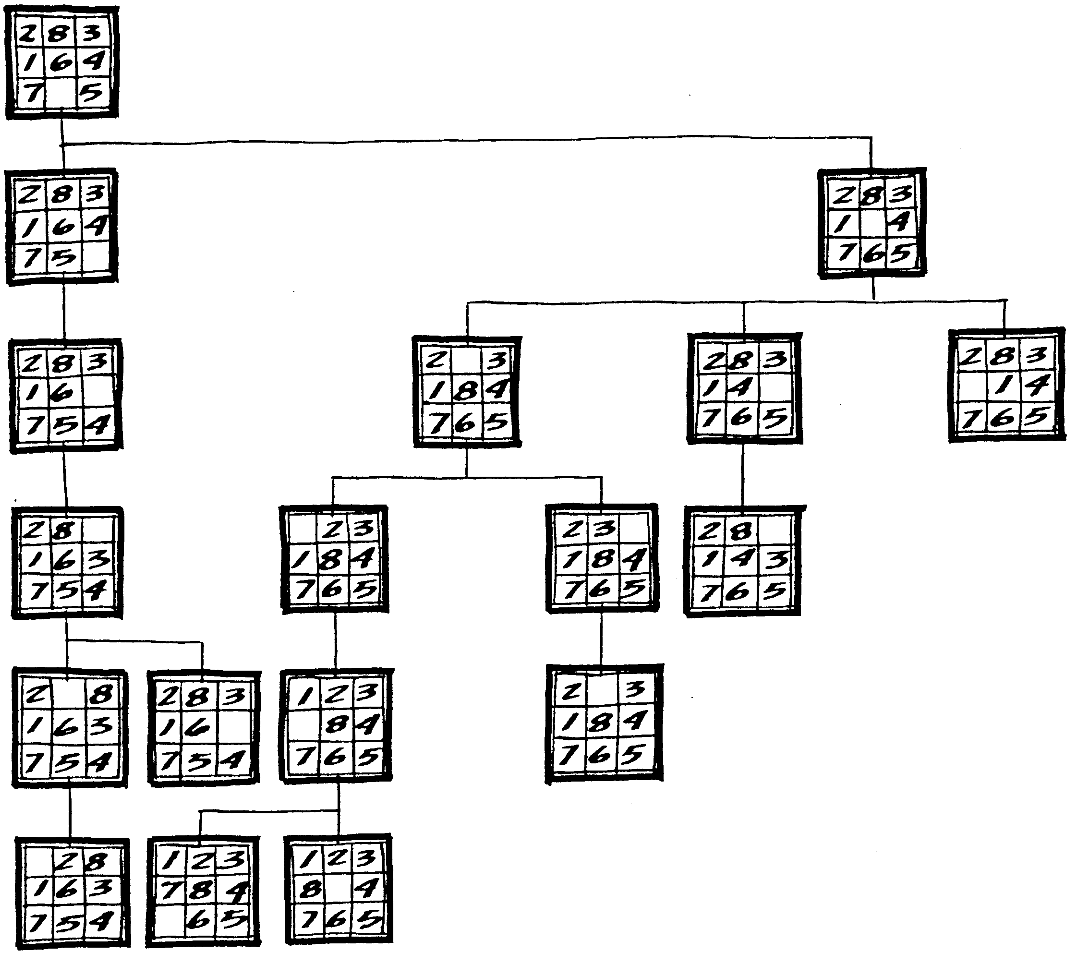

2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6
7 5 4
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5
2 8 3
1 4
7 6 5
2 8
1 6 3
7 5 4
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
2 8
1 4 3
7 6 5
2 8
1 6 3
7 5 4
2 8 3
1 6
7 5 4
1 2 3
8 4
7 6 5
2 3
1 8 4
7 6 5
2 8
1 6 3
7 5 4
1 2 3
7 8 4
6 5
1 2 3
8 4
7 6 5

FIGURE 2.10 :

substantially a problem space. In the 8-puzzle tree of figure(2.6), all operators were applied blindly to each problem state, without recognizing that for every type of operation -center,side,corner- there is a movement that reverses the situation to the state we had before its application. Preventing their application results in the smaller partial tree of figure(2.10).

If besides these reductions, the 'promise' of each node can be evaluated in terms of the length of the path and the number of misplaced blocks, then the 'EVALUATION FUNCTION' (10):

$$f(n)=g(n)+W(n)$$

where g(n) is the path length, and W(n) is the number of misplaced blocks, can help us to select the nodes in a "BEST-FIRST" manner and reduce the search to the tree in figure (2.11).

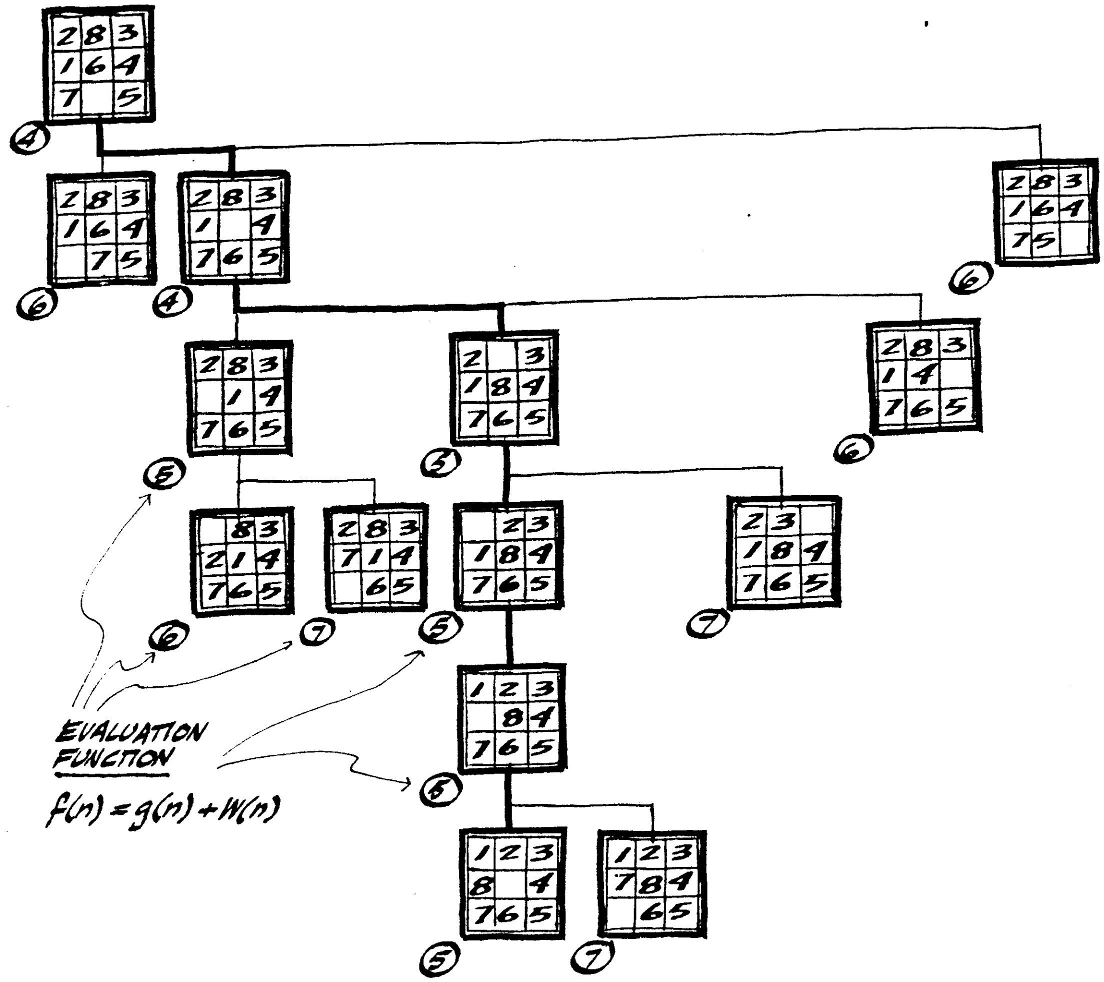


FIGURE 2.11

2.2.4.2.- Backtracking:

One of the exhaustive techniques for searching the set of all possible solutions to a given problem is 'BACKTRACKING'. (13)

It explores systematically the solution space of a problem, by partially expanding solutions an element at the time, in a depth-first fashion, and by 'backtracking' or retracing its steps to the state of a previous decision in order to try another possibility, whenever it reaches a point where no further elements can be added, or whenever all the components have been added to form a valid result.

Problems amenable to being solved by this technique have a combinatorial structure that permits the sequential expansion of their solutions. These are formed by several parts, each of them capable of taking one of several values, depending on some general definition of the problem. In this definition the set of parts is clearly established together with all their values and the restrictions or 'constraints' that stipulate what constitutes a valid result.

Their structure consists of:

- A set of parts, or 'selection spaces'

(X1, X2, ..., Xn)

each of which represents a set of possible values from where a particular decision or selection can be made according to a

- Criteria of constraints

*(x1, x2, ..., xn)

In order to expand a

- Solution represented as a 'vector' of length n

(x1, x2, ..., xn)

where every element 'x1' 'x2' to 'xn' correspond to a valid selection from the set of parts 'X1' or 'X2' or 'Xn' respectively.

An exhaustive search for solutions in this structure means that all the possible values for 'X1' have to be considered, one way or another, against all the possible values for 'X2' and, their result similarly compared with all the possible values of all the 'selection spaces' until 'Xn' with all its elements has been explored. The 'Cartesian Product' of all these sets, or the product of all the elements in the set 'X1' times all the elements in the set 'X2' and so on until set 'Xn', i.e.: X1 x X2 x ....x Xn, represents the number of possibilities that have to be explored in order to find out the set of all vectors that satisfy the constraint restrictions for valid solutions.

In a BRUTE FORCE approach, what we would do, is proceed to construct each of the possible complete vectors resulting from these combinations and once constructed, test them against our criteria in order to find out if we have a valid solution.

The way backtracking works however, makes unnecessary the explicit consideration of all the values in the 'selection spaces'. By proceeding sequentially in the selection of values for a solution, we can always test at whatever point we are, what are the chances for succeeding in the vector being expanded.

Looking at the criteria of constraints we can always tell whether the next set, from where we can select an element, contains a candidate for a valid extension of the vector, or whether by having none of these, our solution can not be expanded in that direction any more.

At any point in time during the generation of a solution, we can not guarantee that a valid solution is being formed, but we can always know when a partially valid solution can not be extended anymore. We can not guarantee continuous advance towards a solution, but we can provide a stopping rule that excludes large sectiors of our solution space, without having to explore them explicitly, and without having to wait for a complete vector in order to test for its validity.

Backtracking can be better understood using our graph notation: figure(2.12).

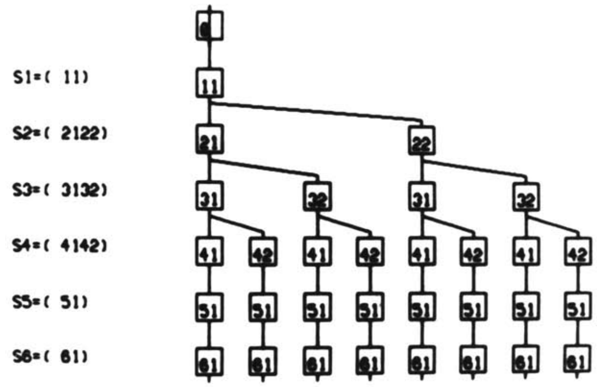


FIGURE 2.12

- At the beginning of the tree of possibilities, we have an initial solution, or a node that represents our initial vector of length zero, as no decisions have been made yet.

- From this starting point, we can construct a tree by representing each of the possible selections as branches that grow out of this root. Each of the values in "X1" would appear as a node at the end of these branches and stand for a possible selection to be added to our vector.

- From each of these nodes, we can select now one of the values in 'X2' which, in its turn, would expand into a second level of branches; and from the resulting nodes we continue branching on until we have included all the possible selections of 'Xn'.

- Constructing a solution, consists then in pushing our path towards further levels down in the tree, until we reach a terminal branch, or until we hit a dead end.

By convention, we can select branches out of a node, in a left-to-right manner, such that we always pick out the first branch in the left to exit a node, and we always return to the next available left branch when we retrace our steps to return to a previous node.

With these two directions, down and left-to-right, we can move systematically across all the paths in the tree, visiting the nodes in the following way: (FIGURE 2.13)

The importance of backtracking as a search mechanism, however, relies in a stronger criteria for branch selection that incorporates, as described before, tests or 'stopping rules' to help reduce the number of nodes that have to be explicitly explored. With such criteria, everytime we advance from a given node to its

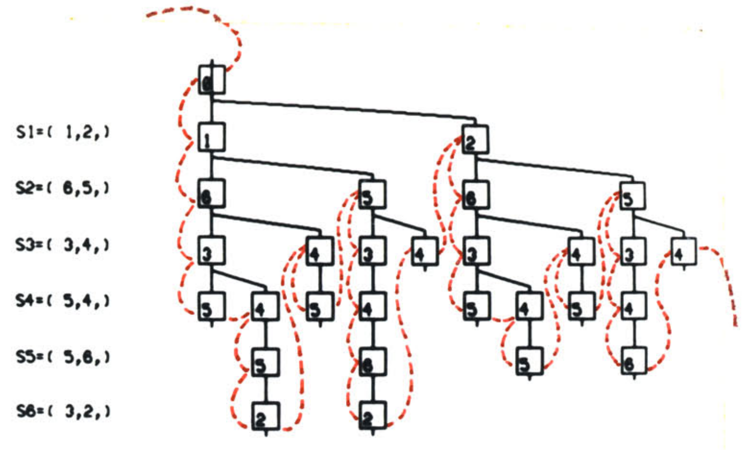


FIGURE 2.13

successors, we look first for a subset of valid choices that do not violate any of the problem constraints, and from this subset we pick out its leftmost member.

A graphics example in figure(2.14) shows the consequences of this procedure. Starting from the root, we find first the node "x11" as a possible extension and we advance there. At this point, we look now for the valid subset of "X2" and find out that the nodes "x21" and "x22" are both invalid selections and only the node "x23" constitutes a valid possibility of extending our path. By doing so, we can see now, how a whole region of the tree, extending below the invalid nodes, is ruled out of consideration, since all the paths that go through these

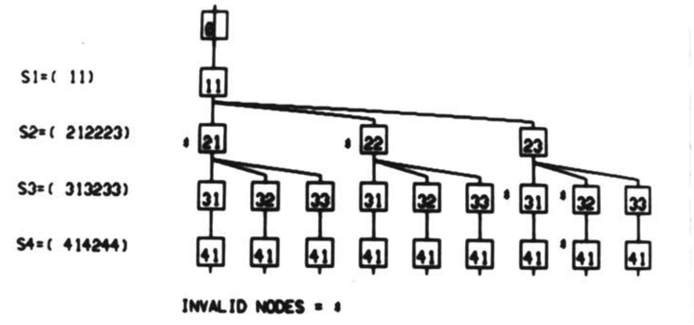


FIGURE 2-14

nodes would by definition be wrong.

This cutting out regions during the search process is called 'preclusion'. By discovering a dead end at a certain level of the tree, we 'preclude' or exclude from further considerations, all the paths below such points. To preclude large regions of the tree we have to formulate our constraints in a way that makes such sequential analysis possible, and structure our solution space in a way that brings forward these violations as soon as possible.

One way of doing this is to sort the 'selection spaces' by increasing number of choices along the different levels of the tree, so that we have the sets with the smaller number of elements at the beginning or near the root, and the larger sets at the bottom of the tree. As violations occur in certain combinations of elements, having the fewest choices at the beginning will tend to produce larger preclusions of paths than if it were done otherwise.

Together with "preclusion" and "branch ordering", some other techniques such as "branch and bound" and "branch merging", are used to help reduce the amount of work spent searching for solutions.

"Branch and Bound" incorporates to the criteria for branch selection, considerations for preference values among different successors. Besides knowing if a successor is a valid or an invalid option, we can rank it now against the others according to a predefined scale of preference, .and proceed in out selection trying to maximize or minimize the overall preferences in a solution. Bounded by lower of upper limits respectively, our criteria for acceptable solutions is continuously modified as we move along the branches of the tree and encounter new choices that can be made. Looking at them we can decide whether or not we can improve our situation, and by increasing or decreasing our previous bounds, drop out branches that extend beyond out limits, effectively reducing the regions of nodes that have to be considered.

'Branch merging', on the other hand, recognizes the fact that in many cases what increases the size of a solution space, is not only the explotion in combinatins of elements, but redundancy in the definition of paths. We might spend a lot of time considering different regiors that constitute only different versions of a same solution. As might be the case in problems whose solutions share symmetries, rotations or translations, all of them transformations that allow us to construct new solutions out of old ones. Solutions that share these properties are said to be equivalent under such transformations, or 'isomorphic' to one another. Branch merging or 'isomorph rejetion', as it is also called, loks for these equivalences either before or during the search and tries to merge or collapse 'clusters' of equivalent paths into sequences of non-isomorphic solutions which reduce our solution space, but allow us nevertheless to expand the results to all the possible variations if desired.

2.3. Conclusions:

As a combination of all these techniques, backtracking provides an organized approach to exhaustive searches. Increase in size of •selection spaces• can still bring back combinatorial explosions, and its solution time, even with the use of digital computers, can take in some cases, more than anyone could wait. But a clever use of preclusion, the implicit vs. explicit enumeration of solutions and the sequential expansion, with the implications that this has on memory resources, still makes of backtracking a valid method for enumeration problems which could not be solved otherwise.

The importance that it has in generating solutions, is frequently critiziced in the same grounds. Having to construct solutions in orded to find out if they exist at all, might not be a graceful or elegant way in a theoretical sense, but must of the time, for good or for bad, it is the only choice we have for problems whose structure still lacks a more powerful explanatory theory.

In our particular case, the generation of design configurations, this criticism should not stop us from using tree-searching methods, but rather take it as it is, an indication to a larger need that demands future and related development.

And realize that the "...computerization of these processes is only of secondary importance. The main issues are still the better understanding of the theory of spatial configurations and of our reasoning in manipulating them. Here seem to lie the significance of investigations..." (14).

Chapter Two, Notes:

(1)(2) BOEKHOLT J.T., THIJSSEN A.P., DINJENS P.J.M., HABRAKEN N.J.
"Variations: The Systematic Design of Supports",
forthcomming M.I.T. Press. Cambridge Mass. 1976.

(3) WEIZENBAUM J.,
"Computer Power and Human Reason".
W.H. Freeman, San Francisco 1976.

--- DREYFUS H.L.,
"What computers can't do".
Harper & Row, N.Y. 1972.

--- PAPERT S.,
"The Artificial Intelligence of H.Dreyfus"...

(4) ARCHER, B.L.,
"An Overview of the Structure of the Design Process".
in "Emerging Methods in Environmental Design and Planning", (Ed. G.T. Moore), M.I.T. Press,
Cambridge 1970, pp.285-307.

--- RITTEL, H.,
"Some principles for the design of an educational system for design".
part I, DMG-Newsletter Vol. 4 No. 12, Dec. 1970 pp.3-10.
part II, DMG-Newsletter Vol. 5 No. 1, Jan. 1971 pp.2-11.

(5) KAPLAN A.,
"The Conduct of Inquiry",
Chandler Pub.Co., San Francisco 1964.

--- McCARTHY J.,
"Some Philosophical Problems from the Standpoint of Artificial Intelligence".
Machine Intelligence 4/1969, (Ed. Meltzer and Michie).
American Elsevier, N.Y., 1969 pp.463-502.

(6) NEWELL A., & SIMON H.,
"Comouter Science as Empirical Inquiry: Symbols and Search".
Comm. A.C.M. Vol. 19, No. 3, March 1976, pp.113-126.

(7) MINSKY M.,
"Computation: Finite and Infinite Machines".
Prentice-Hall, Englewcod Cliffs, N.J., 1967.

(8) PIAGET J.,
"Structuralism",
Harper Torchbooks, N.Y., 1971.

(9) HARARY,

(10) NILSSON N.,
"Problem-Solving Methods in Artificial Intelligence".
McGraw-Hill, N.Y. 1971.

(11) Committee on Systems Concepts for Fire Protection
In Structures, Nov. 1974, NATIONAL FIRE PROTECTION
ASSOCIATION, Boston Mass..

(12) SEARCH

(13) GOLOMB S. & BAUMERT L.
"Backtrack Programming",
J.A.C.M. Vol.12 No. 4, Oct.65, pp.516-524.

---- BITNER J.R., & REINGOLD E.M.,
"Backtrack Programming Techniques",
C.A.C.M. Vol.18 No. 11, Nov.75, pp.651-656.

---- KNUTH D.,
"Estimating the Efficiency of Backtrack Programs".
Mathematics of Computation, Vol. 29, No.129, Jan.75,
pp.121-136

---- SWIFT J.D.,
"Isomorph Rejection in exhaustive search

techniques".
Amer. Math. Soc. Proc. Symp. Appl. Math. 10
(1960),
pp.195-200.

---- WALKER R.L.,
"An enumerative technique for a class of
combinatorial
problems".
Amer. Math. Soc. Proc. Symp. Appl. Math. 10
(1960),
pp.91-94

(14) RITTEL, H.,
"Theories of cell configurations" in Emerging
Methods in Environmental Design and Planning, ed. G.T.
Moore, M.I.T. Press, Cambridge, 1970.

3.- SPACE AND FUNCTION ANALYSIS.

This chapter expands the general principles of the S.A.R. methodology to the description of spaces, the formulation of functional standards and the analysis of their relations.

It describes in detail first, how standards about functions can be defined in terms of a site, a set of elements, their relations and their positions; and proceeds then to present a process that enumerates all the possible alternatives, along the lines of the 'state-space' and 'problem-reduction' methods for generating alternatives.

3.1.- Formulation of Standards:

3.1.1.- Site:

The site constitutes the environment where we place elements according to certain rules. At the scale of functional analysis, the environment is formed by a space or a set of spaces that define a room or an area within a room where a function can be performed.

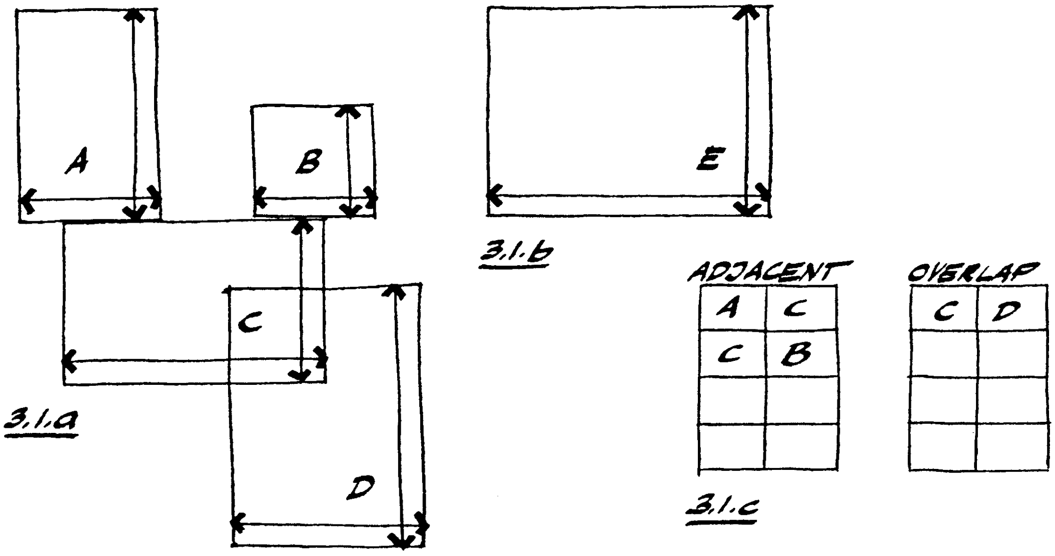
As container for this function, standards about the site can be defined at two main levels: one in terms of what is called the ACTUAL SITE, and another in terms of the FORMAL SITE.

3.1.1.1.- ACTUAL SITE:

The actual site represents the area under consideration, a given existing space or a space being proposed as part of a design. Characteristics of this site can be defined in terms of its SPATIAL or MATERIAL ELEMENTS.

As SPATIAL ELEMENTS we can describe the set of areas that form the total space, and for each of them define their SHAPE, DIMENSIONS and RELATIONS. Shapes in this application, have been restricted to rectangular figure or any combination of rectangular components. figure (3.1a, 3.1b). Dimensions include the length, width and height of all spatial parts. The representation of how these parts fit together to form the total site is done through the description relations between them, such as the adjacencies, overlappings and containments shown in figure 3.1.c.

As MATERIAL ELEMENTS we can describe the physical contrapart that delimits the spatial elements, like blank walls, access walls, windows, etc., and define



3.1.a

3.1.b

ADJACENT

A	C
C	B

OVERLAP

C	D

3.1.c

FIGURE 3.1.

again for each of them, besides its TYPE, their SHAPE, DIMENSIONS and RELATIONS. figure 3.2.

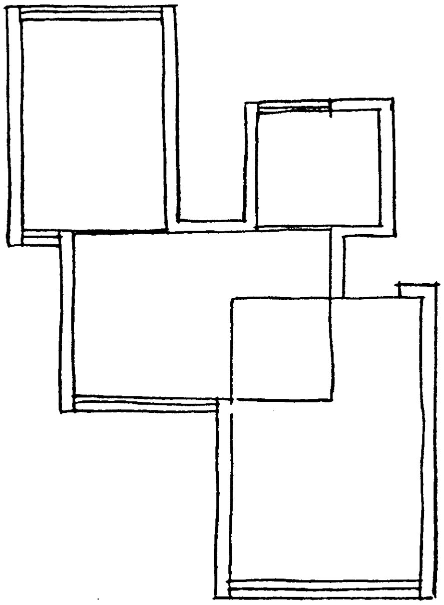


FIGURE 3.2.

The actual site might be thought as the built total space that can contain a function. It can be represented, in the graph notation, as a set of nodes that correspond to spatial and material elements together with their shapes and dimensions, and a set of links that stands for the spatial or material relations that exist between the two kinds of elements. figure 3.3.

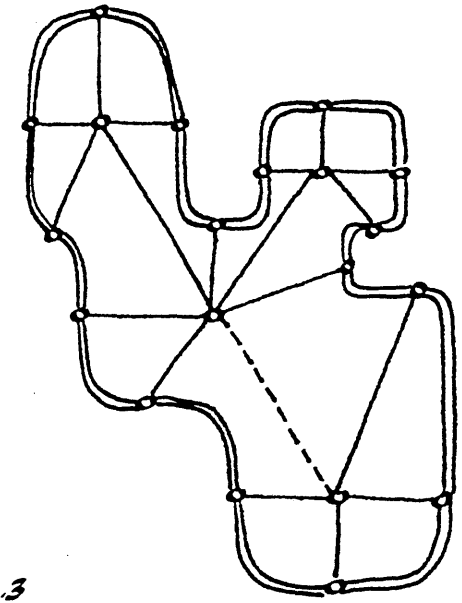


FIGURE 3.3

3.1.1.2.- FORMAL SITE:

The formal site, on the other hand, represents spaces or areas that we might say, do not exist at all in the sense of built space, but are conventions used for the formulation of standards.

As in the case of rooms in the housing situations described with the S.A.R. Methodology, we can look at several functional layouts and notice that among their variety, furniture pieces tend to be grouped in certain ways. They are, most of the time, aligned along the wall in most living spaces, or they are centered in space, surrounded by circulation or operation spaces, in most equipment layouts.

To describe such schemes of agrupation, we can talk, as is done at the housing scale, of ZONES and MARGINS. A system of 'functional' (1) ZONES and MARGINS represents those areas in the actual site, where functional elements are, or can be, positioned.

Through them we can make general statements about possible arrangements. A room, for example, where a function can be accomplished only on one of its walls, lets say a kitchen where furniture and appliances tend to group along the necessary connections, would be represented as the site in figure 3.4.

As an actual site, we can describe its spatial dimensions and physical elements (figure 3.4a). As a formal site we can specify a ZONE and a MARGIN adjacent to the wall 'W1' where kitchen furniture may be positioned. No particular layout has been defined yet, but we have made a general 'site' statement about possible layouts

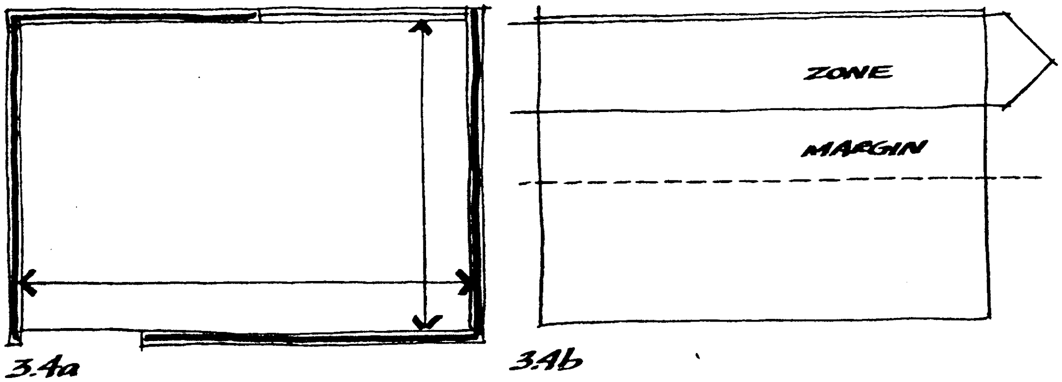


FIGURE 3.4

along the wall 'W1'.

A zone can help us define where elements can be POSITIONED, a margin adjacent to the zone can help us define what are the different lengths that elements can have. If we agree, for instance, that in a system of zones and margins, elements positioned in zones must always end in margins, then the site in figure 3.4., would represent a statement about possible layouts, and represent at the same time a restriction on the DIMENSIONS of kitchen pieces that might fit such a site. (figure 3.5).

Zones and Margins can be used to represent conventions about the POSITION and DIMENSIONS of elements

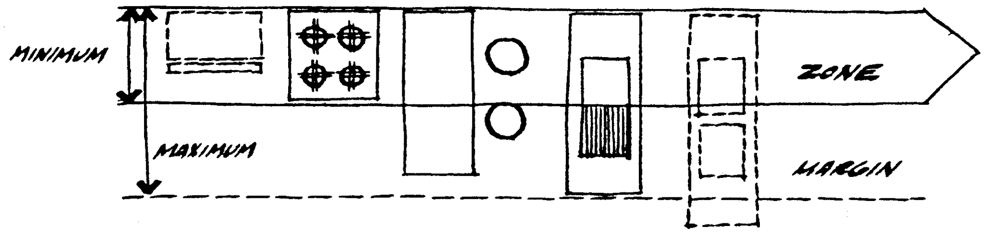


FIGURE 3.5

in one direction.

A system of zones and margins can have different widths, run along one boundary of the space, cross it in the middle, extend across its whole length or width, or simply cover one part of our actual site. (figure 3.6).

To define conventions on positions and dimensions in the opposite direction, we can again use a S.A.R. concept. A SECTOR is a part of a zone and margin which length can be specified. If for instance, in our kitchen example, the wall "W1" has a window and we want to say that some of the furniture pieces must always be in front of the window, then we can break a zone into several parts, one of which has a length that corresponds to the

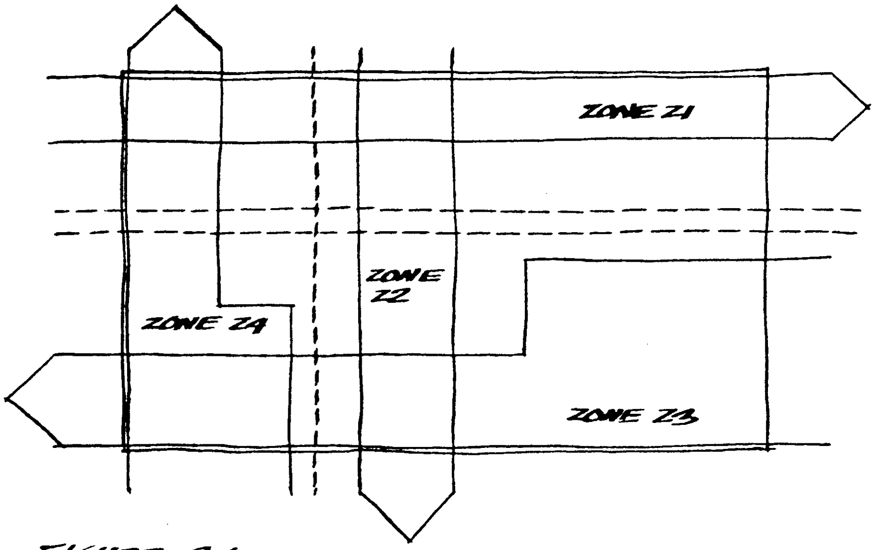


FIGURE 3.6:

length of the window, and stands for the sector where

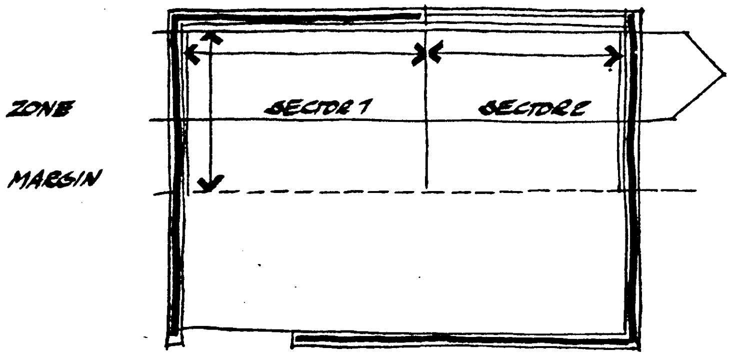


FIGURE 3.7:

those pieces can be positioned. (figure 3.7).

A zone, as a formal construct, has only one dimension: width. The sectors can have both dimensions: length and width. A zone in an actual site can take its length from a combination of sectors, or take it from the actual dimensions of the space where it is positioned. In such case, we can say that a zone with its length and width defined, has only one sector equal to itself.

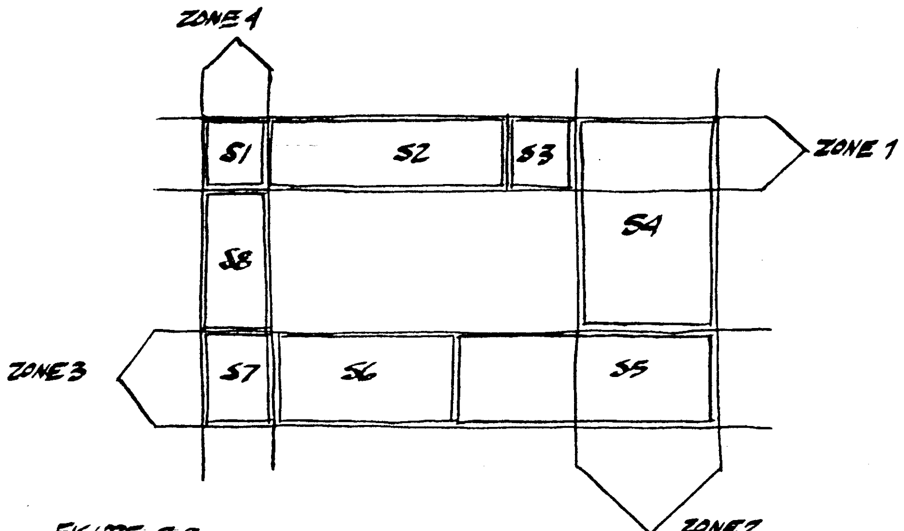


FIGURE 3.8

A system of zones, sectors and margins is called a ZONE DISTRIBUTION. It stands for a set of well defined areas within a room, which can be used to describe general statements on how elements are positioned, and what element layouts are acceptable. (figure 3.8).

3.1.2.- Elements:

The elements are the pieces of furniture or equipment that are needed to perform a function. As was said in the assumptions of chapter 1, a function can be defined in terms of the possible layouts of elements used while performing it.

Elements can be defined in terms of their PHYSICAL UNITS or their USE SPACE, and for each of these certain RESTRICTIONS might be described.

3.1.2.1.- PHYSICAL UNITS:

The physical units of furniture or equipment are the actual material pieces that constitute them. For each element we can describe the SHAPE and DIMENSIONS of its pieces, together with the RELATIONS that exist among them. Similar to the restriction of site pieces, the shape of physical units of an element is limited to rectangular figures or any combination of them. As shown in figure 3.9 elements can have one or several physical units, assembled in different ways, but for each of them we have to define their length, width and height.

3.1.2.2.- USE SPACE:

The use-space is the space that is needed along or around a physical unit to be able to use it. Use-space can be the space we need around a table in order to sit

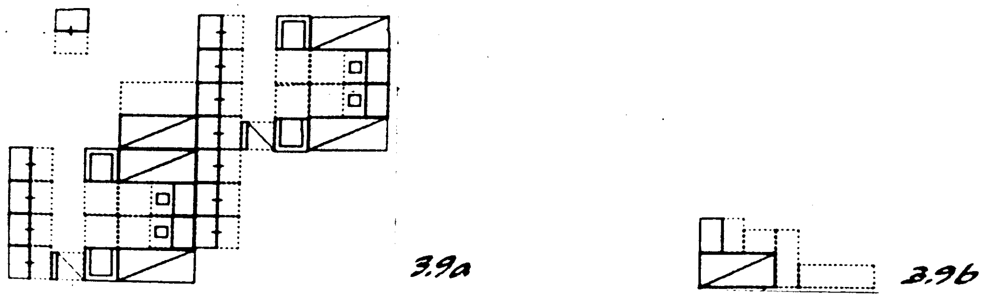


FIGURE 3.9.

down, the free space needed to swing doors open, the space needed to open drawers, or the space that should be left free for equipment parts to move around.

Use-space can be described also as one or several rectangular shapes with DIMENSIONS and RELATIONS. The dotted lines in figure 3.9b delimit use-space rectangles for several pieces. They can be related among themselves to form complex use-spaces, or related to specific physical units through position, or adjacencies.

3.1.2.3.- RESTRICTIONS:

Physical units and use-spaces can be restricted in certain ways. For a given piece of furniture, certain of its physical parts or use-spaces could be overlaped by parts of different furniture pieces.

Independent on how the relation between two elements is stated through the next issue in the definition of standards (i.e. 3.1.3.- Relation between elements), we can specify at the level of each furniture piece, if the piece can or cannot be overlapped by either physical units or use-spaces of other elements. Together with this restriction we can define for each piece if a minimal access side is required and by how much.

3.1.3.- Relations between elements:

Elements can be located related to one another in several ways.

Relations between elements, called here "element constraints', are in theory any specification about relative positions that can be described and that can be tested. In this application, however, only three -in fact, three variations of one- relations are supported:

- ADJACENCY.
- OVERLAPPING.
- CONTAINMENT.

Element constraints relate two elements at the time and express certain conditions that should be satisfied in a functional layout. They can regulate, for example, whether 'element-1' should be adjacent to 'element-2', 'element-2' should be contained by 'element-3', or whether 'element-1' should overlap

•element-4•.

3.1.3.1.- Simple and Compound Element Relations:

Relations can be expressed in two ways, in one we can list a series of •simple• constraints in statements of the form •ELEMENT-RELATION-ELEMENT•, as was done in tre last paragraph.

This is quite straightforwards and in this sense clear and simple. But, on the other hand, it restricts tre formulation of relations to long lists of equally important, all having to be satisfied, constraints which rarely corresponds to the way we think about relations between elements, or the way we might select one constraint over another if we could have the option.

To include trис option, there is a second way in which relations can be expressed: •compound• constraints. •Compound• relations are simple constraints linked by logical connectives •AND• •OR•, and capable of being denied by •NOT•. With them we can formulate relatlons such as:

S1. "•element-1• cr •element-2• or •element-3• should be adjacent to •element-4•, but •element-2• should never be adjacent to •element-1•, neither should 'element-3' be contained by 'element-4'".

Compound relations have a special notation (1), perhaps sufficiently bisarre to obscure their advantage. Instead of saying:

element-1 relation element-2

as we said before, simple statements are expressed now as:

(relation element-1 element-2)

where, inside a pair of parenthesis we define first the relation and then the list of elements that are related by it. So that the statement:

"'element-1' should be adjacent to 'element-2'"

is turned into the statement

(should_be_adjacent_to 'element-1' 'element-2')

or for simplicity

(adjacent e1 e2)

The reasons for this inversion might become clearer, although perhaps not justifiable in terms of a user, if we think that elements in a relation can be in themselves other relations, as the case would be for two statements connected by 'and'. The relation 'and' has two elements: statement-1 and statement-2. So if we want to say: "'statement-1' and 'statement-2'", we say

(AND statement-1 statement-2)

and, if be each statement we substitute simple relations with the same format as

(adjacent e1 e2)

(adjacent e2 e3)

then we have

(AND (adjacent e1 e2)(adjacent e2 e3))

or a compound relatior formed by the binar relation "AND", that is a relation with two elements, each of them a nested binary relation "ADJACENT", with the format:

(relation1 (relation2 e1 e2)(relation3 e1 e2))

(relation1 element1 element2)

Compound element relations use as building blocks, the "ADJACENT", "OVERLAP" and "CONTAINS" binary relations, which can be nested at the bottom of a hierarchy of other binary relations as "AND" and "OR", together with the unary relation "NOT", to form more complx relations like: (S2)

Which corresponds to our previous statement S1 .

A graphical representation of these relations can be visualyzed in terms of a tree, which shows a relation on each node, and for each node two or one

```
S2:     (AND (OR (OR (ADJACENT E1 E4)
                     (ADJACENT E2 E4))
                 (ADJACENT E3 E4))
             (NOT (OR (ADJACENT E2 E1)
                      (CONTAINED E3 E4))))
```

elements as successors of binary or unary relations respectively. The statements in (S1) , and in (S2) , would look like the following figure 3.12, which can help explain the nature of the system of parenthesis.

Or if we don't break our building blocks, but show them as a line statement, it would look like the tree in figure 3.13.

Simple constraints represent one list of relations to satisfy, compound constraints represent several alternatives that can be accepted depending on the combination of connectives that we use.

Statements linked by "AND" have to be both satisfied. Both relation-one and relation-two, represent

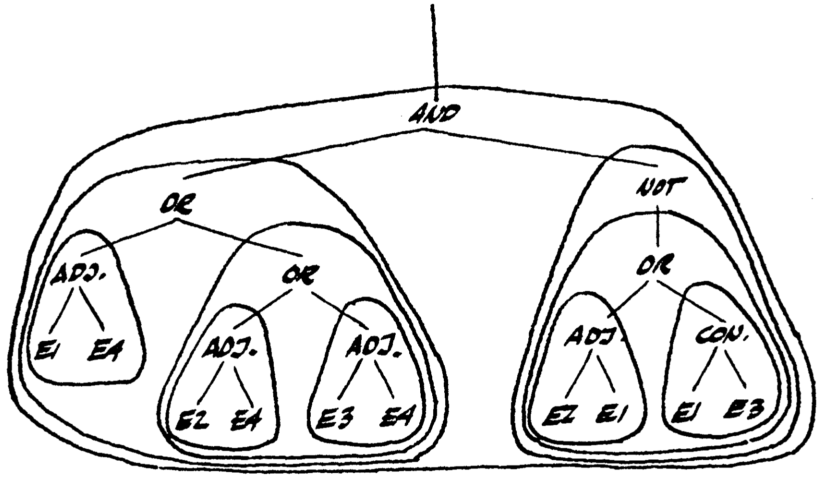


FIGURE 3.12 :

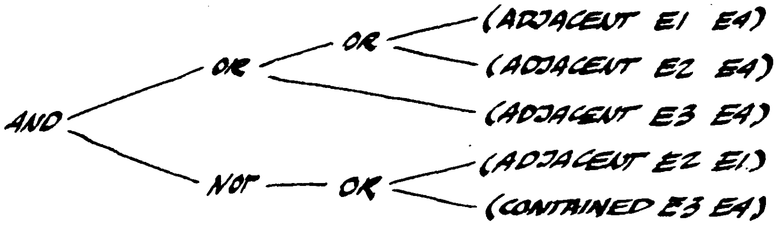


FIGURE 3.13:

constraints that we want satisfied in a functional layout.

Our first formulation of simple statements would be equivalent then to a list of statements linked only by AND's.

Statements that are linker by 'OR' connectives, can be considered in two ways: as 'inclusive-or' or 'exclusive-or'. Inclusive OR's imply that for two statementts, we can satisfy either or the two, or both of them in the final layout. Exclusive OR's imply that if we have one of the two satisfied, then the other cannot be present at the same time.

Statements preceded by 'NOT' simply reverse their final relations.

For our example in figure 3.11, if the OR's are considered as inclusive OR's then, there would be 7 possible alternatives (figure 3.14) which would be accepted as valid combinations if appear in the final arrangement.

If the OR's were exclusive, then only the first three lists would be considered acceptable.

3.1.4.- Position of elements in the site.

The location of elements in the site is defined through 'POSITION RULES'. These rules specify a relation between an element and a site where it can be positioned.

1.- (ADJACENT E1 E4)
2.- (ADJACENT E2 E4)
3.- (ADJACENT E3 E4)
4.- (ADJACENT E1 E4)
(ADJACENT E2 E4)
5.- (ADJACENT E1 E4)
(ADJACENT E3 E4)
6.- (ADJACENT E2 E4)
(ADJACENT E3 E4)
7.- (ADJACENT E1 E4)
(ADJACENT E2 E4)
(ADJACENT E3 E4)

FIGURE 3.14 :

Position rules can relate elements to the site at different levels. Elements can be located simply in the space that forms the actual site, 'room-x', they can be located in zones within the room, 'zone-1', or they can be positioned in sectors within the zones, 'sector-a'.

Similar to the relations between elements, position rules can be expressed as 'simple' position rules, or 'compound' position rules. The simpler relation between an element and its site would be:

element1 is positioned in site1, or

(PUT_ON element1 site1)

in the same notation that we used for element constraints.

The compound position rules will use again the connectives 'AND' 'OR' and 'NOT' to form nested position rules that define several alternative locations for an element, or several alternative elements that a space in the site can contain, like the rule in figure 3.15 and its tree representation (figure 3.16).

```
(AND (AND (OR (PUT-ON E1 ZONE1)
              (PUT-ON E1 ZONE2))
          (AND (PUT-ON E2 ZONE3)
               (PUT-ON E3 ZONE3)))
     (OR  (PUT-ON E4 ZONE1)
          (PUT-ON E4 ZONE2)))
```

FIGURE 3.15 :

3.1.4.1.- Levels of Definition and Expansion of Position Rules:

Even though both express relations, position rules are different from element constraints in several ways:

- First, they relate elements to site, vs. relating elements to elements.

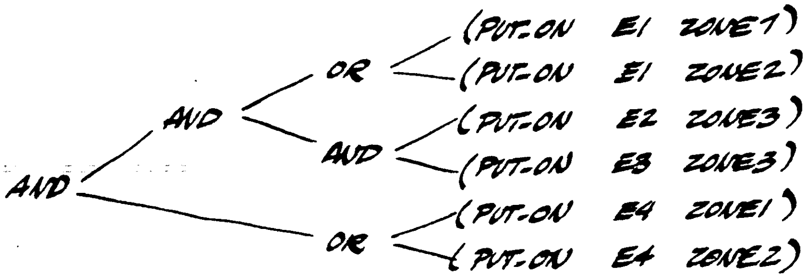


FIGURE 3.16 :

- Second, for a site structured as a space with zones and sectors, a simple position rule (P.R.) can define relations between elements and site at any of these three levels. A compound P.R. can use any combination of these 'building blocks', using the same terms as in

```
(AND (PUT-ON element room)
     (AND (PUT-ON element zone1)
          (PUT-ON element sector1)))
```

element constraints, to describe a position standard as:

To generate all the possible layouts that correspond to this rule, however, we have to know the positions that 'e1' can actually have within 'space1', and the positions that 'e2' can take within 'zone1'. For enumeration, all positions have to be defined at the most detailed level of the site.

In a similar way to the case made for compound element constraints, there is a conflict between how much information we should give in a standard to permit enumeration, and the way we think about positional constraints.

For a position rule, this conflict can be solved in the following way:

- Each P.R. defined at any level of the site implies all the possible p.r.'s that can be formed by relating its element to each of the parts that the site has one level down.

If, for example, a site has two zones: Z1 and Z2, then the p.r.:

(PUT_ON e1 site1)

implies both:

(PUT_ON e1 Z1)

(PUT_ON e1 Z2)

- As the rule is defined generally, i.e. site, vs. specifically, i.e. Z1 or Z2, then we can assume that either of the two positions is valid, and we can proceed to link them with an "exclusive-or", to form the new rule:

```
(OR (PUT-ON  E1  ZONE1)
    (PUT-ON  E1  ZONE2))
```

which simply says, if we want "e1" positioned in "site1" then "e1" positioned in any of the two parts of "site1" would be accepted as a valid solution. If each zone, in turn, has two sectors, for example: Z1 has S1 and S2, and Z2 has S3 and S4, then the rule would turn into:

```
(OR  (OR  (PUT-ON  E1  S1)
          (PUT-ON  E1  S2))
     (OR  (PUT-ON  E1  S3)
          (PUT-ON  E1  S4)))
```

which again assumes that "e1" in "S1", or "e1" in "S2", or "e1" in "S3", or "e1" in "S4", would all be valid positions.

By automatically expanding a rule from one general level to its constituents in the following levels, we can avoid having to define each and everyone of the possible positions that an element can have. Our original rule at the beginning of the section, can be expanded then

```
(AND (OR (OR (PUT-ON E1 S1)
             (PUT-ON E1 S2))
         (OR (PUT-ON E1 S3)
             (PUT_ON E1 S4)))
     (AND (OR (PUT-ON E2 S1)
              (PUT-ON E2 S2))
          (PUT-ON E3 S1)))
```

into:

```
(AND (PUT-ON E* SITE)
     (NOT (PUT-ON E2 S1)))
```

A position standard can be expressed also as:

where if e* stands for all our elements, lets say: 'e1' and 'e2'. It means: "put all the elements in any place of the site, as long as element 'e2' is not positioned in sector 'S2'", and by a similar procedure as we did before, the expression e* is first expanded into all the elements in the problem definition:

(PUT_ON e1 site)

(PUT_ON e2 site)

linked then by "AND", because we want all

```
(AND (PUT-ON E1 SITE)
     (PUT-ON E2 SITE))
```

satisfied

```
(AND (AND (PUT-ON E1 SITE)
          (PUT-ON E2 SITE)
     (NOT (PUT-ON E2 S1)))
```

and we have the resulting rule:

which for a site with two zones, Z1 and Z2, two sectors S1 and S2 in Z1, and only one sector S3 in Z2,

```
(AND (AND (OR (OR (PUT-ON E1 S1)
                  (PUT-ON E1 S1))
              (PUT-ON E1 S3))
          (OR (OR (PUT-ON E2 S1)
                  (PUT-ON E2 S2))
              (PUT-ON E2 S3)))
     (NOT (PUT-ON E2 S1)))
```

would be converted into:

3.1.4.2.- Position Rules in Overlapping Zones:

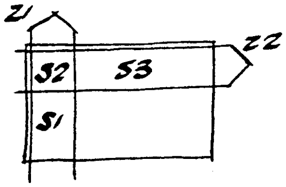
A third difference between p.r.'s and element constraints results from the site having two or more overlapping zones where one sector is shared by both areas.

If position rules were always simple and always defined at the level of sectors, this would represent no problem, as we would have to list all the elements that go in each sector. If we allow compound rules at several levels, however, then we have to define a way to find out which element goes where.

For example, a problem with the following definition:

Site: Z1 with S1 and S1, Z2 with S2 and S3,

Elements: e1 and e2,



```
(AND (OR (PUT-ON E1 Z1)
         (OR (PUT-ON E2 Z2)
             (PUT-ON E3 Z1)))
     (OR (PUT-ON E2 Z2)
         (PUT-ON E3 Z2)))
```

Position Rule: does not define which elements can be positioned in sector S2. If we think of P.R.'s as

DISCLAIMER

Pages has been ommitted due to a pagination error by the author.

describing subsets of elements that can be positioned in a site then,

Z1 can have (e1 or e2 or e3)

Z2 can have (e2 or e3)

and if we expand the rule into its sector

```
(AND (OR (OR (PUT E1 S1)
             (PUT E1 S2))
         (OR (OR (PUT E2 S1)
                 (PUT E2 S2 ))
             (OR (PUT E3 S1 )
                 (PUT E3 S2 ))))
     (OR (OR (PUT E2 S2)
             (PUT E2 S3))
         (OR (PUT E3 S2)
             (PUT E3 S3 ))))
```

definition:

then the subsets for each sector would be:

S1 can have (e1,e2,e3)

S21 can have (e1,e2,e3) from Z1

S22 can have (e2,e3) from Z2

S3 can have (e2,e3)

where S2 is undefined because it has two different subsets of elements that can be positioned in it. Having these subsets, however, we can decide a convention on how to position elements in overlapping zones. If elements in S2 can only be those that appear in both zones Z1 and Z2, then the intersection of S21 and S22 defines the position rules for S2.

S21 = (e1,e2,e3)

S22 = (e2,e3)

S2 = (e2,e3)

```
(AND (OR (PUT-ON E1 S1)
         (OR (OR (PUT-ON E2 S1)
                 (PUT-ON E2 S2))
             (OR (PUT-ON E3 S1)
                 (PUT-ON E3 S2 ))))
     (OR (OR (PUT-ON E2 S2)
             (PUT-ON E2 S3))
         (OR (PUT-ON E3 S2)
             (PUT-ON E3 S3 ))))
```

and the position rule would be:

To expand the rules of intersecting zones, we can proceed then as we did in 3.1.4.1. but for each overlapping sector, we have to check first for the intersection of rules, and select those positions that satisfy this test.

3.2.- Standards:

A set of functional standards is defined by a system of elements and relations. The elements correspond to the spatial parts of both site and furniture pieces. The relations correspond first, to element constraints that regulate relative locations, and second, position rules that regulate absolute locations of an element in the site.

In a more formal manner, a set of standards consists of a 4-tuple:

(S, E, R, P)

where:

- 'S' is the set of spatial, actual and formal, parts that form the environment.

- 'E' is the set of spatial, physical-units and use-space, parts that form the furniture pieces.

- 'R' is the set of desired relations, simple or compound, between elements in 'E'.

- 'P' is the set of desired positions, simple or compound, that relate elements in 'E' with elements in

'S'.

A set of stanoards implies a set of possible graphs L , which are formed by nodes that correspond to elements in 'S' or 'E', or both; and are linked by edges that correspond to relations in 'R' or 'P', or both.

A functional layout, or a furniture variant, is one of the possible graphs in L, where the links correspond to one of the desired combinations of relations in 'R', together with one of the desired combinations of positions in 'P', for a given S,E,R,P.

To evaluate a set of standards, the subset L* of L, which contains all the functional layouts, has to be enumerated.

3.3.- Enumeration:

For the set L* we do not have a list of members, but we have instead a criteria for membership. We do not know a priori what are the possible functional layouts that a standard can have, but we have a criteria for judging when a layout is a valid or an invalid furniture arrangement.

To enumerate L* then, we have to construct all the possible graphs in L which qualify as functional layouts according to this criteria. From the set L of possible graphs, our "solution space", we have to extract all the variations that are members of L*.

To do this, we need rules that partition the solution-space into different "chunks" where solutiors might exist, and equally important, we need rules that reject, as soon as possible, "chunks" that do not contain any solution at all. (figure 3.17)

Not having these rules would mean having a situation where all the points in the design criteria have the same importance, and therefore, all have to be analized to the same level of detail, checking all the combinations and variations of this criteria on each configuration.

When these rules can be defined, we can express through them the structure of our solution-space. We can construct or reconstruct whatever the case might be, entire portions of this space whenever this becomes necessary. We can state, through them, the possibility of a layout, or its validity in terms of some conditions.

At different levels of detail, we can construct layouts one at the time, and check that some conditions are satisfied in order to know if the next, more developed, layouts are worth looking at. We can systematically look for members of L* needed to evaluate a standard.

figure 3.17:

3.3.1.- Overview:

The S.A.R. formulation of standards, with its parts and relations, provides a way for expressing these rules.

The generation of furniture variants can be carried on by sequentially constructing a solution or 'graph', where we add one element to the site according to the position rules, and we check at each step the satisfaction of the element constraints between the positioned elements.

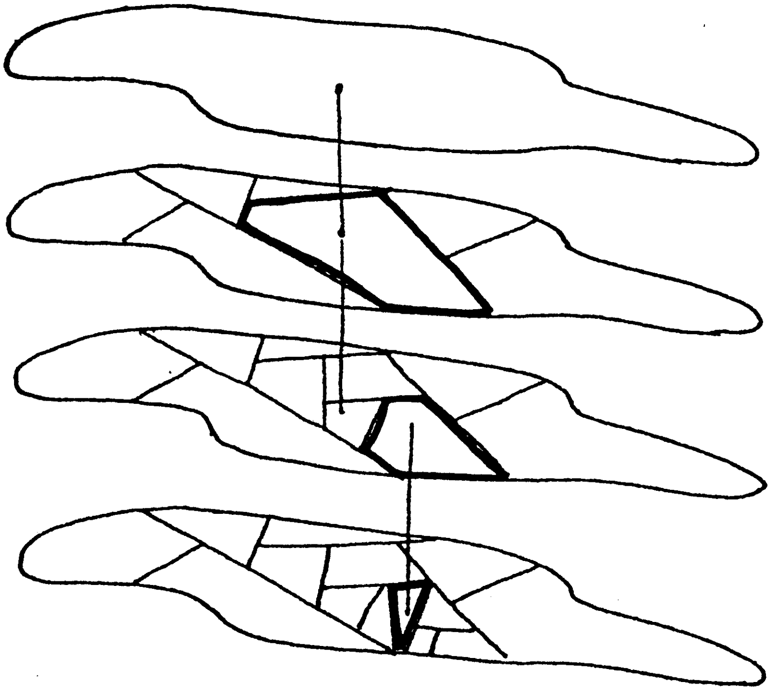


FIGURE 3.17

At the most simple level, this generation can be carried on as a 'depth-first' search, and the enumeration of layouts can be represented as a 'State-Space' model where:

- the graph being constructed represent our 'state-descriptor',

- the position rules constitute the 'state-operators', and

- the relations between elements are the criteria against which states are tested, as shown in

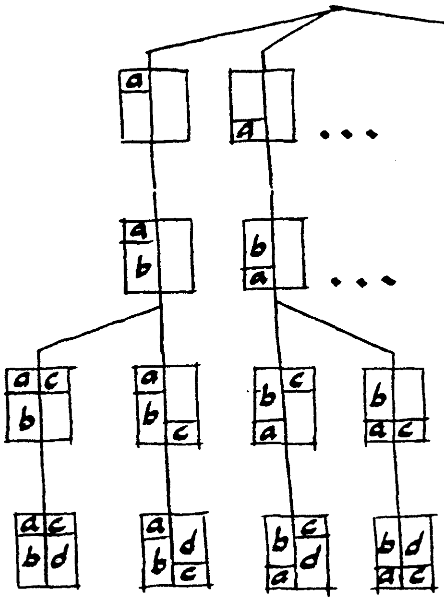


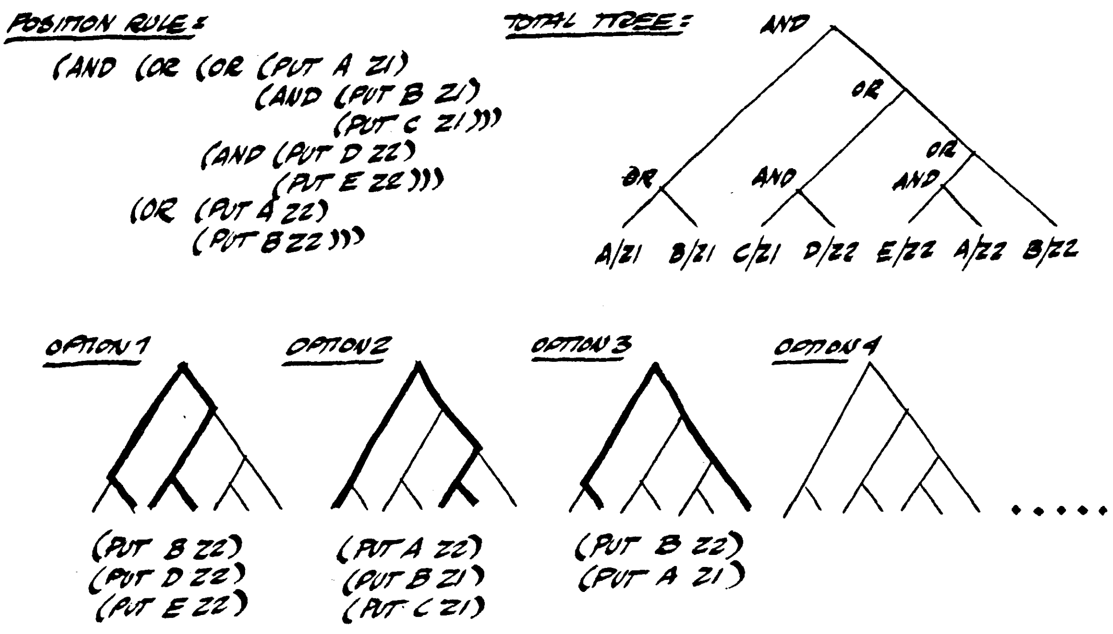
figure 3.18:

figure 3.18:

At a higher level, the compound position rules can be used to decompose the problem into different subproblems which can make the search simpler. Each alternative combination of position rules in compound position rules, can be considered as a separate problem with several subproblems expressed as state-space descriptions, as shown in figure 3.19:

figure 3.19:

3.3.2.- Description of the process:



POSITION RULE:

```
(AND (OR (OR (PUT A Z1)
             (AND (PUT B Z1)
                  (PUT C Z1)))
         (AND (PUT D Z2)
              (PUT E Z2)))
     (OR (PUT A Z2)
         (PUT B Z2)))
```

OPTION 1

```
(PUT B Z2)
(PUT D Z2)
(PUT E Z2)
```

OPTION 2

```
(PUT A Z2)
(PUT B Z1)
(PUT C Z1)
```

OPTION 3

```
(PUT B Z2)
(PUT A Z1)
```

FIGURE 3.19

The process for generating L* breaks down the task of exhaustive enumeration into a hierarchy of smaller problems with different levels of complexity. The description of this hierarchy will be done, first in a quick outline of the problem reduction steps, and second, in a search for functional layouts presented through a detailed example. Generalizations and definition of terms will be made along the way as it becomes necessary.

3.3.2.1. Problem reduction:

The solution space of a functional standard, can be constructed through the application of two kind of rules:

- GENERATION RULES

- TEST RULES

GENERATION rules produce alternatives or partition the solution space into subsets that may contain solutions. TEST rules check the existence of solutions in those partitions produced by GENERATION. Generate and Test, through 'operators' and 'constraint criteria', systematically expand and preclude regions of the solution space.

Generate rules are of two different kinds, corresponding to the two levels, simple and compound, that position rules can have:

- TRUTH TABLE, and
- PERMUTATION OF ELEMENTS

For compound position rules, we can explore the different alternative position that are acceptable for an element (figure 3.19) through the construction of a TRUTH TABLE, as explained in 3.3.2.2.. For simple position rules, we can explore the different locations an element can take within its zone or sector through the PERMUTATICN OF ELEMENTS, or the variations in absolute positions.

Test rules are also of two different kinds:

- POSITIONAL, and
- DIMENSIONAL

These are operations that check the POSITION and DIMENSION of elements in a site, as regulated by a functional standard.

Positional Tests are the:

- EVALUATION OF POSITION RULES, and the
- EVALUATION OF ELEMENT CONSTRAINTS

Absolute positions of elements can be tested by the EVALUATION OF POSITION RULES, and relative positions can be tested by the EVALUATION OF ELEMENT CONSTRAINTS.

Dimensional Tests include tests for:

- ZONE DIMENSIONS
- SECTOR DIMENSION
- MARGIN DIMENSION

and check the size of an element against the width of a zone, as in ZONE DIMENSION, against the length of a sector, as in SECTOR DIMENSION, and against both the length and width of a margin, as in MARGIN DIMENSION.

An important, both dimensional and positional, constraint is the CIRCULATION between elements in the site. It can be defined either by absolute position if assigned to be in a certain zone, or it can be defined by relative position if assigned to be through the different use-spaces or remaining margins in a given layout.

These operations have a preference order between themselves. For instance, we can not attempt a permutation of elements in a zone until we know what are the position rules that assign such elements to the zone. If these position rules are compound, we have to decide first what valid alternative location of elements we will try, before doing any permutations or changes.

Once such locations are known, we have to check the dimensions of the elements against zones and sectors, to find out if that location can be, in fact, occupied by them or not. Only then we can permute elements we know can have valid positions and valid dimensions, and check while constructing these different arrangements, that the relative positions are being satisfied, that the margins can hold . all the elements in the adjointing zones, and that the overall circulation pattern is respected.

The different levels into which the enumeration task is broken down, are then in order of importance:

1.- TRUTH TABLE.

2.- EVALUATION OF POSITION RULES.

3.- PRECLUSION.

4.- ZONE DIMENSION.

5.- SECTOR DIMENSION.

6.- PERMUTATION OF ELEMENTS.

7.- MARGIN DIMENSION.

8.- EVALUATION OF ELEMENT CONSTRAINTS.

9.- CIRCULATION.

corresponding to the expansion or pruning operations as shown in figure 3.20.

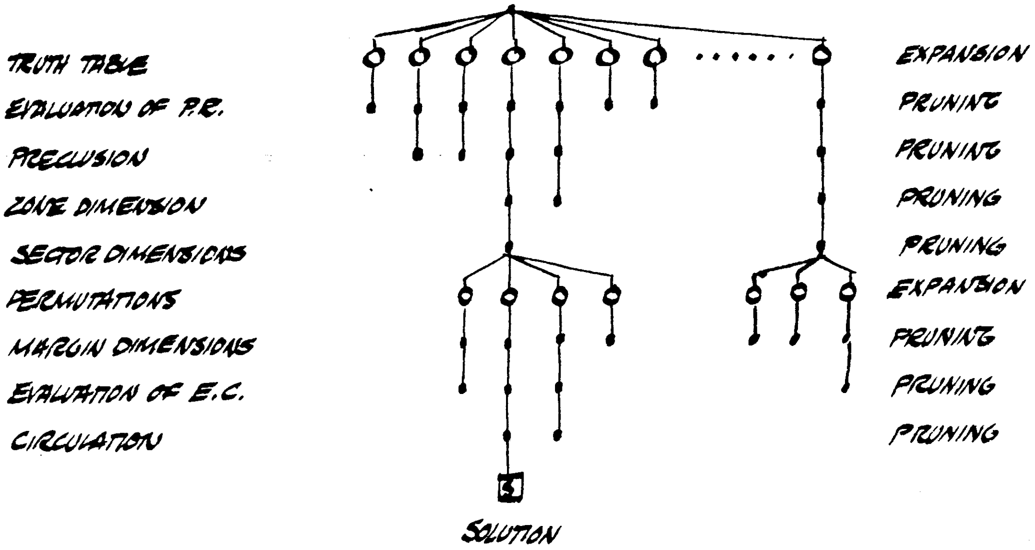
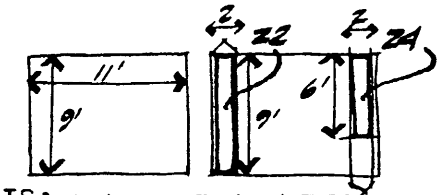


figure 3.20:

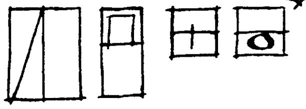
EXAMPLE:

The following example will be used to describe how these operations interact to enumerate all the possible layouts for the standard:

SITE:



ELEMENTS:



```
RELATIONS: (AND (OR (ADJACENT DESK  BED.PHYSICAL UNIT)
                    (ADJACENT DESK  CHAIR))
                (ADJACENT DESK CLOSET))

            (AND (OR  (PUT-ON BED  Z2)
POSITIONS:            (PUT-ON CHAIR Z2)
                 (OR  (PUT-ON BED Z4)
figure 3.21:          (AND (PUT-ON CLOSET Z4)
                           (PUT-ON DESK Z4))))
```

The first operation to be applied to start the enumeration process would be then:

3.3.2.1. Truth Table:

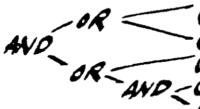
If the solution space stands for all the functional layouts that a given standard can have, then the first partition that we can make corresponds to the possible alternative position rules that are implicit in a compound rule.

To do this, we can consider each 'building block' in a compound rule, as a simple relation that is or is not satisfied in different alternative position rules. To each simple relation, we can assign a 'value', lets say TRUE or FALSE, according to whether or not we decide to have these positions satisfied in the region of the solution space that we want to explore.

All the different combinations of values that these relations can have, represent all the subdivisions that can be made out of a solution space at the general level of position rules. Without having explored yet any actual layout, we decide first what alternatives should be pursued among the different permited by the position rules.

Compound statements can be TRUE or FALSE depending on whether the combination of values for each simple position rule, represents a valid or an invalid position rule.

The subdivision of the solution space into alternative combinations of values can be expressed then by a TRUTH TABLE, that assigns TRUE or FALSE values to each of the simple rules in all the possible combinations.



(Put bed 22)	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
(Put chair 22)	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
(Put bed 24)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
(Put closet 24)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
(Put desk 24)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

FIGURE 3.22

figure 3.22:

In our example (figure 3.22), there can be 32 of these possibilities. The position rule is represented by a horizontal tree on the left side, and the Truth Table is represented by a matrix where each single rule appear as a row that can take the values True or False, 0 or 1, and the different possibilities appear as columns that cross along alternative values for each row.

Truth Tables are 'binary counters' insofar they enumerate or 'count' with True or False, 0 or 1 values, all the alternatives for a compound statement. As can be seen in figure 3.22, each column represents a number from 0 to 31 in binary. As such, and for large compound rules, there can be a 'counting problem', that is, each new simple rule added to the compound, increases the number of alternatives from 2 to 2. So for one simple relation there are two values, for a compound relation with two simple relations there are four values, for a compound relation with three simple relations there are 8 values, and so on; running into the enumeration or 'counting' of large numbers very easy.

For the time being, this problem has been kept in mind but no solution has been implemented to reduce this generation of alternatives. One possibility could be to direct the assignment of values towards those combinations must likely to produce valid position rules, starting our 'counting' from the first valid combination, such as column ' ' in figure 3.22. How to find out these valia combinations is the problem in EVALUATION OF POSITION RULES.

3.3.2.2. Evaluation of Position Rules:

Only some of the partitions for compound position rules are valid combinations that interest us. These are combinations of simple relations that have a TRUE value for the compound statement. To find out these alternatives, we EVALUATE each of the columns in the TRUTH TABLE in the logical sense.

As expressed in 3.1.3.2, the connectives that tie together simole relations into compound statements have a definite meaning:

- for each, 'AND', the two elements in the relation have to be 'true' to have the whole relation

AND	element1	0	1	0	1
	element2	0	0	1	1
		0	0	0	1

evaluated to 'true',

- for each exclusive 'OR', either one of the two elements have to be 'true' to make the compound statement 'true'.

OR	element1	0	1	0	1
	element2	0	0	1	1
		0	1	1	0

- for each inclusive 'OR', one of the two or both elements being 'true' produces a 'true' compound relation.

OR	element1	0	1	0	1
	element2	0	0	1	1
		0	1	1	1

- for each 'NOT', a 'false' element makes a 'true' relation and viceversa.

		0	1
NOT	element1	1	0

'AND','OR','NOT', are evaluated then according to these simple tables. When several 'AND','OR' or 'NOT's are nested in compound statements, we first find out the values for the 'lower' relations in the hierarchy, pass then the resulting values as values of the elements for the next relation up, and continue doing so until we reach the final relation, and have the whole statement

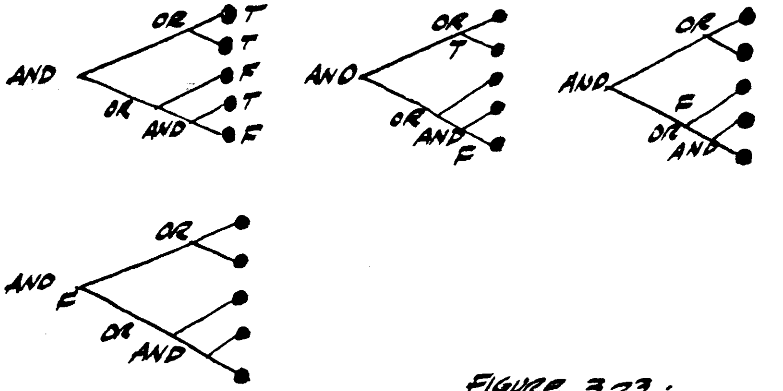
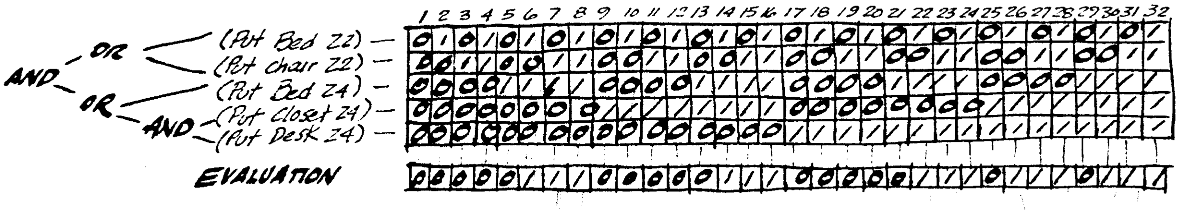


FIGURE 3.23 :

evaluated, as shown in figure 3.23.

figure 3.23:

In the truth-table generated for our standard example of figure 3.21, the position rules that evaluate to 'true' are only columns: 26,27,28,30,31 and 32, as shown in figure 3.24. Only these combinations of simple relations represent valid alternatives of the compound position rule at the left side of the table, and only these combinations make any sense to continue exploring for possible different layouts. If we think of the columns in the matrix as branches going out of our tree root, we can preclude then from further consideration all



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
(Put Bed Z2) —	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
(Put chair Z2) —	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
(Put Bed Z4) —	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
(Put Closet Z4) —	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
(Put Desk Z4) —	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EVALUATION	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	1	0	1	1	1

FIGURE 3.24

the regions that extend down those paths.

figure 3.24:

Through compound relations we can decompose a problem into the different possible locations for the elements. Through the assignment of truth values, we can explore all the possible decompositions that can be made for each problem. Through the evaluation of these values we can decide which of the alternative positions should be considered valid and continued being explored.

3.3.2.3. Preclusion of repeating elements:

Moving in our example in a left-to-right manner across the different position alternatives, we would look now into branch 26, as compound statement evaluated to be true.

This statement and all the rest that have passed our previous test, are checked now for PRECLUSION of repeating elements.

As can be better seen in brach 30, jumping a little ahead, there are some cases when compound rules can be evaluated to true, but assign two times the same element, in here "bed", to different positions in the environment: zone Z and zone Z2. This repetition of positions for one element cannot be, obviously, accepted. An element can not be in two places at the same time. Even though evaluated to TRUE, this compound statement makes no sense when interpreted as a real position rule.

The test for valid branches with repeating elements, would preclude then columns, or branches, 6, 8, 14, 16, 22, 24, 30 and 32, from further considerations and reduce the search for functional layouts to branches, 7, 23, 26,27,28 and 31, as shown in figure 3.25.

figure 3.25:

3.3.2.4. Zone Dimensions:

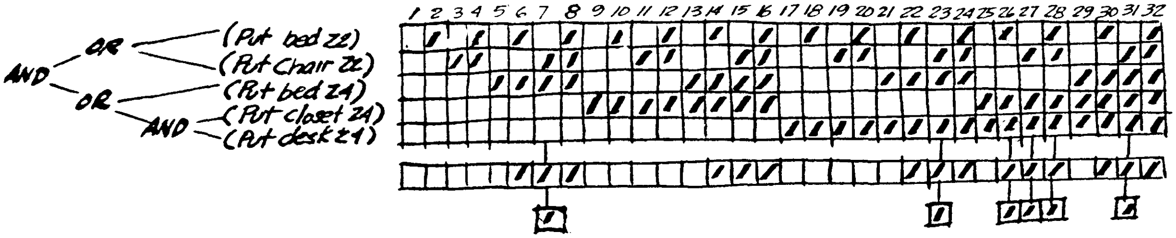


FIGURE 3.25

After selecting one combination of position rules, valid and without repeating elements, there is only one part in the site where each element can be positioned. The position of elements is assigned to only one of the possible spaces in the environment, and we have to check now if dimensionwise this assignment is correct.

Elements can be positioned in zones or sectors if, first there is a rule that defines so, and second, there is an agreement on how the dimensions of element and site should be considered. If, for instance, elements are only allowed to end in margins, then we have to check now that at least one of the element dimensions -length or width- is equal or larger to the width of the zone where it is going to be positioned, and equal or smaller than the width of both zone and adjoining margin.

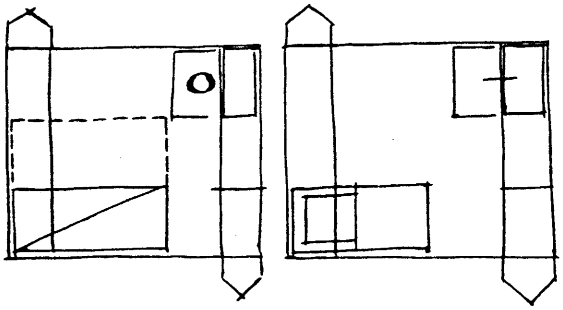


figure 3.26:

For our example, both zone Z1 and Z2 are suficciently small to contain any of the four elements in any position. So this test is passed by all the remaining branches as shown in figure 3.27.

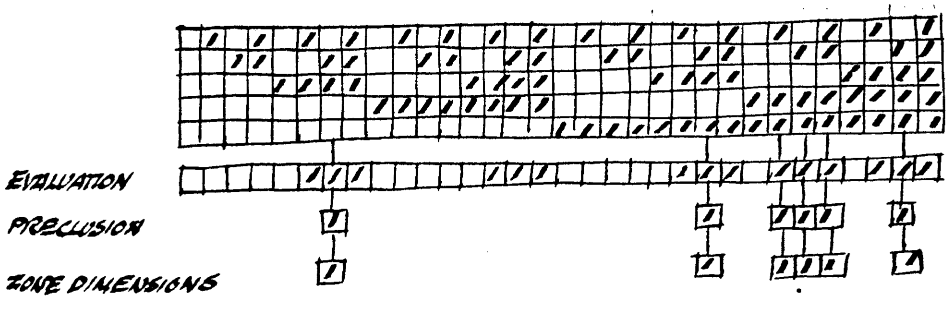


FIGURE 3.27

figure 3.27:

3.3.2.5. Sector Dimensions:

Valid positions in zones have to be checked now along the other dimension: length. Elements, we know, can be located in zones Z1 and Z2, for branches 26,27,28 and 31, without repetition and fiting within the width of both zones and margins.

The test for Sector Dimensions, checks if all the elements assigned to a zone can also fit along its length, or along the length of the sector where they have been assigned if this would have been the case.

Sector Dimensions checks that the sum of lengths or widths, depending on how they are positioned, of all the elements in a zone/sector does not exceed the length of such part of our site.

figure 3.28:

When the sum of lengths or widths of all elements is smaller than the corresponding dimension of the zone or sector, the difference is occupied by an empty space with that length or width.

As a convention, this space is treated as one entity. It is not broken down into several empty spaces between elements but appears as one unit that can be

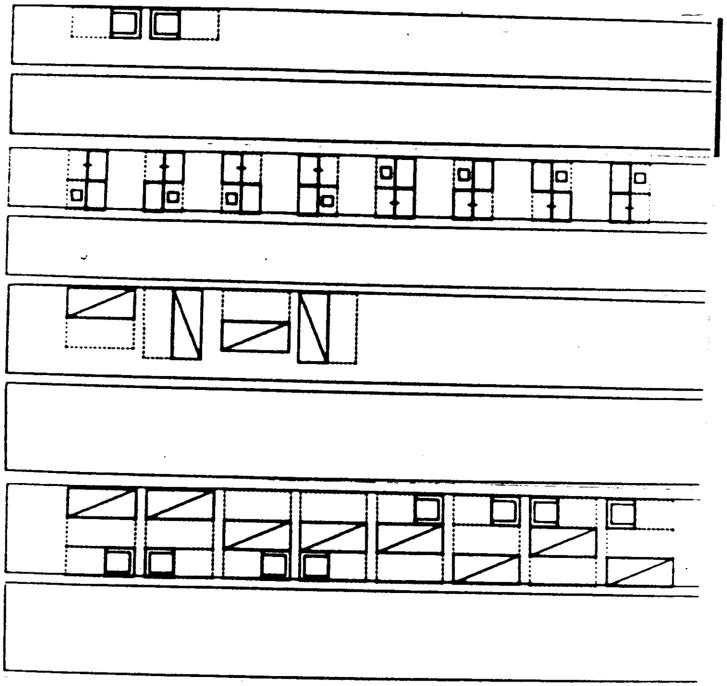


FIGURE 3.28

changed in position but keeps always its dimension.

Under this test, branch 23,31 is excluded from further expansion, but all the rest continue as valid

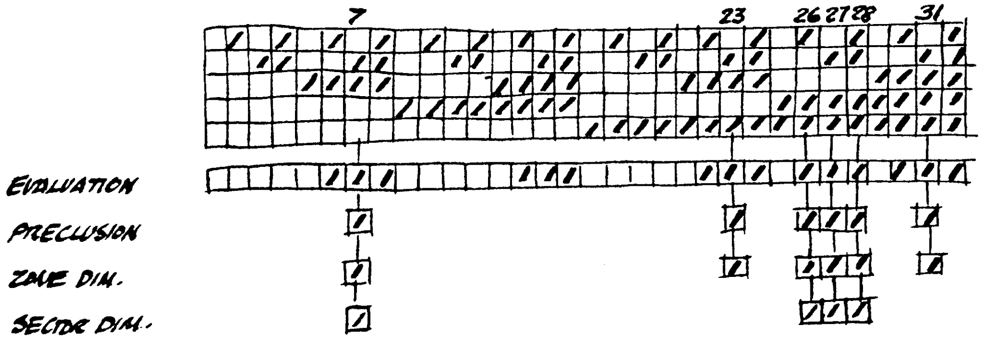


FIGURE 3.29

options where furniture layouts might exist.

figure 3.29:

3.3.2.7. Permutation of Elements:

For each of these options, as was said before, we have SORTED each element in the rule to only one valid and possible space in the site.

For branch 26, this assignment of elements to site would be:

Z2 with closet and desk.

Z4 with bed.

This sort present two interesting characteristics:

1.- It produces a CLASS of furniture layouts.

2.- It permits sorting the element constraints into GLOBAL or LOCAL constraints that can be used for pruning criteria.

1.- We know that as far as zones and sectors are concerned, that is without considering margins, we have already a valid furniture layout, which schematically can be represented as:

This layout satisfies one alternative position rule, its elements fit in the width of the zones where they have been assigned, and they also fit the length of the only sector that each zone has.

If we forget for a moment that elements within the zones can switch positions, we can say we have already found a furniture variant. If on the other hand, we accept that each element can vary its location in the zone, we can say then that we have found a CLASS of furniture layouts. We have found, indeed, a region in the solution space where several arrangements share the same position rule and are validly assigned to the same sector or zones of a site.

The different arrangements in this CLASS are formed by the permutations of the element locations in each zone. Zone Z2 can be, for instance, either:

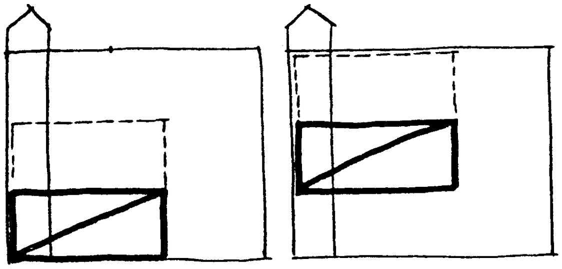


FIGURE 3.30

and Zone Z4 can be either:

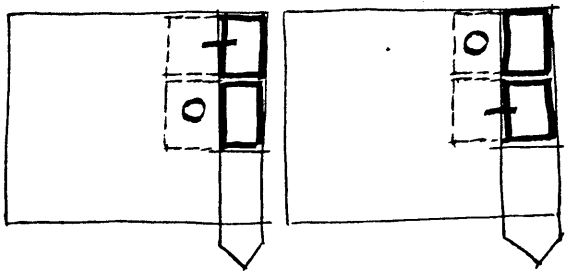


FIGURE 3.31

The combination of these different locations, generates several equivalent arrangements that have the same elements in the same zones, and that constitute and EQUIVALENCE CLASS of furniture layouts in terms of the relation 'position'.

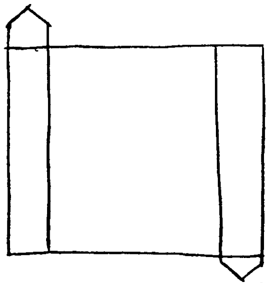
Exploring our solution space from 'top-to-bottom', we have partitioned the set of all possible layouts into EQUIVALENCE CLASSES where layouts are grouped by similarities. By 'merging' our arrangements into branches *re. 2.2.4.2) that can be tested without positioning yet any furniture piece at all, we reduce the exponential explotion we could have had, had we started putting the bed in the lower corner of Z2 then tried to put the desk in the upper corner of Z4, and so on, for each possible combination.

To enumerate the layouts in each equivalence class, we have to construct now all the permutations of elements in the site. We build a combinatorial tree, or in terms of our State-Space representation, we model our class by an initial state and a series of rules that can generate all the equivalent furniture layouts.

For our example in branch 26, the equivalence class would be generated by the following representation:

1.- The 'state-descriptor' is the formal site plus the positioned elements.

2.- The initial state is the empty site.



3.- The goal state is the site with all the elements positioned.

4.- The state-operators are the list of simple

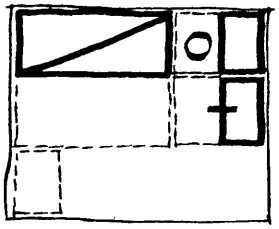


FIGURE 3.32 :

prosition rules with 'true' values in the Truth Table column that we are exploring:

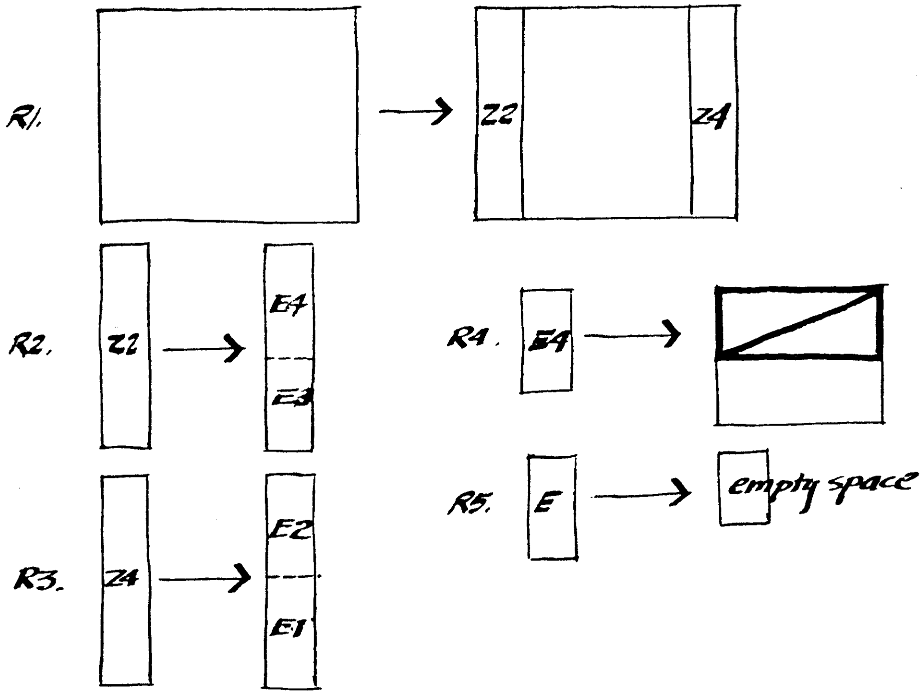


FIGURE 3.3. : (part 1)

Where the operators simply state that a layout should be formed by the two possible arrangements of elements in zones Z2 and Z4. That an arrangement in Z4 should be formed by two elements E1 and E2 either of which can be a desk or a closet, and that an arrangement for Z2

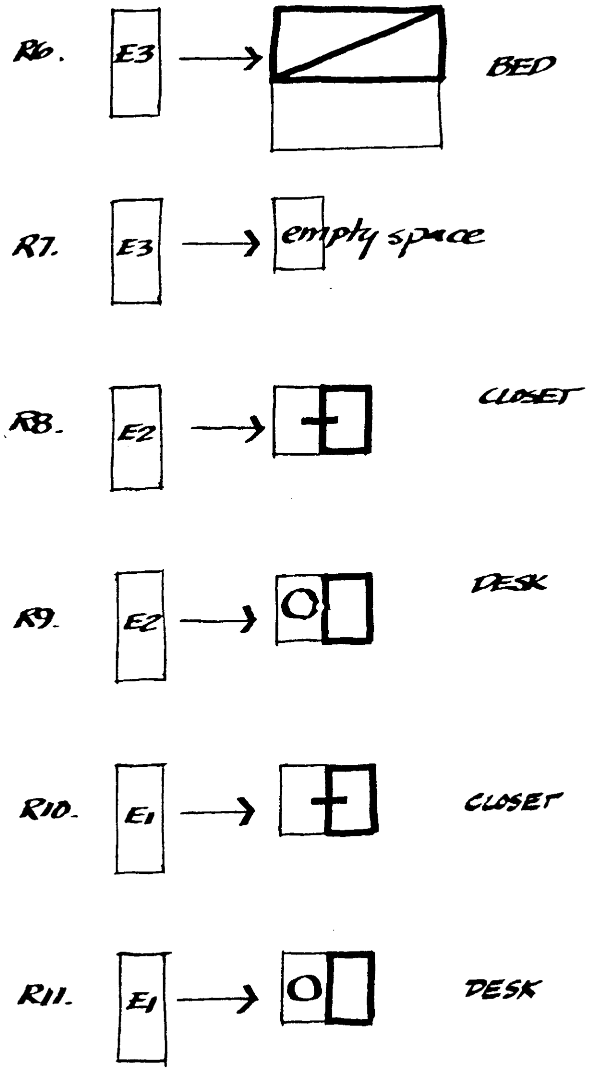


FIGURE 3.33: (part 2)

should be formed by two elements E3 and E4 which can be either a bed or an empty space.

Applying these operators to the initial layout, first E1 then E2 then E3 and then E4, we generate the following 4 layouts as shown in the bottom of our tree in

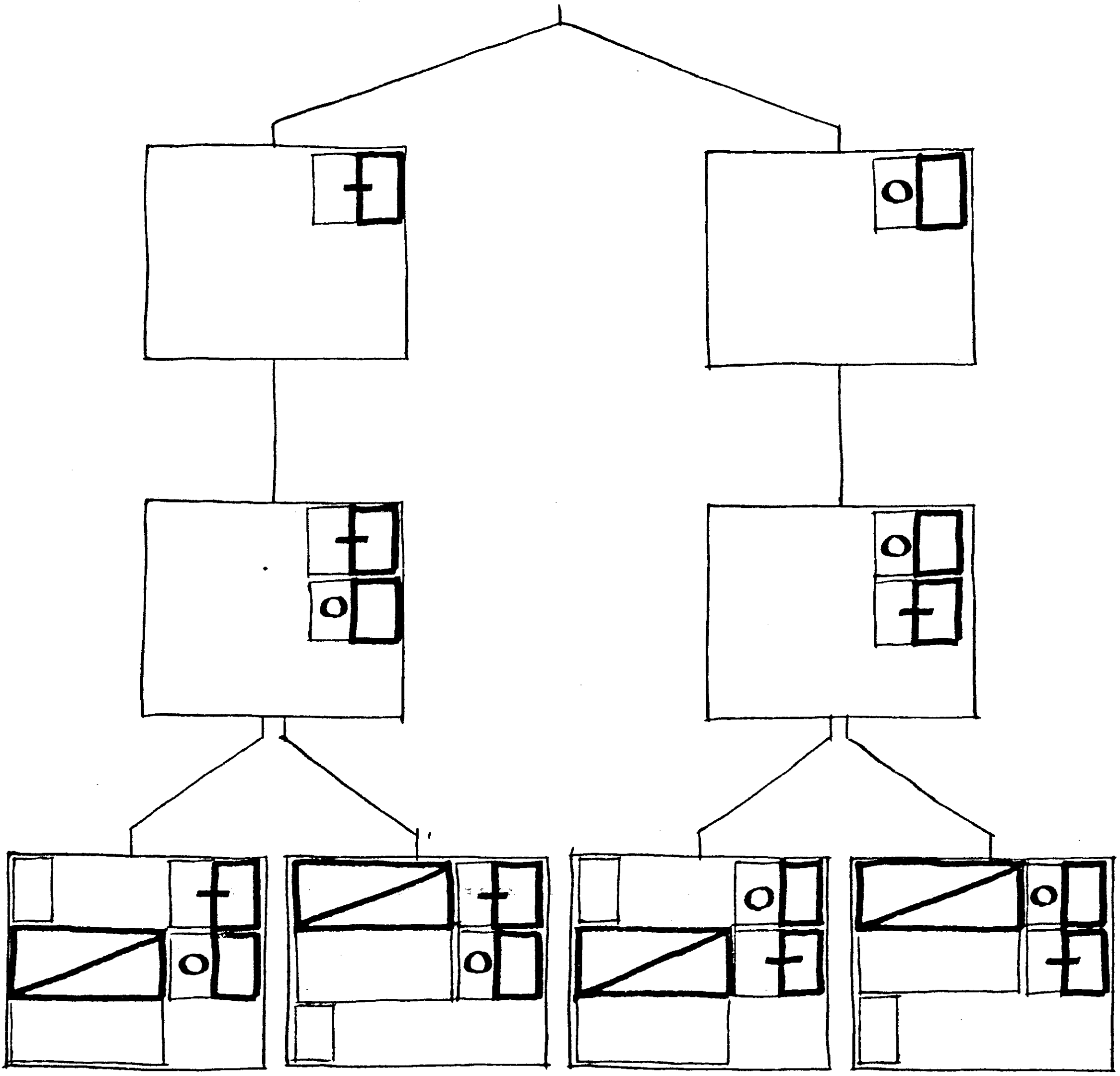


figure (3.34).

Elements are positioned here always in the same way, however, elements can be positioned differently within the same zone. The bed for example could be

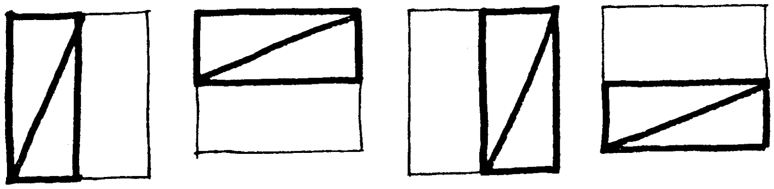


FIGURE 3.35

assigned to Z4 as: which corresponds to its four 90 degrees rotations. From SECTOR DIMENSIONS, we know that this piece is smaller than its site (Z4), and ther is a remaining empty space. Therefore we can decide on any of these positions to appear in the furniture layout, and

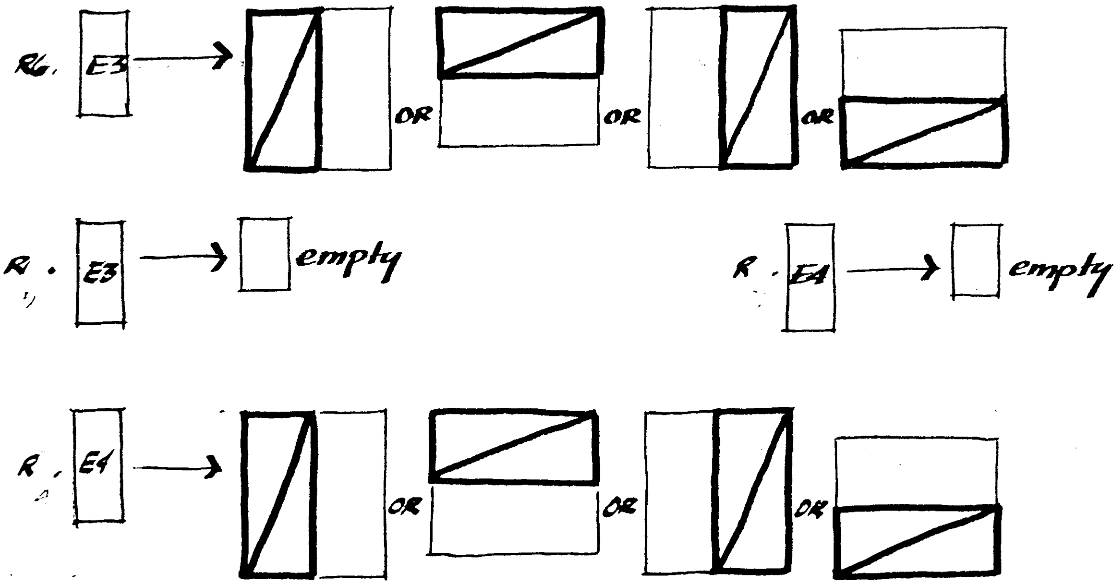


FIGURE 3.36

change the operators E3 and E4 into

The elements desk and closet, on the other hand, fit exactly in zone Z2, therefore they can only be rotated

180 degrees, which keeps their dimensions along the zone

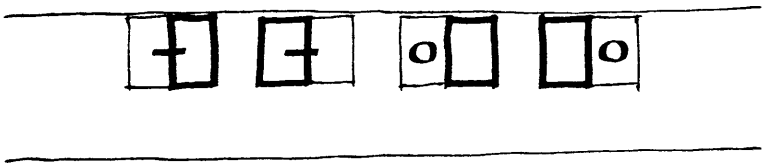


FIGURE 3.37:

the same

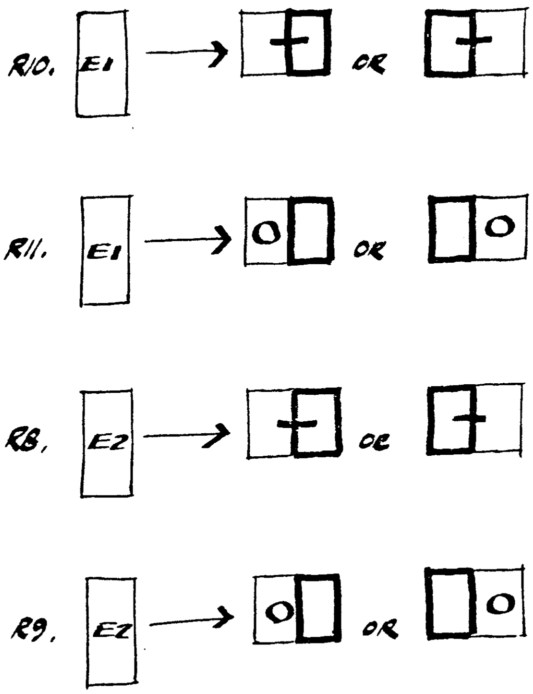


FIGURE 3.38:

changing the operators E1 and E2 into

Which could produce a combinatorial tree like the one partially represented in the following picture, generating 64 possible layouts with similar positions.

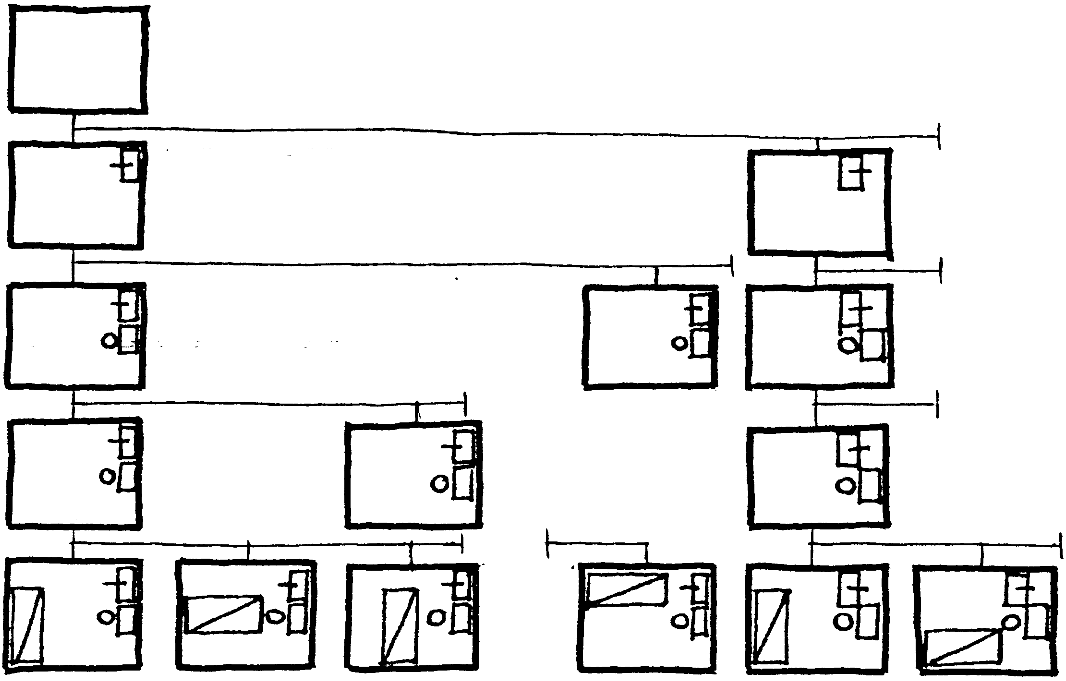


FIGURE 3.39:

2.- Classes of furniture layouts help us also sort the element constraints into Global or Local constraints. If we think of a furnitdre layout as a room arrangement formed of different arrangements at the zone-sector level, then we can break down our previous State-Space representation into the following model:

room level:

- state descriptor = same,

- initial state = same,

- goal state = same,

- state-operators = assignment of zone arrangements generated by:

zone level:

- state descriptor = Z2,

- initial state = empty Z2,

- goal state = complete Z2,

- state-operators = assignment of elements to zones as:

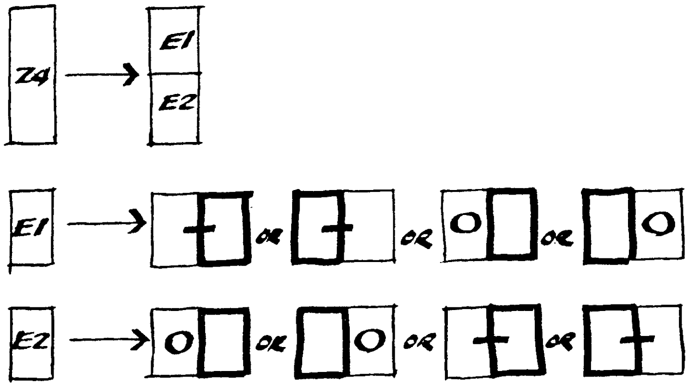


FIGURE 3,40a :

zone level:

- state descriptor = Z4,

- initial state = empty Z4,

- goal state = complete Z4,

- state-operators = assignment of elements to zone as:

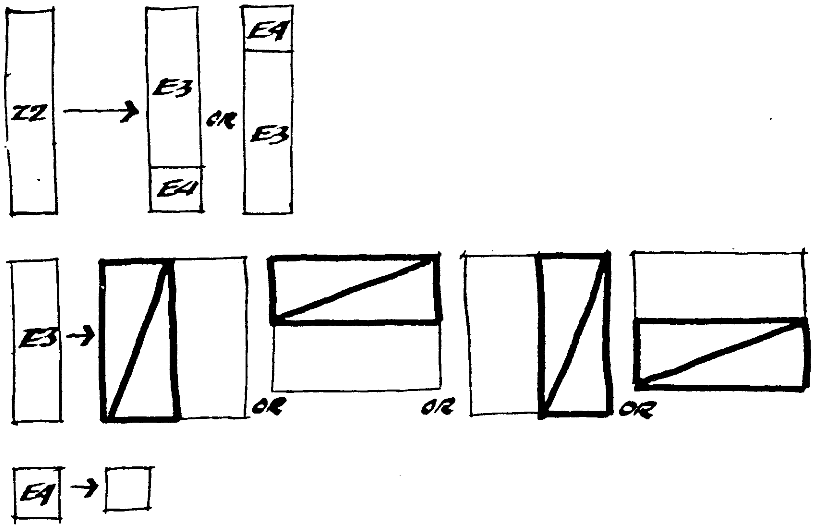


FIGURE 3.40b:

which would produce the following room state-operators:

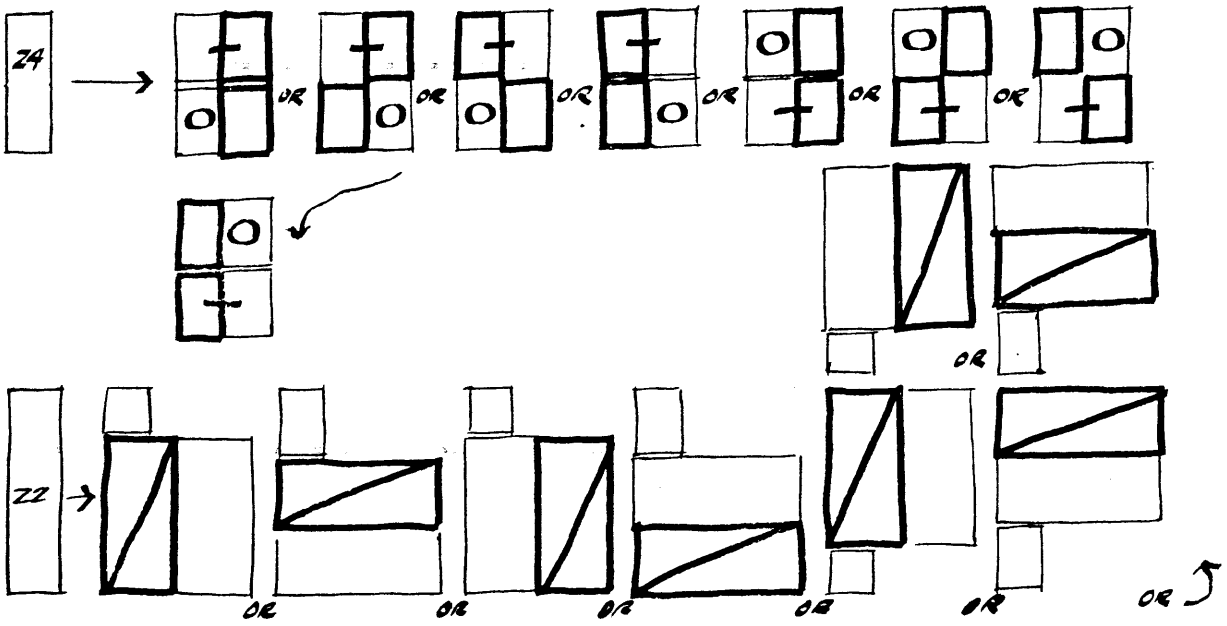
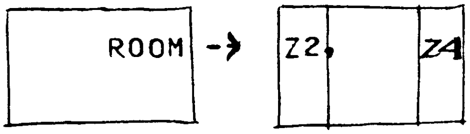


FIGURE 3.40c:

In the generation we would proceed first to apply one of the operators at the room level, i.e.:



but in order to do this we would have to find out first an arrangement at zone Z4 which can be used as this operator, therefore we have to construct it by

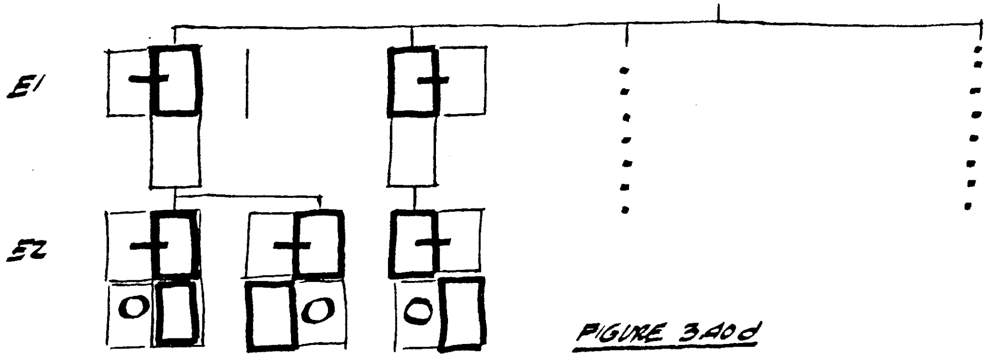


FIGURE 3.40d

applying the operators at the zone level: (3.40d)

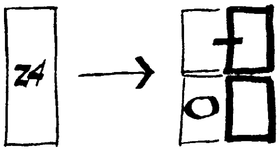


FIGURE 3.40e

which produce: (3.40e)

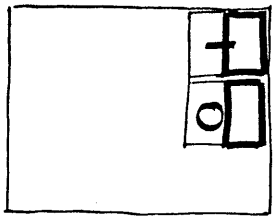


FIGURE 3.40f

that we can apply to form: (3.40f)

and continue with Z2 in a similar way, first with: (3.40g)

to get: (3.40h)

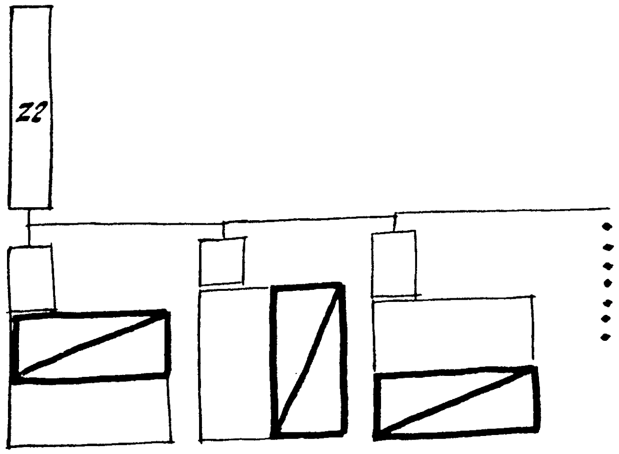


FIGURE 3.40g

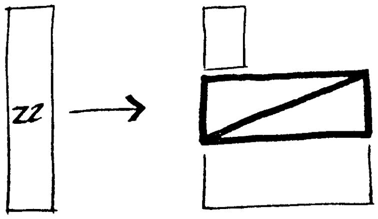
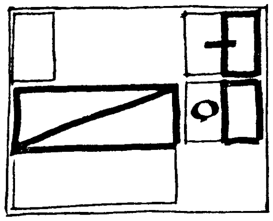


FIGURE 3.40h

and then: (3.40h)



to form:

This representation of NESTED State-Space descriptions, where the result of one search produces the operators for the next search one level up, produces the same equivalence class, and permits us to sort the constraints in the following way:

A GLOBAL CONSTRAINT relates elements in different zones or sectors,

A LOCAL CONSTRAINT relates elements in the same zone or sector.

As will be seen in the section EVALUATION OF ELEMENT CONSTRAINTS (3.3.2.9), the constraint criteria for the first State-Space at room level would include those relations that apply between elements in Z2 and Z4, while the constraints in the second State-Space would include those relations that apply to element in Z2 and for the third State-Space those relations that constraint element positions in Z4.

These sorted Element constraints, with MMARGIN DIMENSIONS and CIRCULATION are the remaining tests that can help us prune branches in our exploration of the Solution Space.

3.3.2.8. Margin Dimensions:

When two or more zones share a margin between them, the elements that can be positione in each zone, might overlap portions of the margin if their dimensiors are larger than the width of the zone.

As the functior of a margin is precisely to allow the position of elements with different widths, when a furniture piece extends beyond the width of its zone it occupies a portion of the margin.

When two elements in opposite zones end within a common margin, conflicts might occurr: if the sum of both overlappings is smaller or equal than the margin then the elements fit, if the sum is larger than the margin dimensions then the elements overlap. For overlapping elements we have to check if this overlapping is permited or not.

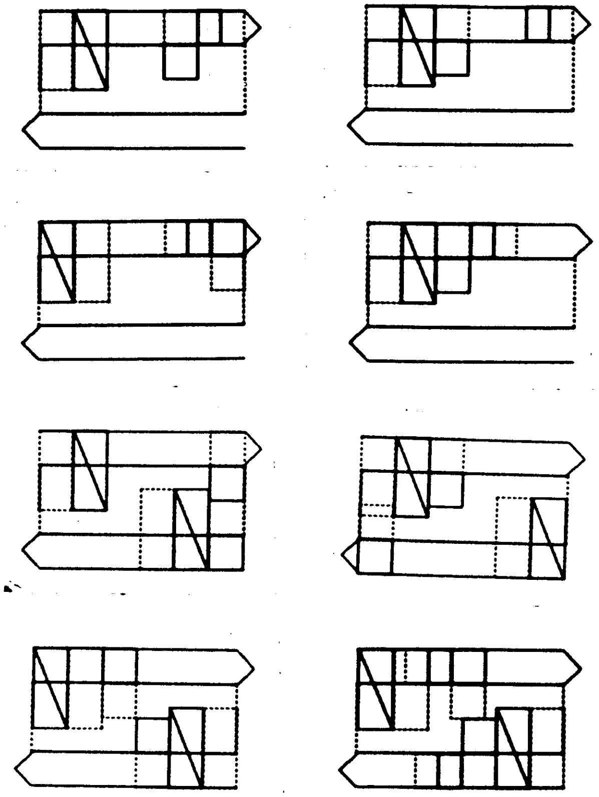
As the generation of layouts proceeds at any of the levels, room, zone or sector, everytime we assign an element to its position, we test the margin dimensions against previous arrangements to see if there is a conflict that stops the search from going any further.

In branch 7 for example, only two arrangements would pass the test while in branches 26,27,28 all the 64 possible would pass it withou any conflict in case we continue our search all the way down to the bottom of our tree, passing the tests of EVALUATION OF CONSTRAINTS and CIRCULATION.

3.3.2.9. Evaluation of Element Constraints:

Element constraints, like position rules, are expressed by compound statements which can be TRUE or FALSE, depending on particular combinations of TRUE-FALSE values in their simple components. Different from position rules, however, the assignment of these values is not done

through a 'binary counter', or a decision on which alternatives to explore, buth through the evaluation of actual, present, relations.



If an element is positioned in the site then it either satisfies some relations to other existing elements or not. If it does, the relations have a value TRUE, if a defined element constraint is not satisfied among the present elements then the relation is set to FALSE.

The evaluation of compound element constraints is done like the evaluation of position rules, from the bottom-up, that is from simpler relations to compound statements as in figure (3.25).

If the relations, however, can be evaluated only when all the values are set to TRUE or FALSE, then we have to wait for a complete layout, but if a layout is generated through a series of nested State-Space representations, we can break the constraints as was said before, so that we can evaluate each representation without having to wait for the results in any other zone or sector arrang .

In branch 26, this would mean that the relation

```
( AND (OR (adjacent desk bed)
          (adjacent desk chair))
      (adjacent desk closet))
```

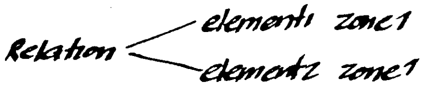
have to be broken down into the following element constraints:

room: (AND (OR (adjacent desk bed)
                  (adjacent desk chair))
             (adjacent desk closet))

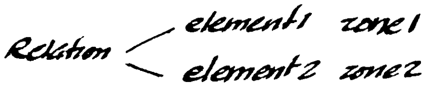
zone Z2: (adjacent desk closet)

zone Z4: none

We can do this by applying the transformations



LOCAL

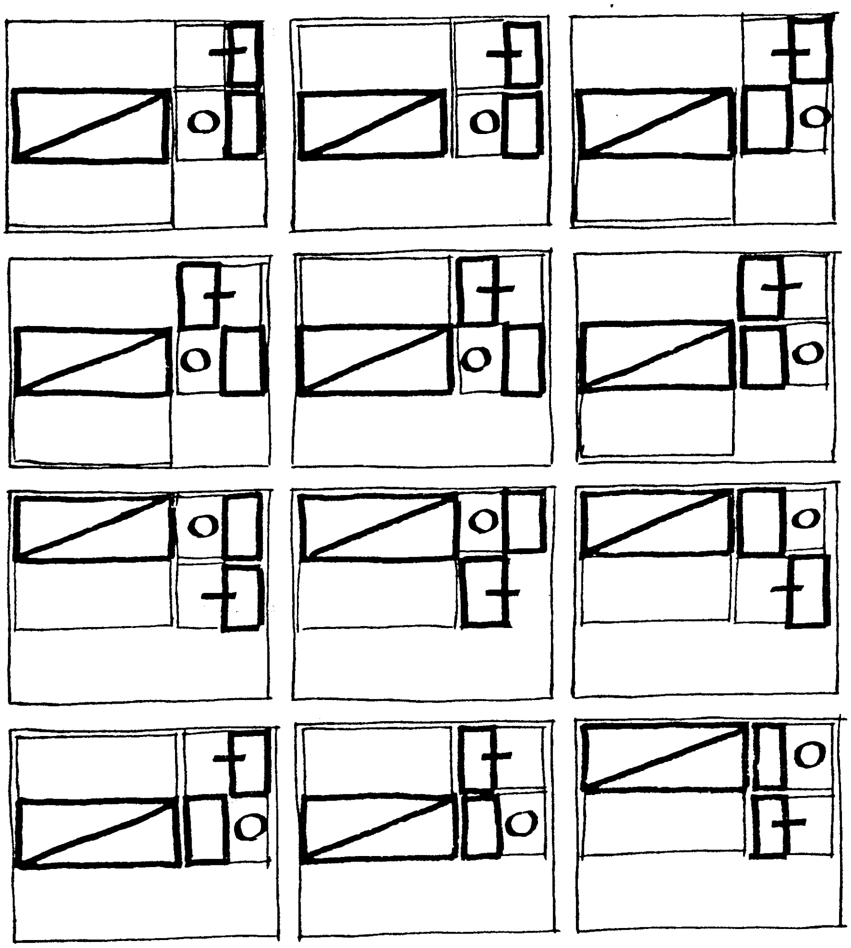


GLOBAL

shown in the next tables:

which reduce our tree of element constraints into several trees, each one corresponding to the relation that have to be satisfied at the level of the site.

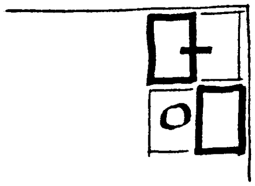
pruning away all those combinations in branch 26 that do not satisfy the constraints and reducing the possible layouts to 12, if there were no further tests from the original 64 that we could have had in figure



3.39.

In this case the State-Space of zone Z2 always produces valid arrangements because its two elements are always adjacent, and therefore preclusion comes at a global level, between X2 and Z4 in the relation of adjacency bed-desk. When a piece is not positioned, like the case of the chair, in our example branc 26, then we do not evaluate that constraint, it is assumed that the position rules have priority over the element constraints.

As long as we can have 'true' values we proceed with our search, when we don't we stop. True constraints, however, still have problems because the desk 'adjacent'



to the closet as in: blocks the access to its use space. Therefore, after checking element constraints we have to check for CIRCULATION.

3.3.2.10 Circulation:

Each element we positioned has to have acces to its use space. IF this access is defined as a spatial element that is located in a zone, then we treat it as any other element, and specify the relations that it should have with the furniture pieces it will serve.

If the circulation is defined simply as access to every use space withour any particular specification, then we have to check that there is a chain of use-spaces, or leftover spaces through which this access can be solved.

This last test corresponds to the second case. Everytime we position a furniture piece we check for this path. If we think of the layout being constructed as a graph, finding a circulation path is then a problem of finding a 'spanning-tree' for that graph. A spanning tree is precisely a path that goues through some of the links but visits all of its nodes. Our circulation path has to be a series of spaces use-spaces, margins or leftover spaces that are linked bh adjacencies of a certain minimum. that we can walk around, and that should allow us to reach each piece of furniture in the room.

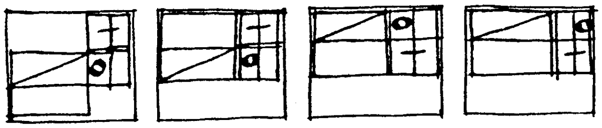
Finding a spanning tree for a graph is a well solved problem with several alforithms that can be used. (2).

We apply this as a prunning criteria in the following way:

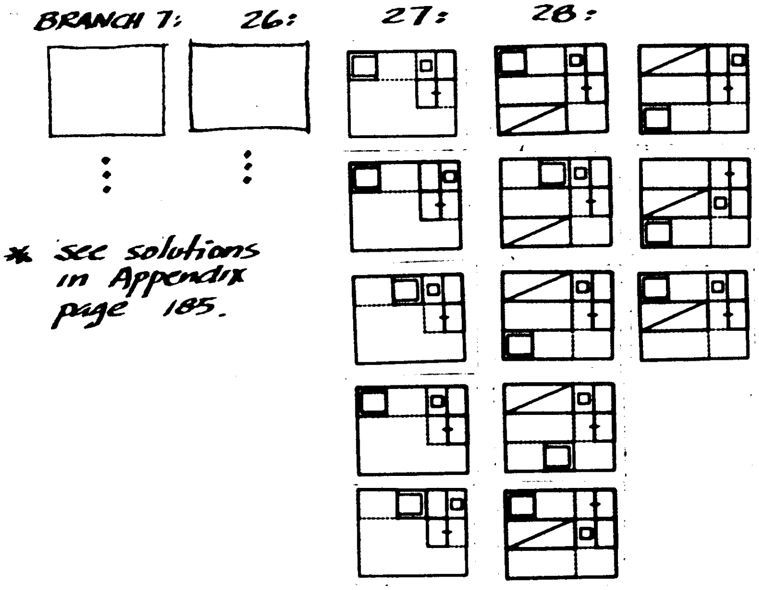
-everytime an element is added, we construct a path or spanning tree for it, if we succeed we have a valid circulation.

-If we fail we stop any position of elements in our combinatorial tree.

With this test our possible 12 layouts for brancn 26, come down to 4 which represent our basic furniture variants for one case of position rules, and which are the end of our long search.



When we apply this complete procedure to all the other branches 7, 27 and 28, we end up with the 19 basic fdrniture layouts that our standard permits.



* see solutions in Appendix page 185.

4. COMPUTER SYSTEM:

A computer system was implemented to carry on the process defined in the last chapter. It can be used as an independent 'furniture shu fler', or be incorporated as the operation of SPACE and FUNCTION ANALYSIS in the computer programs being implemented at M.I.T. for the generation of Basic Variants at the Housing Level by M. Gerzso.

From the user point of view, the system appears as having two main parts: one corresponding to the definition of standards, and another corresponding to the process of querying this informations, querying first for existing relations such as sizes of furnitdre, adjacencies in the site, et ., and second for implicit configurations or furniture variants.

Internally it is organized in four modules:

- a front end 'SARCASM', or user interface of the basic variants program, written by M. Gerzso and M. Gross.

- a Relational Data Base, 'RDB',

- a RDB set manipulation routines, "QUERY LANGUAGE",

- and the SEARCH programs,

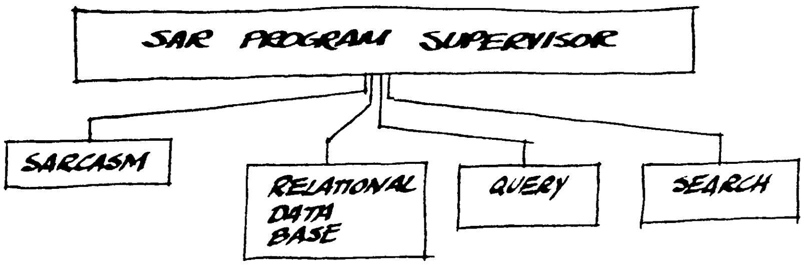


FIGURE 4.1 :

4.1. Relational Data Base:

The standards formulation, the information needed during the enumeration process and the resulting configurations are stored in a RDB(1) which constitutes the main bank of information for the system. There is only another representation for rules used in the generation process. (in the Permutation of Elements).

A small, in core, RDB was written specifically for the furniture shuffler and the S.A.R. Basic Variants program.

In this kind of data base, information is stored as entities and their relations. As is quite obvious now, after the description of S.A.R. and the enumeration process, both our standards and our 'state-descriptors' are basically that: entities and relations. For example,

```
SITE = S1, S2
ELEMENTS = E1,E2
POSITION RULES = (AND (OR (PUT-ON E1 S1)
                          (PUT-ON E1 S2))
                      (OR (PUT-ON E2 S1)
                          (PUT-ON E2 S2)))
ELEMENT CONSTRAINT = (adjacent E1 E2)
```

FIGURE 4.2:

	SITE
1.	S1
2.	S2

	ELEMENTS
3.	E1
4.	E2

POS. RULES	
3	1
4	1
3	2
4	2

ADJACENT	
3	4

FIGURE 4.3:

the simple function: FIGURE 4.2

would be represented in our RDB as: FIGURE 4.3

Where we have lists of entities, one for the site entities, another for our elements entities, and we have a list of relations, in this case binary relations, one for the position rules, the other for the adjacencies.

SITE

1,	S1
2,	S2

ELEMENTS

3.	E1
4	E2

POS-RULES

3	1
4	1
3	2
4	2

ADJACENT

3	4

STATE-DESCR.

3	2
4	1

FIGURE 4.4:

A RDB consists of a general reoresentation for information, it provides a way of oefining lists of entities and lists of relations. What we put in those lists is up to us. We can input a standard as we did before, or we can input a state-descriptor, during our search process, as: (FIGURE 4.4)

Where we can keep track of elements that have been positioned in the site as E1 and S1.

This general representation is concerned with the logical structure of the data, rather than its actual contents, we can change our descriptions as we please, include as complex formulaticns as desired, retrieve partial relations or construct new lists out of existirg ones.

Precise description of this structure can be made through relational algebra or relational calculus, together with a set of operations which can be applied to retrieve or

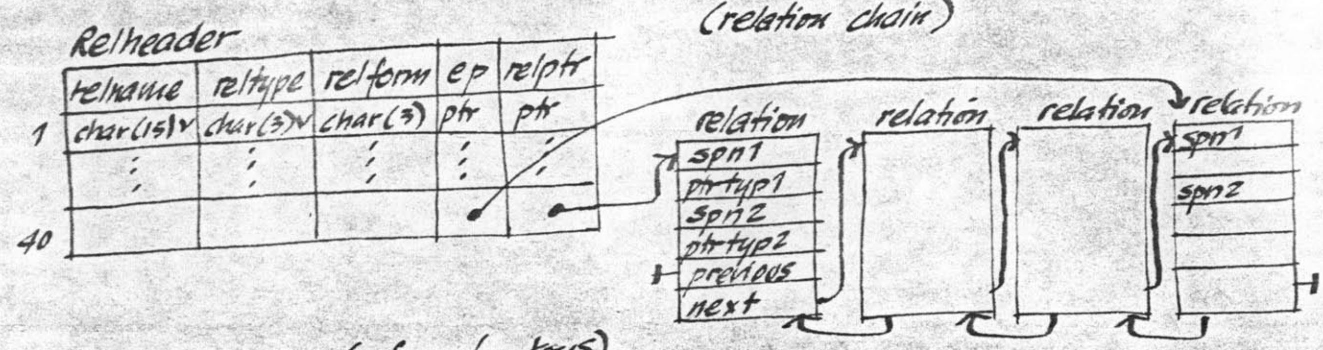
In our case the RDB was implemented with the following data structure: (FIGURE 4.5)

where we keep the entities and their relations as two external lists which contain respectively 20 entries for entity lists, and 40 entries for relaticn lists

In such entr ies we keep basic data about the entity chains or the relation chains, such as name (i.e. site, elements) (i.e. position rules, adjacencies), a

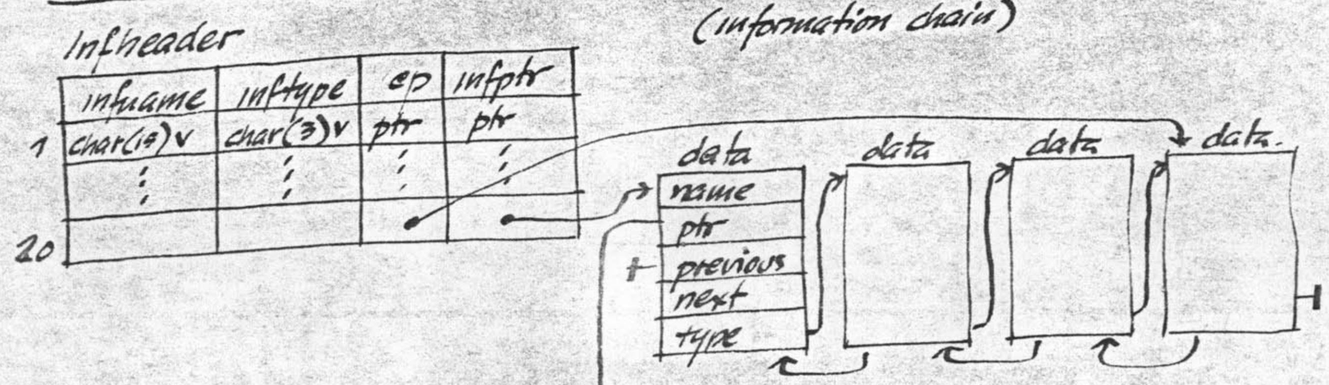
GENERAL RDB, Relational Data Base. and additional Information.

1.- RELATIONS: (relation pairs)



	relname	reltype	relform	ep	relptr
1	char(15)v	char(3)v	char(3)	ptr	ptr
	:	:	:	:	:
40					

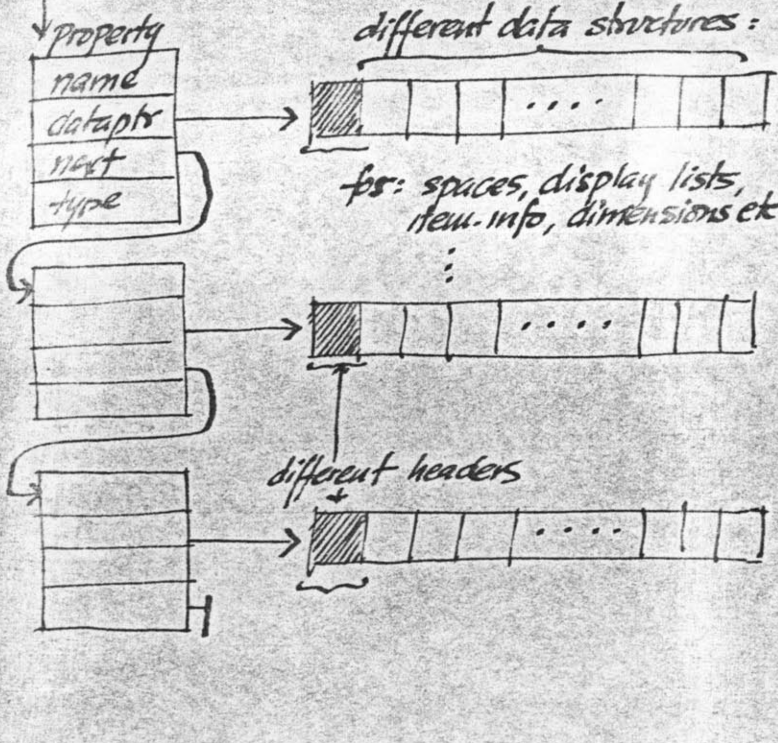
2.- INFORMATION (information keys)



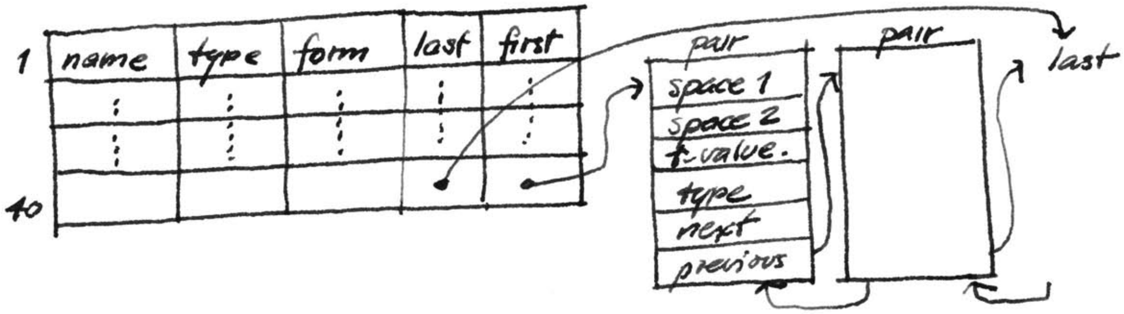
	infname	inftype	ep	infptr
1	char(15)v	char(3)v	ptr	ptr
	:	:	:	:
20				

3.- PROPERTY LIST (additional information).

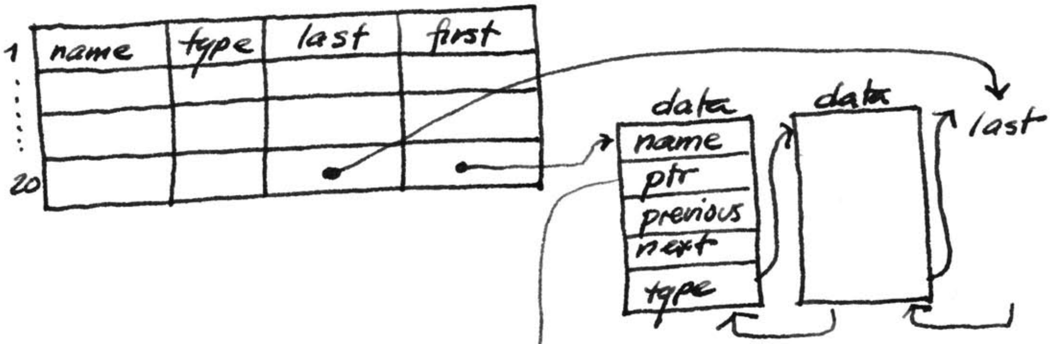
- a.-item-information
- b.-dimensions (teague)
- c.-restrictions
- d-display list
- e.-additional info.
- :
- n.- whatever comes late on.


Property
name
dataptr
next
type
different data structures:
....
for: spaces, display lists, item-info, dimensions etc.
:
....
different headers
....

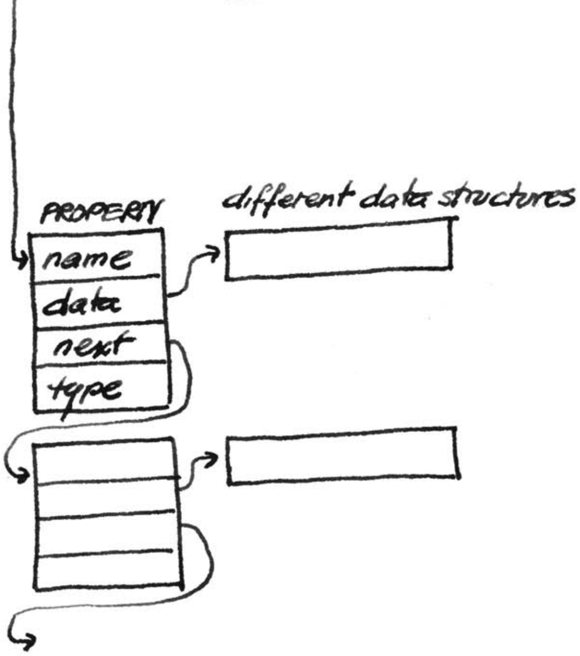
RELATIONS:



INFORMATION:



PROPERTY LIST:



RELATIONAL DATA BASE

pointer to the beginning of each list and a pointer to ·he end of each list, plus additional slots that were thought necessary but were not used at all, such as type and format of entities and relations.

For each infcrmation list, we have an entry in the chain for each entl y that we want to store. Each entry has a name, a type, a pointer to a property list ,explained further down, and two pointers that link it to the previous or to the next entry in the same list.

In a property list, additio al infcrmation is kept for each element besides its name and type. The idea of this list has been to be as flexible as possible in terms of the elements we use for our reoresentat ions.

From the descrlotion in 3.1 we can see that our information can be divided in the following wa y: and we can see that the relational part is taken care of by the

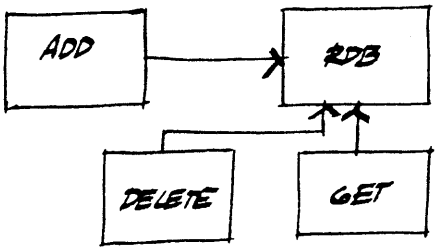
	SPATIAL	NONSPATIAL
RELATIONAL		
NONRELATIONAL		

entity and the relation lists, while the particular information row, is incluced in this property list. In here we can link several 'atributes' that an entry might have, spatial or nonspatial. For each atribute we have a 'property' entry which keeps track of the name of the information, for example: dimensions, restrictions, graphics, etc.; the actual data (in different data structures), and the needed pointers to further elements in the property list.

By subdividing information in this way we can store different kinds of elements in the entity lists, and keep their different data in different entries of property lists.

In the other part of our RDB, we have for each relation list, an entry representing a pair of elements being related by it. In this entry we donot need to store the elements again, but we store insteao an 'id' for such elements, a reference that can help us get to them in tre information lists where they are. By doing so we avoid redundancy of informatior. In our case this 'id' is tre address location of tre data entry in core, and we have therefore a pointer for each element location in the pair. The two other items are poirters that link our pair to the previous and next pairs in the chair.

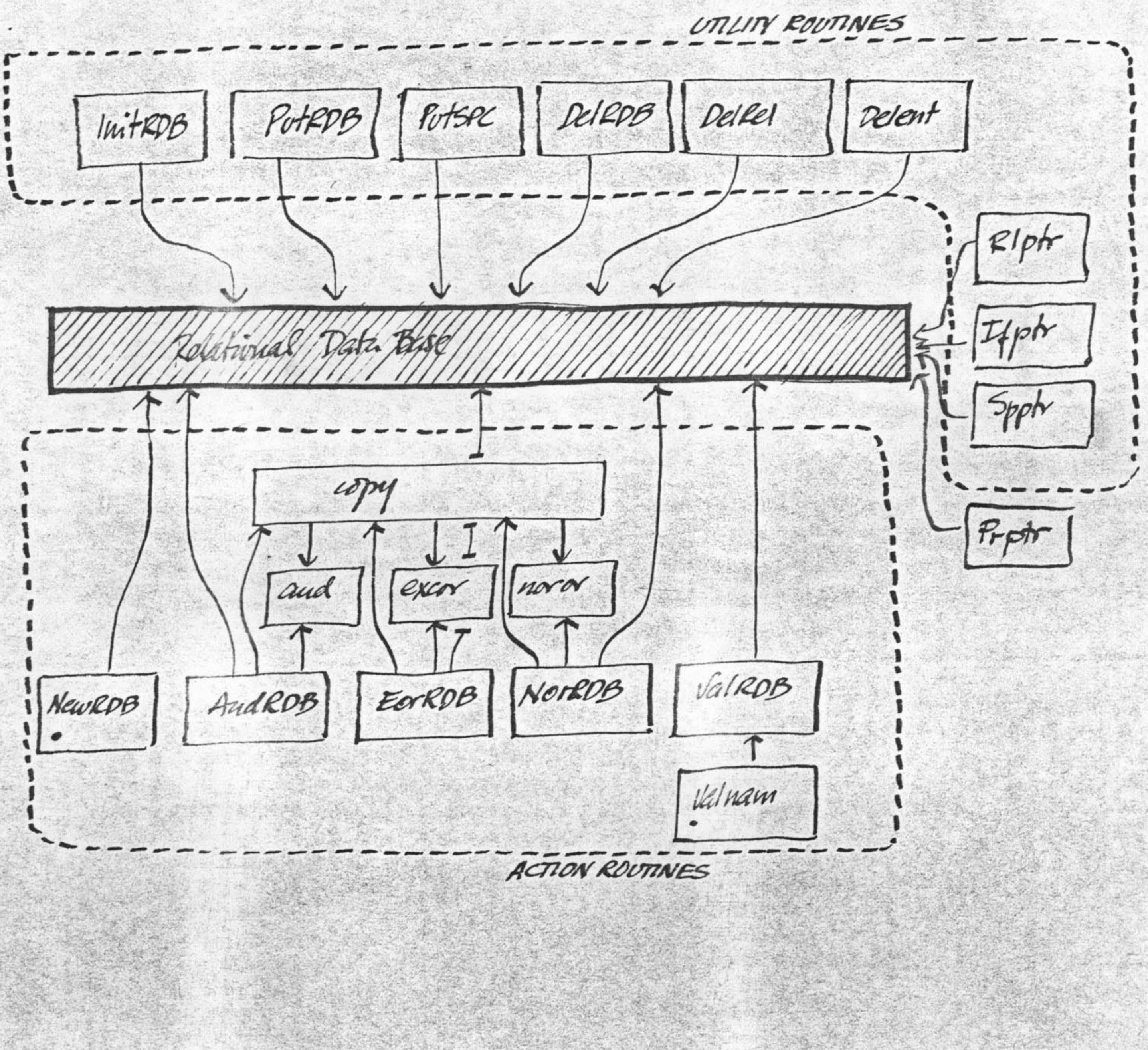
Several routines were written to insert, retrieve, delete or query elements and relations, as shown

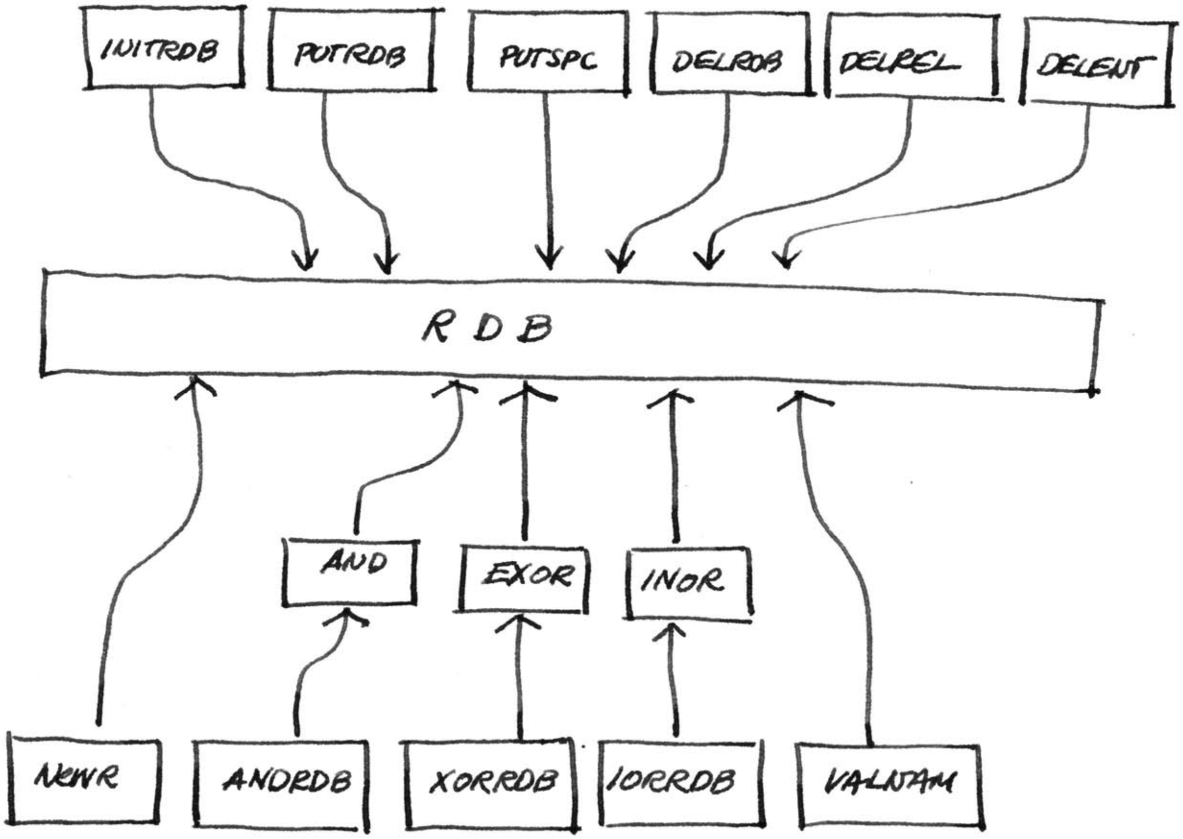


in figure 4.6

and in more detail, they are;

With them we can insert elements (PUTSPC), or relations(PUTRDB); delete entries in information lists (DELENT) or relation pairs in relation lists (DELREL) or relation pairs in all the relation lists (DELRDB); retrieve the values of some relations (VALNAM); or make combined queries as will be hown in QUERY LANGUAGE.


UTILITY ROUTINES
InitRDB
PutRDB
PutSPC
DelRDB
DelRel
Delent
Rlptr
Ifptr
Spptr
Relational Data Base
Prptr
copy
aud
excor
noror
NewRDB
AudRDB
EorRDB
NorRDB
ValRDB
Valnam
ACTION ROUTINES


INITRDB
PUTRDB
PUTSPC
DELRDB
DELREL
DELENT
RDB
AND
EXOR
INOR
NEWR
ANDRDB
XORRDB
IORRDB
VALNAM

The details of these routines are in the programmers manual not included in these Thesis.

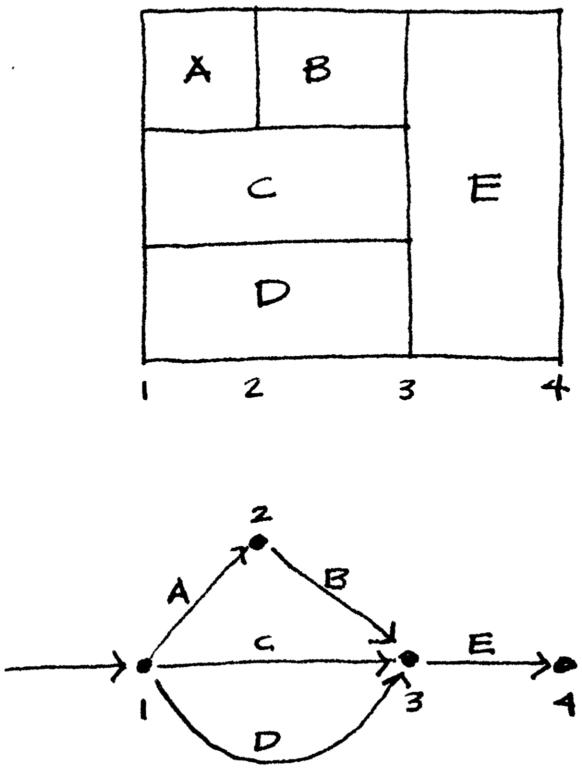
The advantages of an RDB(1) are then: ease of use, as simple tables like the ones in figure 4.2 are easier to understand, flexibility, precision, ease of implementation, data irdependence, clarity and the data manipulation languages (mimically present in 4.3).

4.2. Spatial Representation:

The spatial representation of furniture layouts is organized aroung L.Teague's Ph.D. (2) Thesis on "The representation of Spatial Relationships in a Computer System for building design".

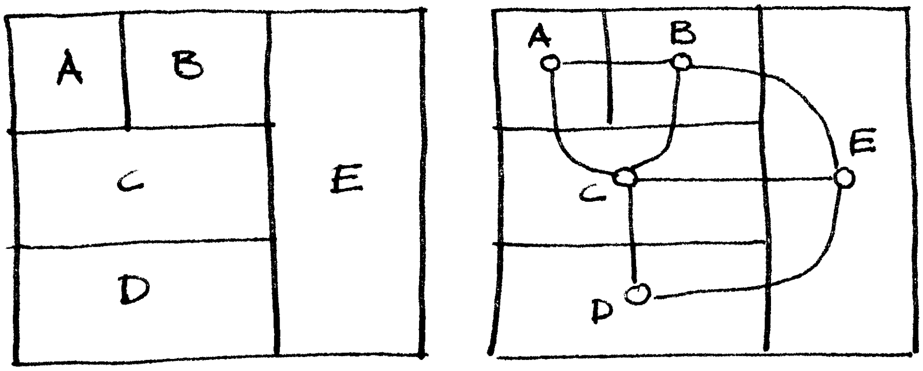
Teague describe spaces in a building as a network of rectangles within a larger rectangle. Based in Tutte's network representation of squared rectangles(3), it extends this description to three dimensions. By using the following representatior.

It express in network terms the spatial organization and makes therefore available the results of network theory for the analysis and synthesis of spatial



relationships.

In this network, spaces are described by 'arcs' or 'directed links' which 'flow' correspond to the vertical (z) or the norizontal(x,y) dimsnsions of tre space. The adjacencies between sides of two soaces are describeo as 'nodes' which receive on one side the 'arc' 'flow' of the left space, lets say, ano which are tre origin for tre 'arc flow' for the spaces in the right



side, like

This mode of representation as opposed for instance, to Eastman's (4) or Yessio's (5), was selected for two reasons:

1.- Its limitatiors to rectangle shapes do not interfere with the princicles in tre metrodology. In fact, even though we have complex shapes, they are always composed of rectangles because in the end, ZONES and SECTORS restrict the analysis to ortogonal soaces, and two directions.

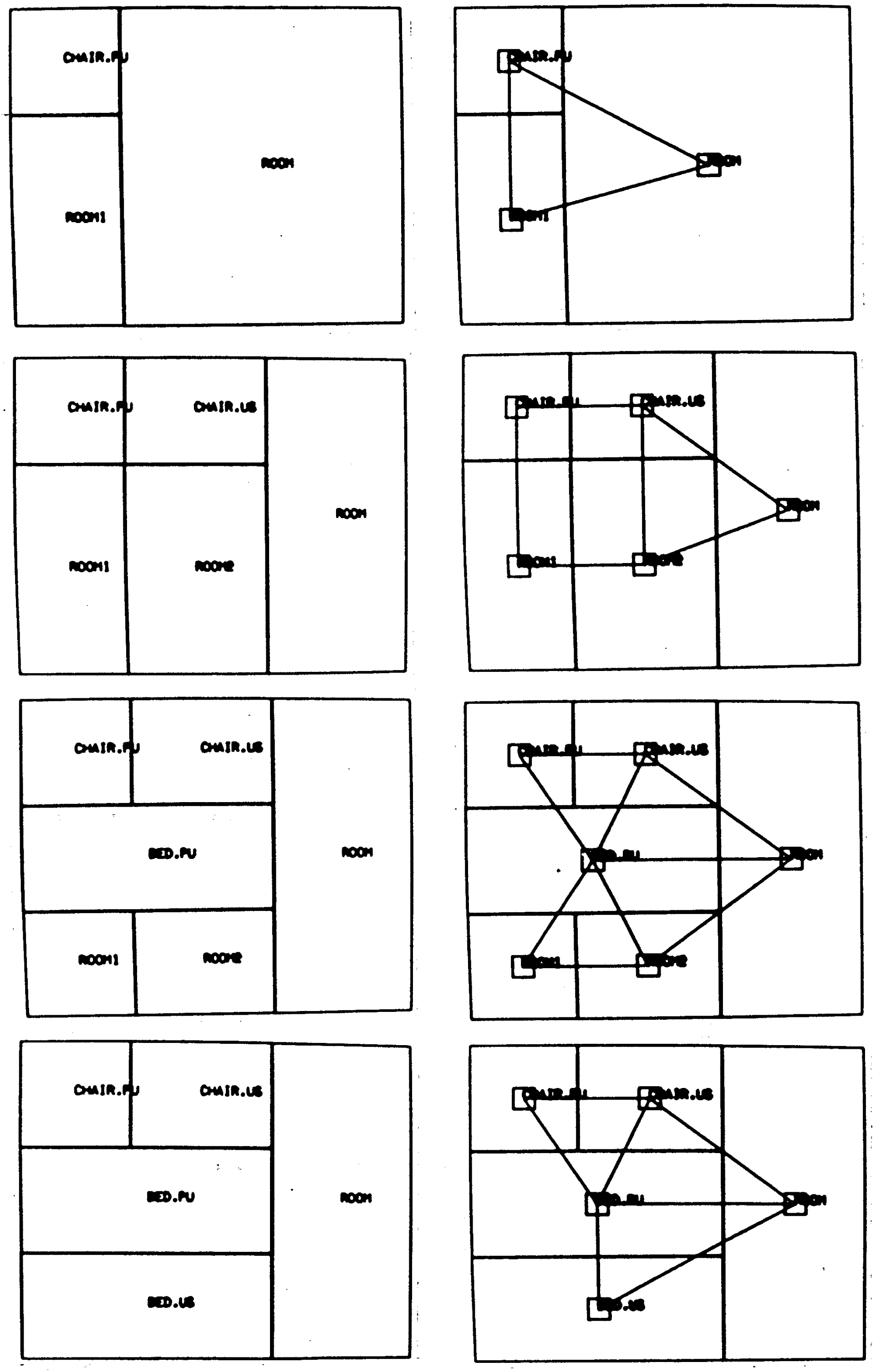
2.- Its network description blends itself quite well with our RDB and the network becomes one more relation in our data for configurations.

One change was made, however, instead of representing spaces as arcs and sides as nodes, we switched spaces as nodes and sides as links. Having then a graph with geometric characteristics such as shape and dimensions for each node, and having at each link the amount of adjacency between two spaces.

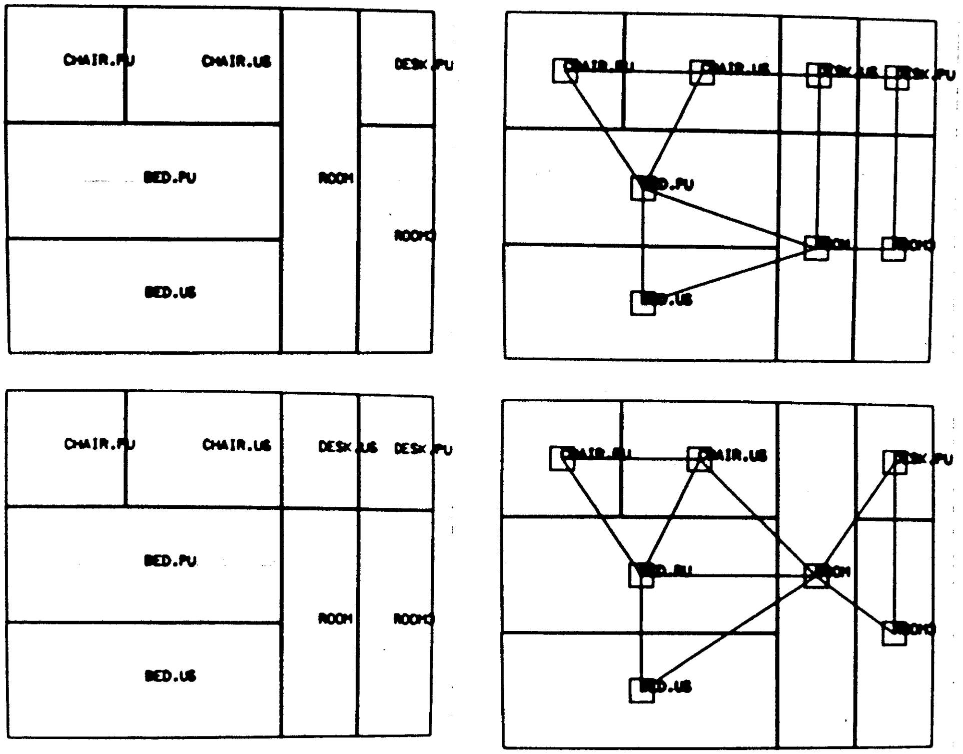
There is some limitations to Teague's elegant representation, when we construct this graph sequentially as it is done during generation of layouts (at Permutation of Elements), we have to keep track that all the resulting spaces are rectangles always. So, as he does, we establish a convention on how to obtain this 'squarec' representation. In our example: we can see how when we add our chair-physical-unit (this layout by the way corresponds to the generation of layout 1 in branch 28) we get an L shaped room. What we do then is extend the 'free' corner, i.e. SouthEast of chair, to the end of our space, and subdivide the room into 'room' and 'room1'.

The same happens with chair-use-space and we get 'room1' 'room2' and 'room'

When we put the bed-p.u. then we just reduce


CHAIR.FU
ROOM
ROOM1
CHAIR.FU
ROOM
ROOM1
CHAIR.FU
CHAIR.US
ROOM
ROOM1
ROOM2
CHAIR.FU
CHAIR.US
ROOM
ROOM1
ROOM2
CHAIR.FU
CHAIR.US
BED.FU
ROOM
ROOM1
ROOM2
CHAIR.FU
CHAIR.US
BED.FU
ROOM
ROOM1
ROOM2
CHAIR.FU
CHAIR.US
BED.FU
ROOM
BED.US
CHAIR.FU
CHAIR.US
BED.FU
ROOM
BED.US

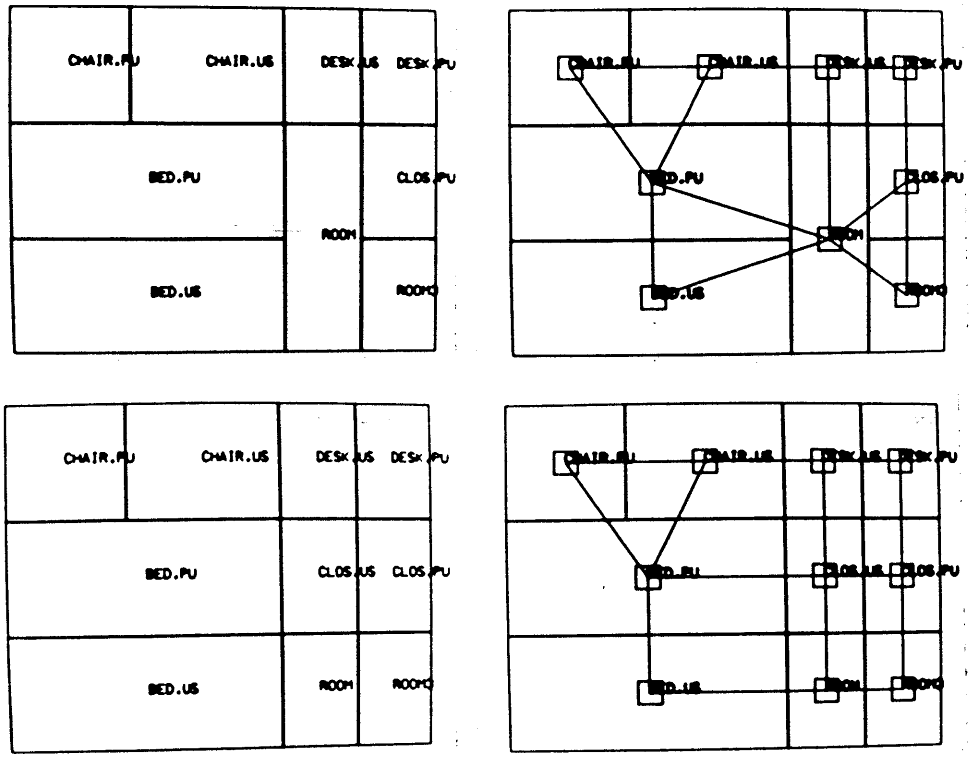
'room1' and 'room2', which d sappear with the position of



the bed-use-space.

The network in the right side of our figures should serve to illustrate that there is always a spanning tree at each level which allow us to move from one element to another.

until we get the final layout which is a basic furniture



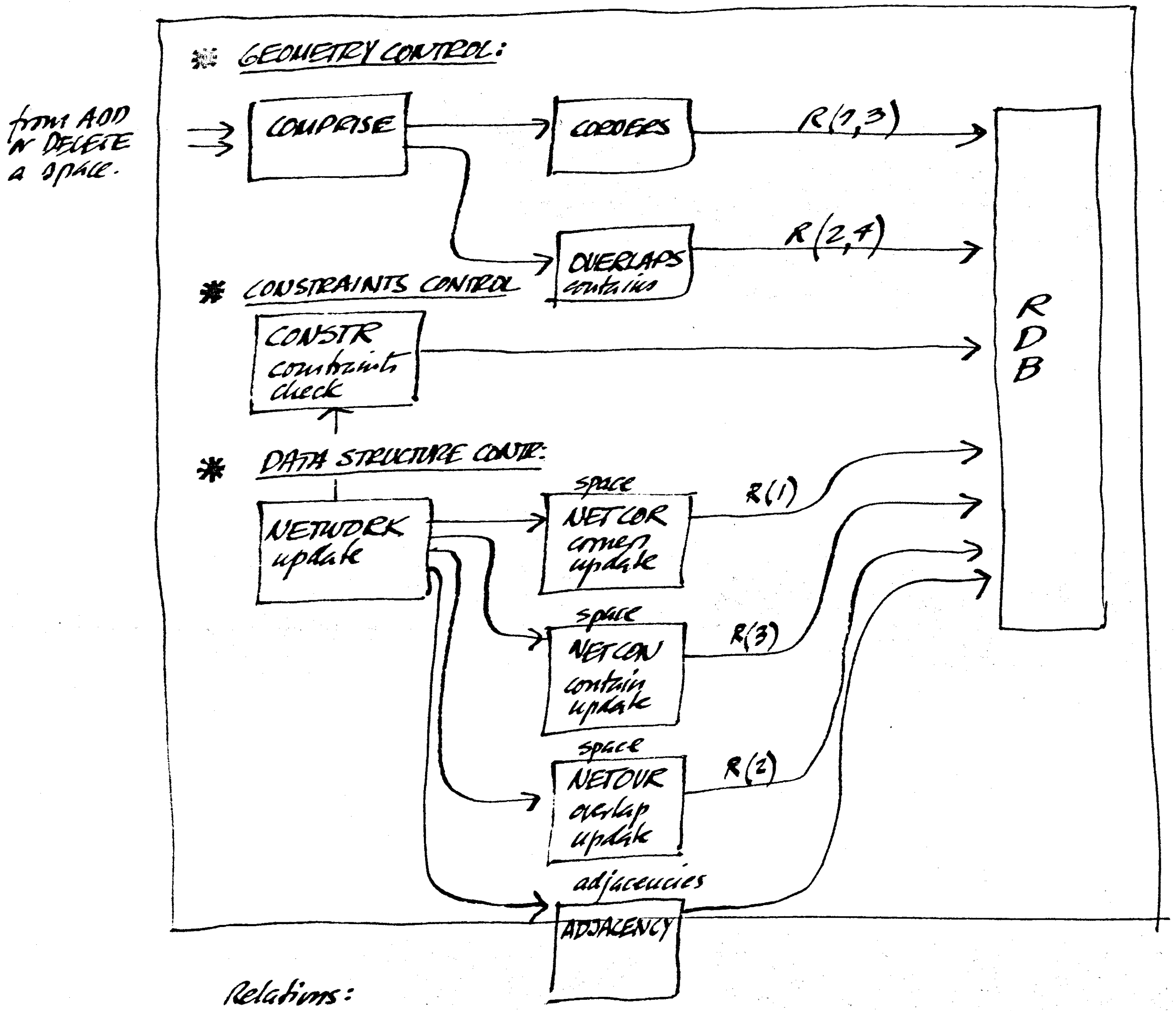
varlant.

As a layer In our RDB there is a set or outines that keep track of this NETWORK control. They chec where corners of spaces fall, what Is the contalnment or overlapping of other spaces, and make the necessary adjustments In our representation as shown before

4.3. Query Language:

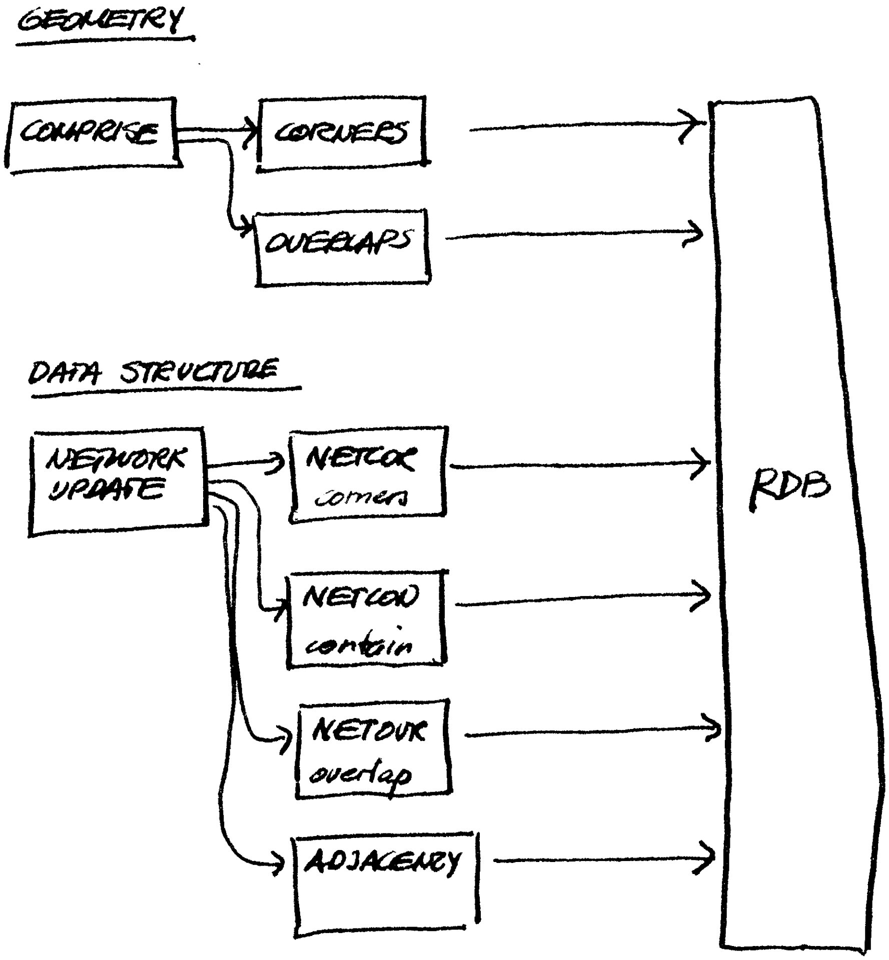
Through this module, I should say, pretentiously called, QUERY LANGUAGE, simple queries can be constructed

NETWORK CONTROL



Relations:

1.- CORNER-OVERLAPS
2.- SIDE_OVERLAPS
3.- IS_CONTAINED_BY
4.- CONTAINS



out of combinations of basic set operations such as: MEMBERSHIP, INTERSECTION and UNION, to retrieve or form new information in the RDB. Together with VALNAM in our previous routines, they constitute a reduced version of a relational calculus, which we can use to express, similarly to our previous rules, the following queries in the SARCASM syntax:

```
(OR (IS ADJACENT-TO-LEFT BED DESK)
    (IS ADJACENT-TO-RIGHT CHAIR DESK))
```

where (is relation_name entity1 enti y2) look for members 1 and 2 of the relation pair under the relation "relation_name", and OR is the same logical connective that we have used before. This query would be answered TRUE after we positioned the fourth element in our layout 1 or branch 28.

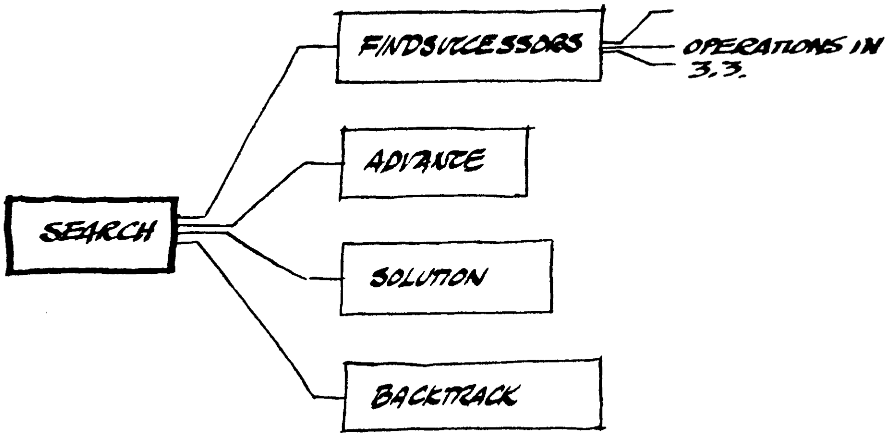
```
(AND.RDB (VALUE ADJACENT-TO-LEFT BED)
         (VALUE BY WALL2))
```

A different example, will get the value of all elements to the left of the bed, the value of all the elements by the wall2 and will find the set intersection of the two, to produce a list of elements each of them adjacent to the left of the bed and by the wall2.

Will return the elements that satisfy any of the

```
(OR.RDB (VALUE ADJACENT-TO-LEFT BED)
        (VALUE BY WALL2))
```

two relations, 'adjacency' or 'by', performing a set union.



The routines that do this work are:

4.4. Search:

Search is a recursive, backtracking proceduce which carries on the enumeration process defined before. Its parts correspond to each of the 9 operations we explained in there, coordinated by a general procedure.

This procedure explores the tree of possibilities in our solution space by applying the following principles:

- starting at the root, it looks first for one valid alternative among the successor nodes, whether in the binary-counter or the permutation of elements.

- if it finds one acceptable alternative, it advances then one level down in the tree, and applies the respective operator, an assignment of TRUE-FALSE values, or the positioning of a furniture piece.

- after advancing one level, it checks if we have a solution or not, if we do, it backtracks to the previous level and tries to find a next successor. If it doesn't find a solution it starts again, looking for the successors at the next level down.

-when there are no valid successors to extend a possible solution, it backtracks to a previous level and starts to look for other nodes in different branches.

- when we have a solution, and only one is demanded, it succeeds in its search and ends the process;

when all are asked it continues lookin for valid successors, advancing, backtracking and recording all tre other solutions until there are no more branches left to explore.

Its general parts are then:

and the operations of enumeration correspond

Chapter Four, Notes:

(1) Martin J.,

"Computer Data-Base Organization",

Prentice-Hall, Englewood-Cliffs N.J. 1975.

(2) Teague L.,

"The Representation of Spatial Relationships in a Computer System for Building Design".

Ph.D. Thesis, Civil Engineering, M.I.T. 1968.

(3) Tutte W.T.,

"Squared Rectangles",

Proceedings IBM Scientific Computing Symposium on Combinatorial Problems, White Plains, N.Y., IBM Data Processing Division, pp.3-9

(4) Eastman C.,

"Representations for Space Planning",

Communications of the A.C.M., Vol. 13, No. 4, April 1970.

(5) Yessios C.,

"Syntactic Structures and Procedures for Computable Site Planning",

Ph.D. Thesis, School of Urban and Public Affairs, Carnegie-Mellon University, Pittsburgh.

5.- CONCLUSIONS:

With this generative method we can find out if there is one possible configuration or furniture arrangement, or we can find out all the possible configurations for a given space.

We can show then what layout consequences are implicit in a functional standard. We can start playing then with the four parts of our definition and change parameters in the SITE, or the POSITION RULES, or the ELEMENTS or the CONSTRAINTS, and observe which new configurations appear or which configurations disappear.

If we are interes in the minimal dimensions that a space should have to contain a function, then we can construct with it the S.A.R. CHART OF CRITICAL LAYOUTS, where for a rectangular site we show in a matrix form, a set of rooms with a certain increment in the x or y dimensions, and we display in the first combination of dimensions that can hold our standard, the first or all (sequentially) layouts that are possible.

The CHART now is used to represent the norms that an architect has in mind when he assigns dimensions to spaces, and as such it is the first step in the design of supports. With the application of the S.A.R. principles to functional definitions, we can now be precise in our formulation of norms, and be able to produce not one, but all the furniture arrangements that exist in each room of the CHART.

This exhaustive capability is not 'proved' in the mathematical sense in this Thesis, but merely 'felt' that it might be true.

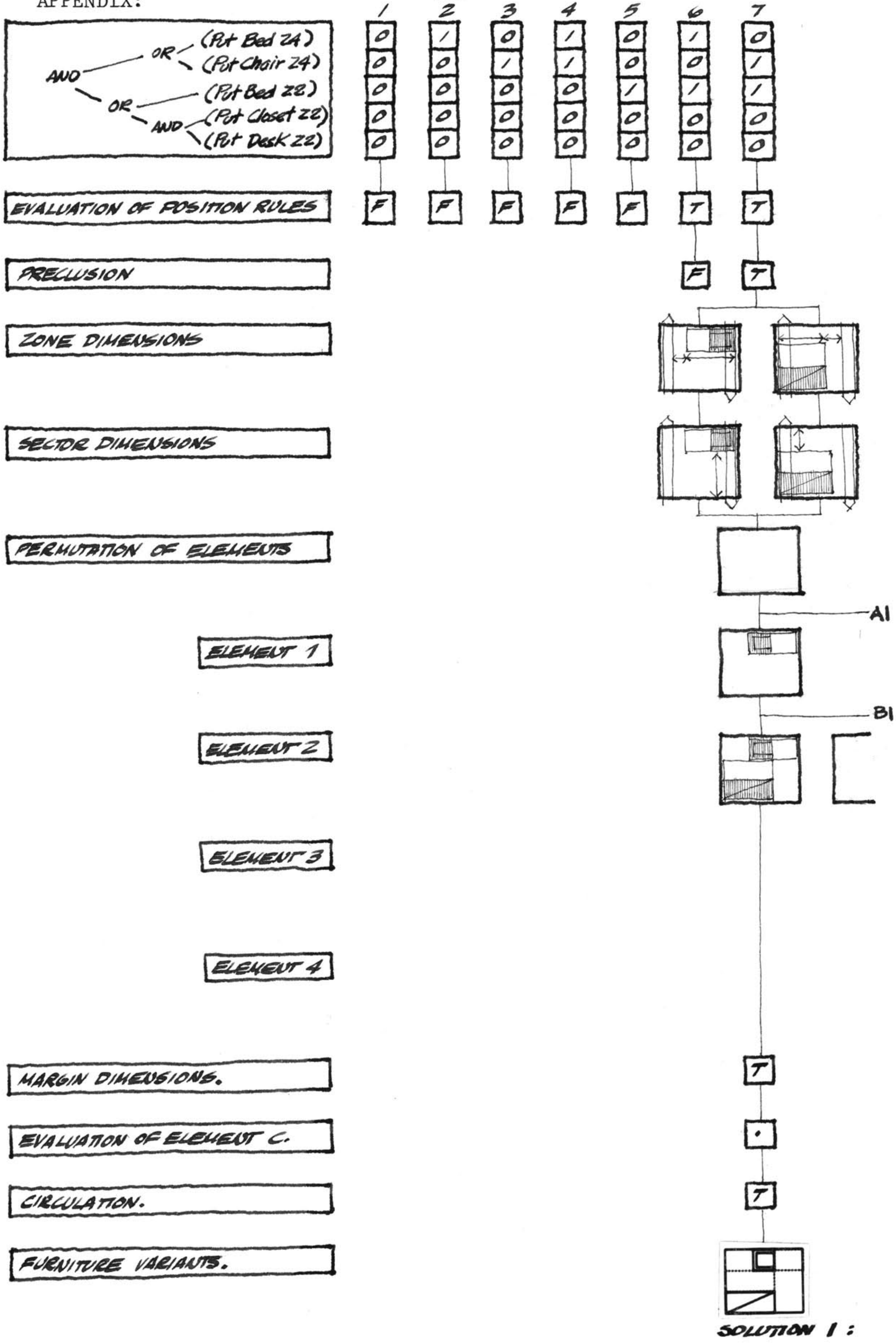
We have however a basis for inters explorations. If we assign preference values to position or relations, we can obtain layouts in terms of more 'desirable' configurations or less 'desirable' arrangements. If we assign cost factors to the different arrangements, we can talk about economic performance of a space. If we show the conseouences of our design standarcs we can have a meaningful dialogue for personal preferences and a tool for analyzing spatial norms, or formulating new spatial standards.

The generation has been made very much in an 'ad hoc' manner, grabing concepts from different places as they were needed, and mixing them perhaps in a very unelegant way, but for the time being it has been an exciting experience to be able to enumerate design alternatives.

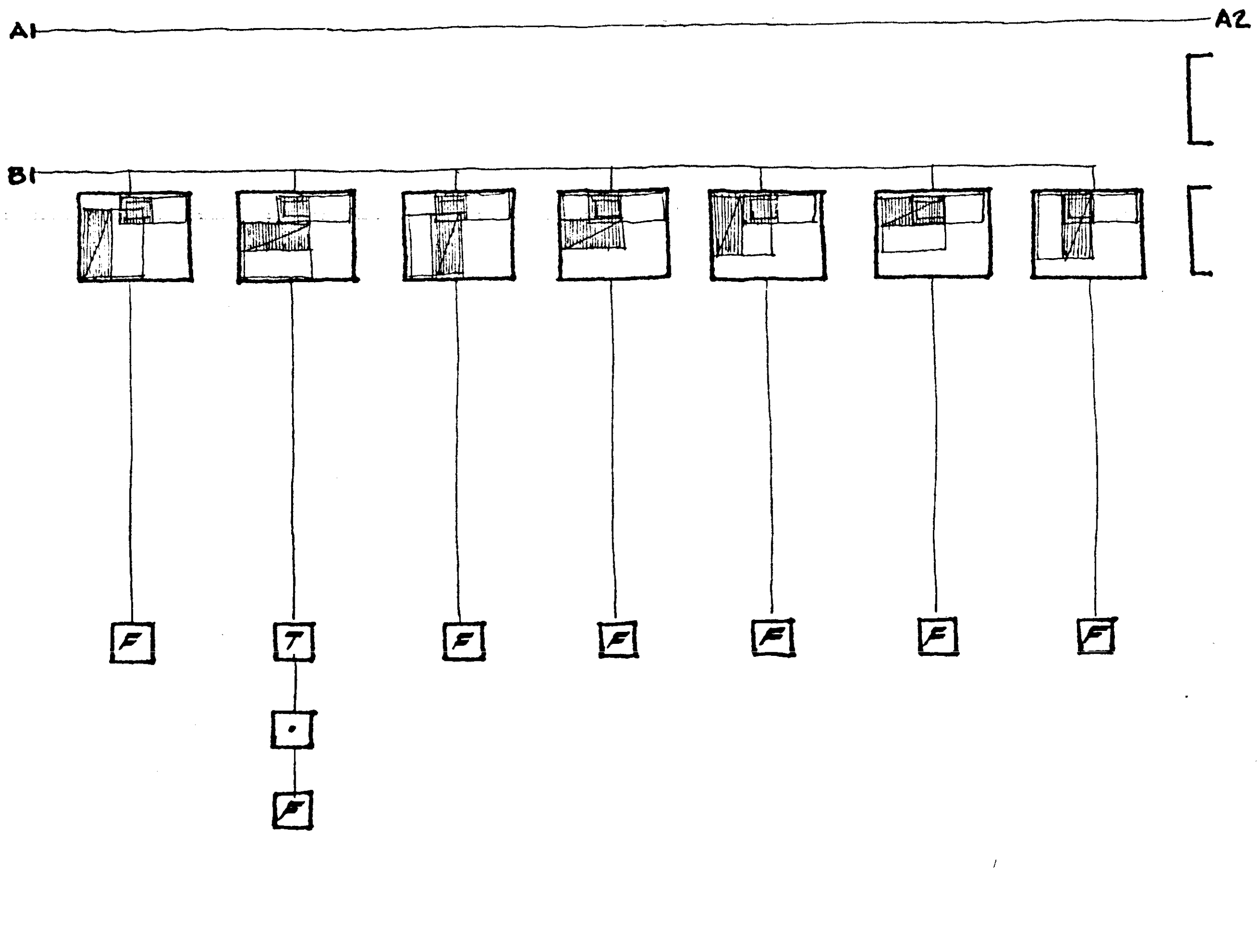
If it started from interests in different fields, it did not end with answers for each, but instead left many questions in all. How to define a design in terms that permit its exhaustive enumeration of possibilities is a problem not solved in this Thesis, but barely touched. How to advance in the direction of this

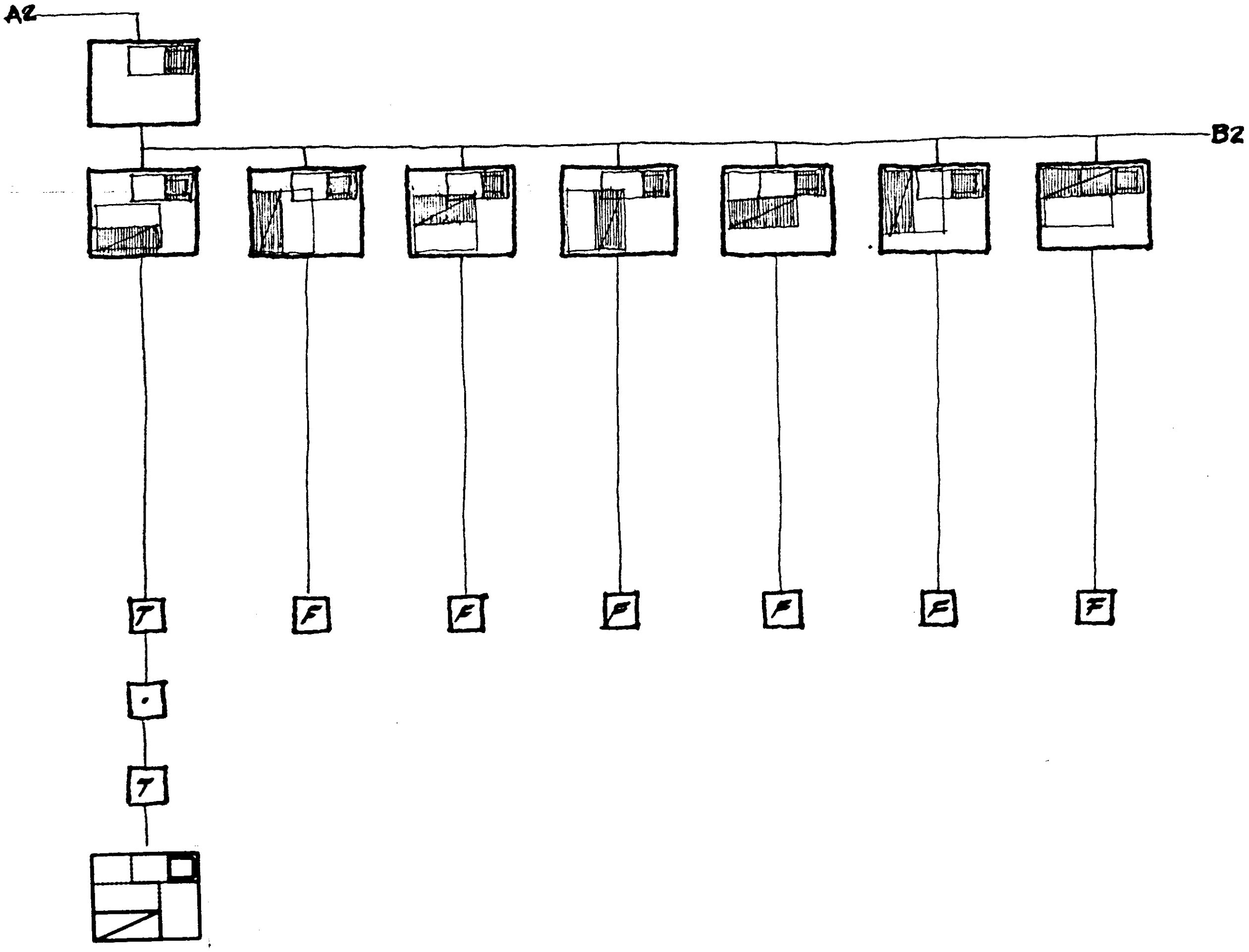
Theory of Spatial Configurations and our reasoning in manipulating them,might not be clear now but certainly worth to continue exploring.

APPENDIX:

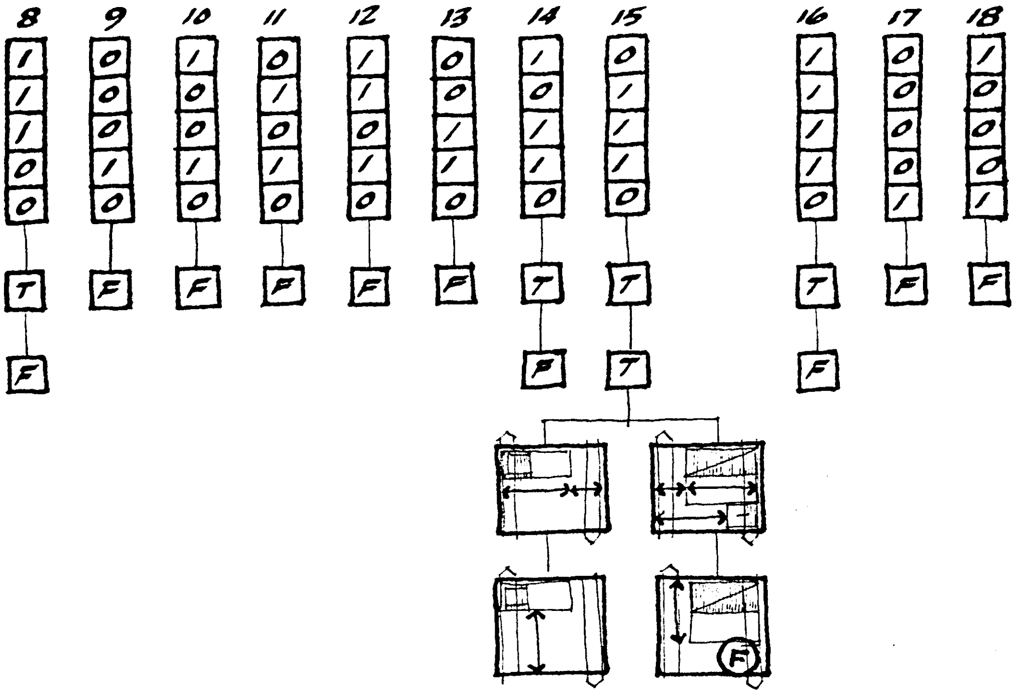


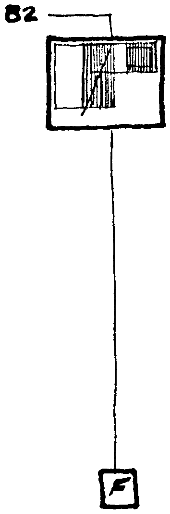
	1	2	3	4	5	6	7
(Put Bed Z4)	0	1	0	1	0	1	0
(Put Chair Z4)	0	0	1	1	0	0	1
(Put Bed Z2)	0	0	0	0	1	1	1
(Put Closet Z2)	0	0	0	0	0	0	0
(Put Desk Z2)	0	0	0	0	0	0	0

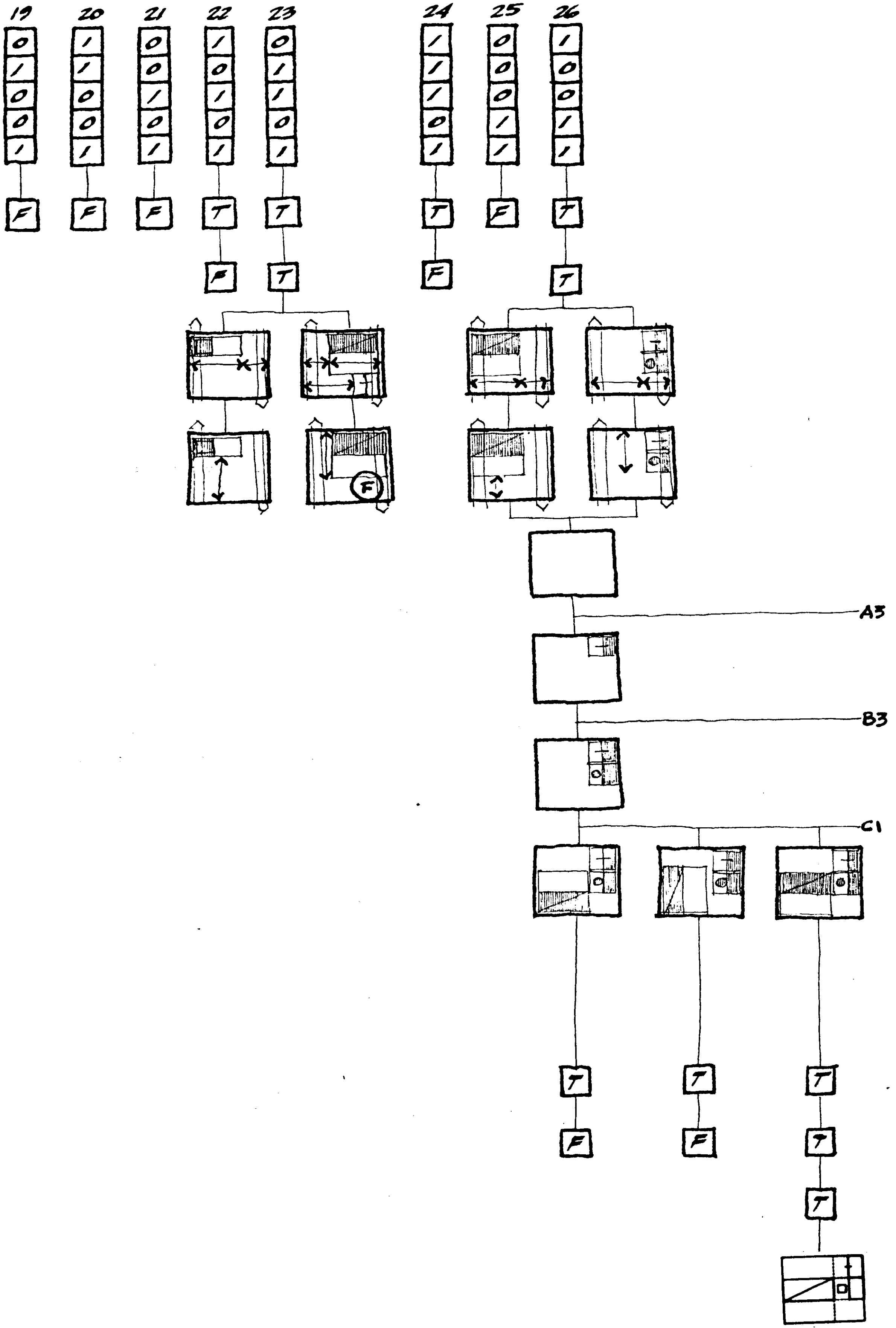

A1
A2
B1
F
T
F
F
F
F
F
F



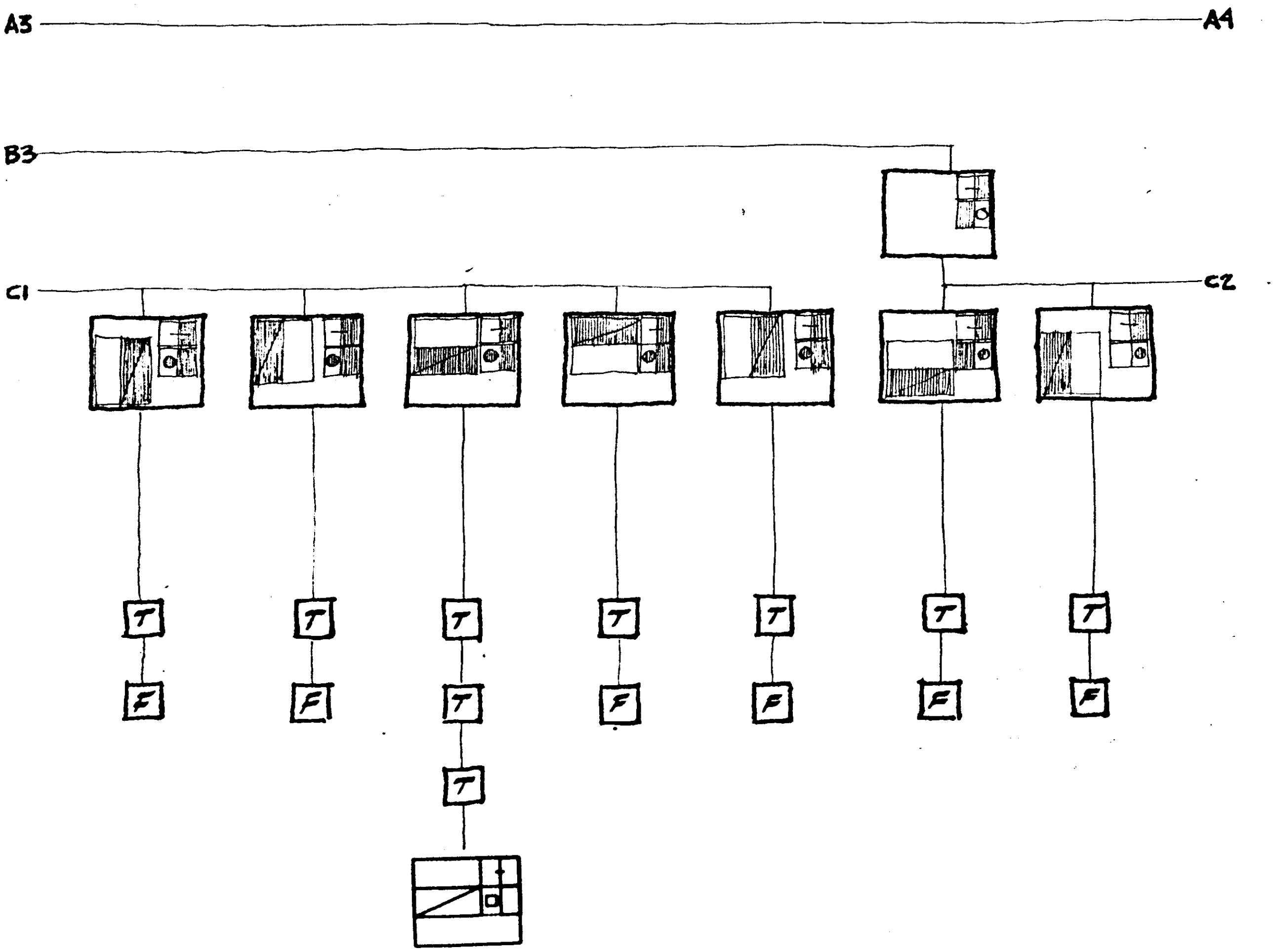
SOLUTION 2:


8
1 1 1 0 0
T
F
9
0 0 0 1 0
F
10
1 0 0 1 0
F
11
0 1 0 1 0
F
12
1 1 0 1 0
F
13
0 0 1 1 0
F
14
1 0 1 1 0
T
F
15
0 1 1 1 0
T
T
16
1 1 1 1 0
T
F
17
0 0 0 0 1
F
18
1 0 0 0 1
F
F

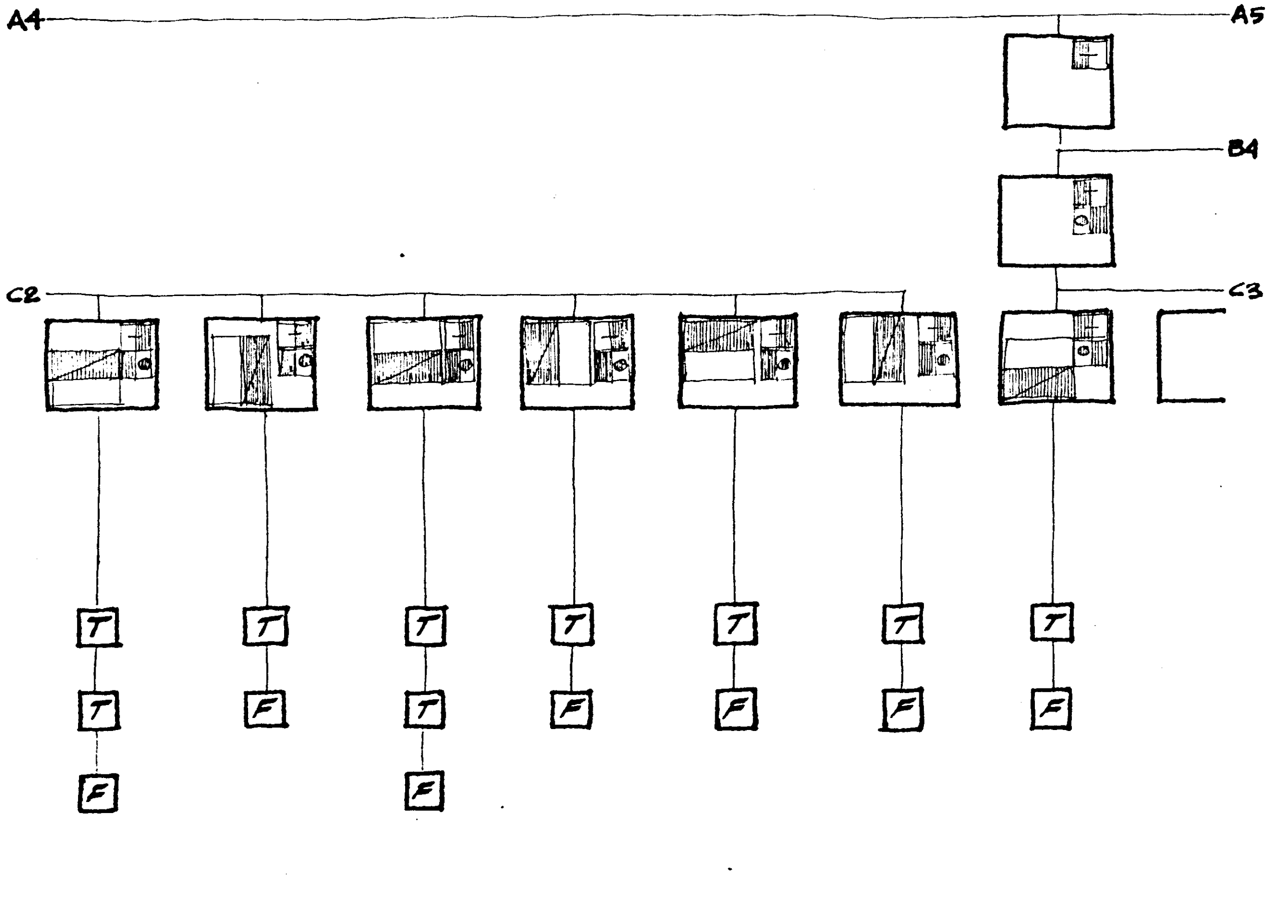

82
F

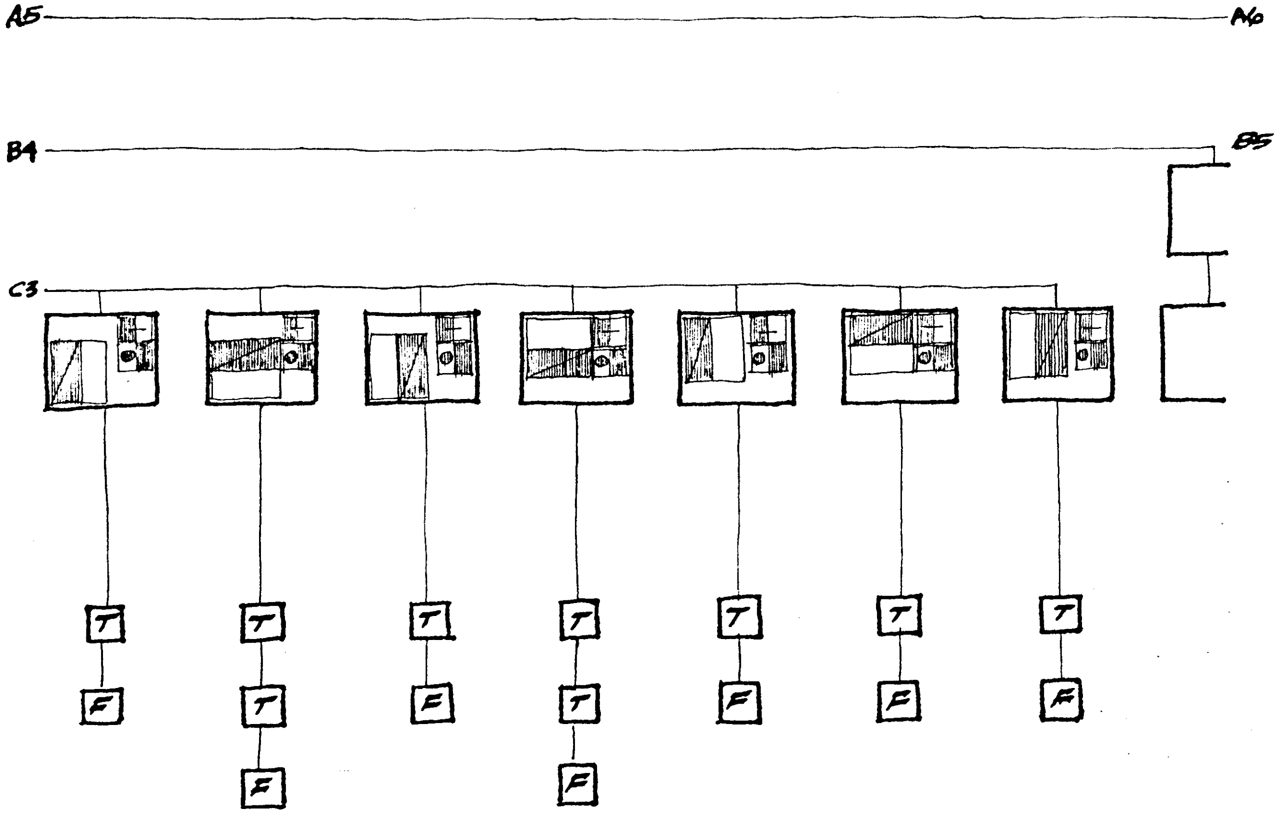

19
0 1 0 0 1
F
20
1 1 0 0 1
F
21
0 0 1 0 1
F
22
1 0 1 0 1
T
F
23
0 1 1 0 1
T
T
24
1 1 1 0 1
T
F
25
0 0 0 1 1
F
26
1 0 0 1 1
T
T
F
A3
B3
C1
T
F
T
F
T
T
T

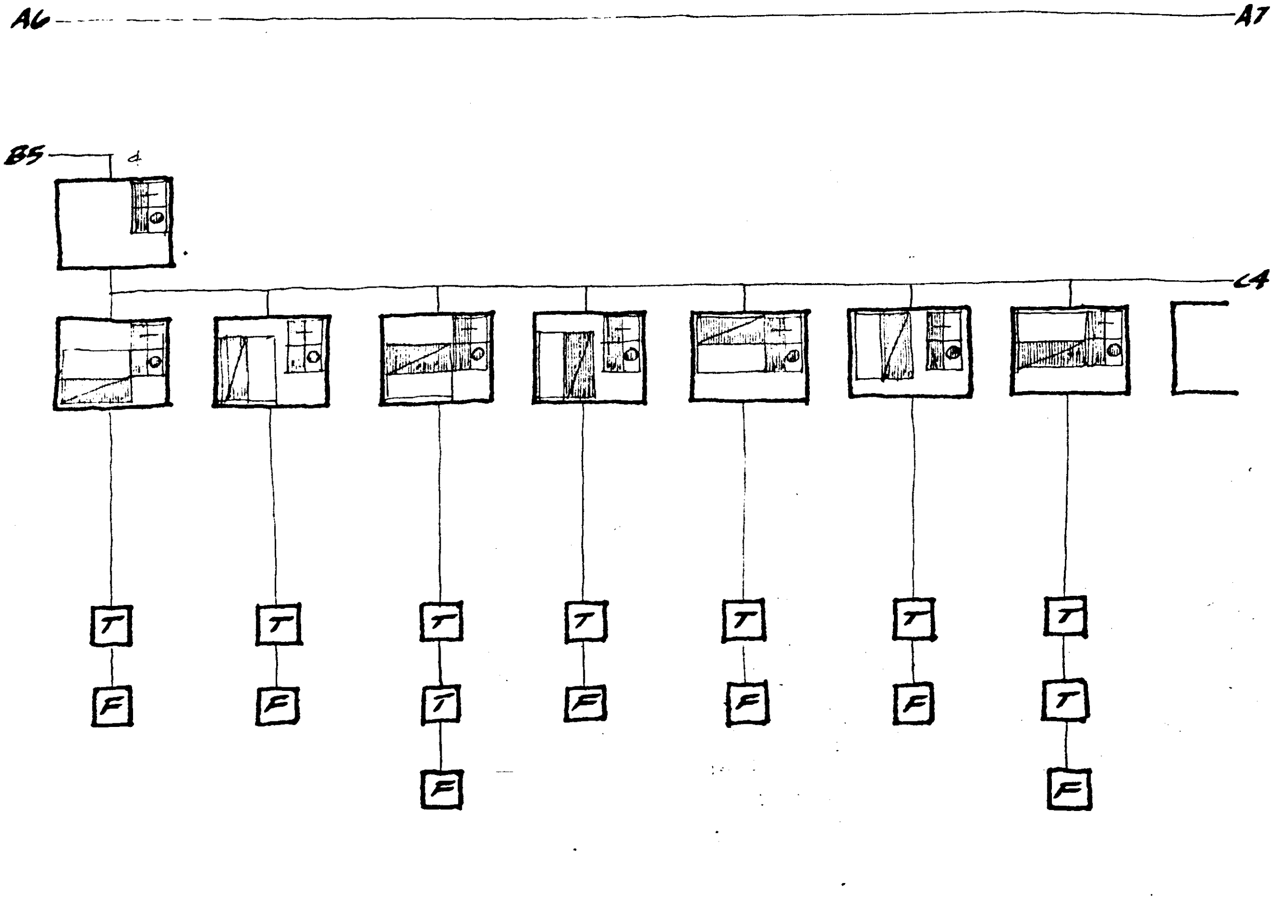
SOLUTION 3:

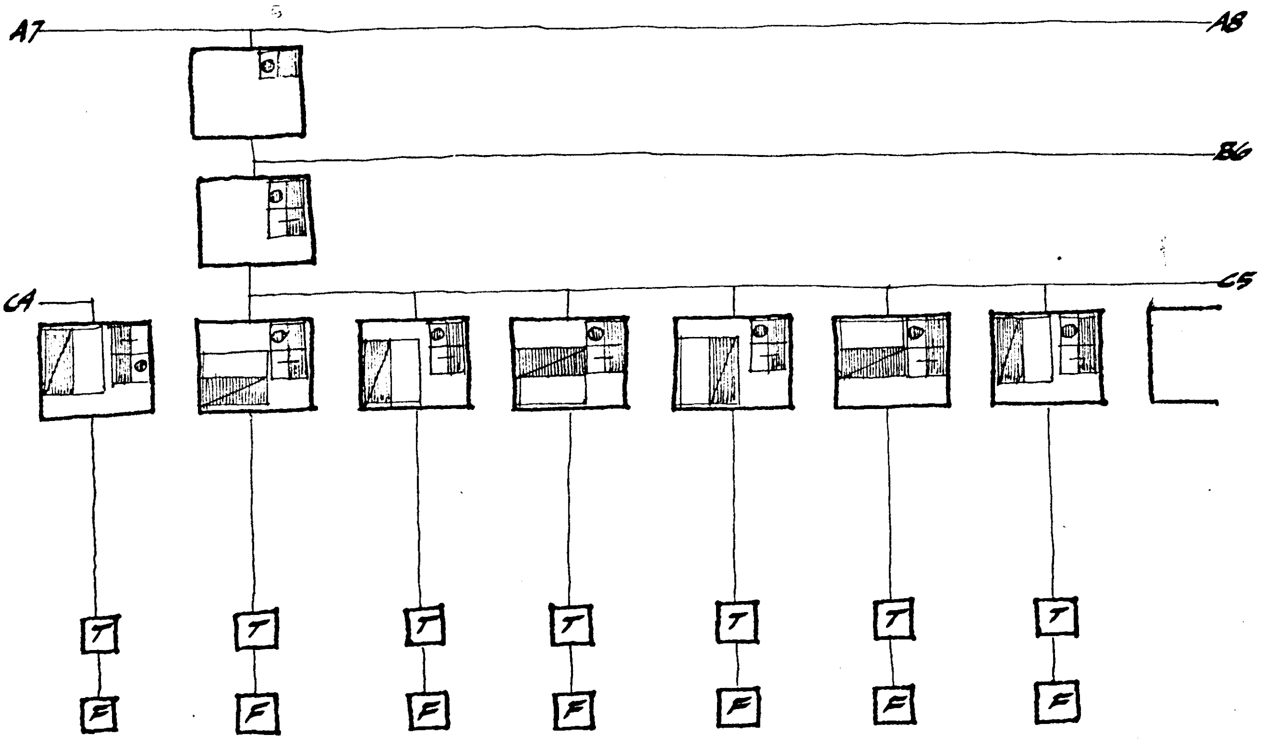


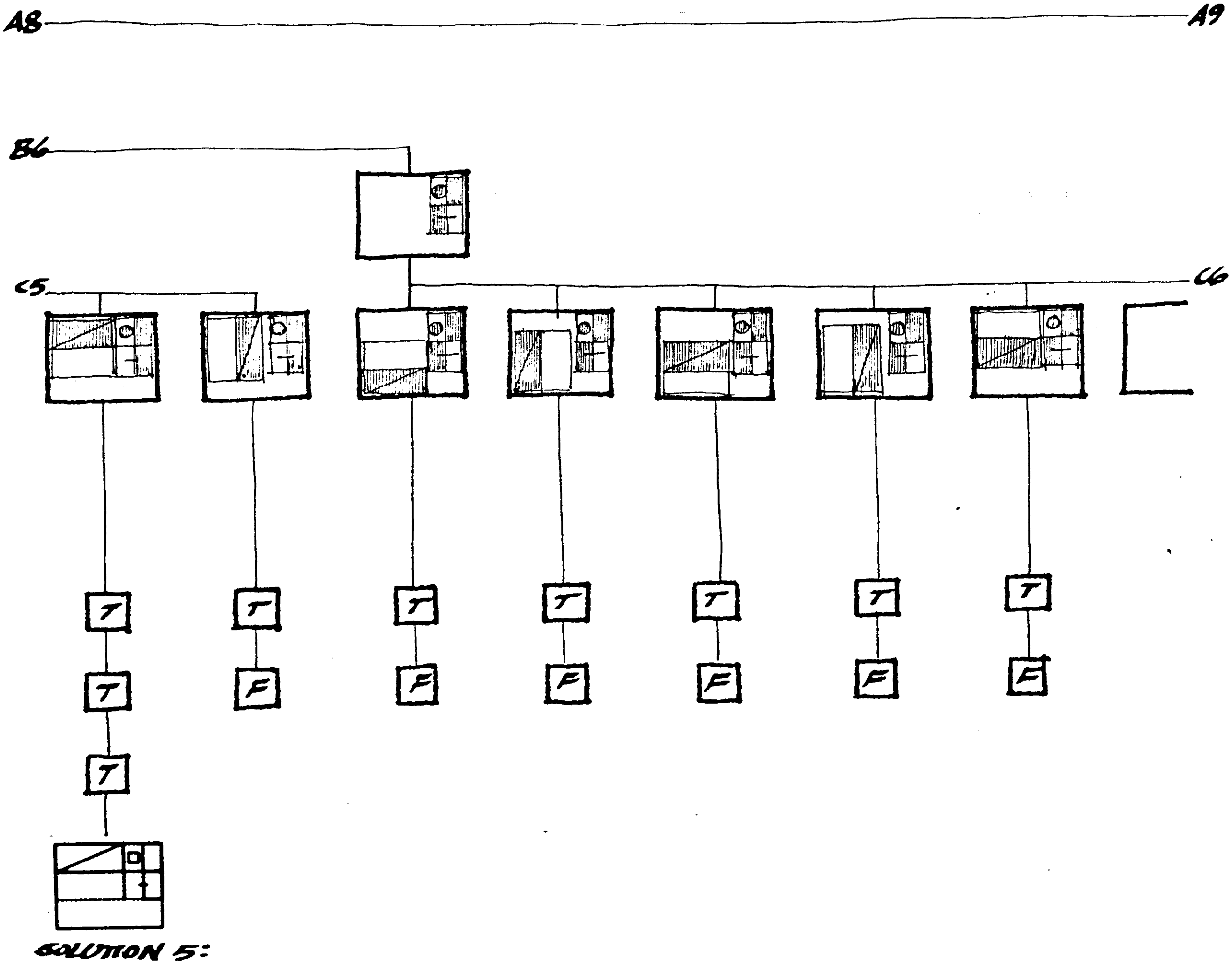
SOLUTION 4:


A4
A5
B4
C2
C3
T
T
F
T
F
T
T
F
T
F
T
F
T
F
T
F

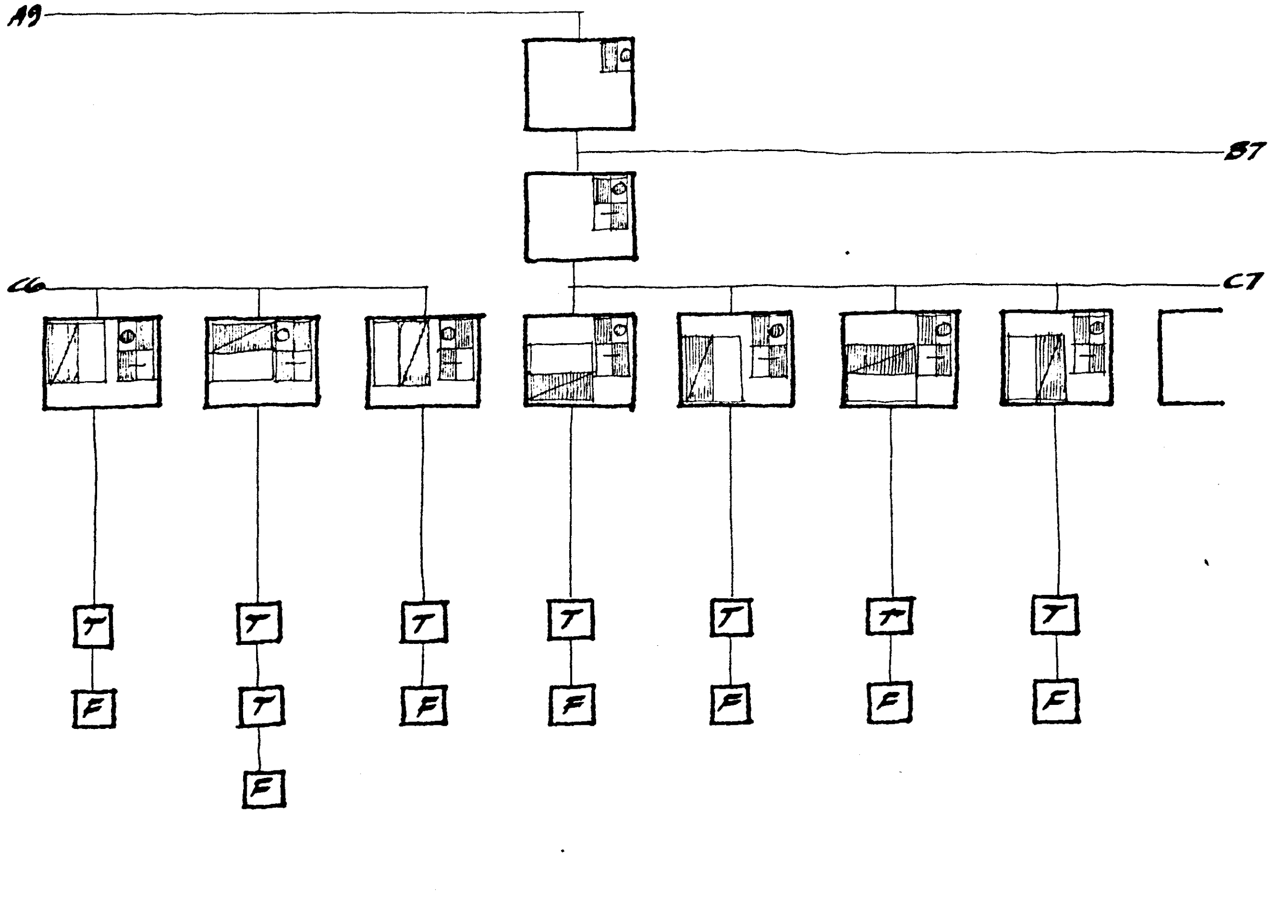

A5
A6
B4
B5
C3
T
F
T
T
F
T
F
T
T
F
T
F
T
F
T
F

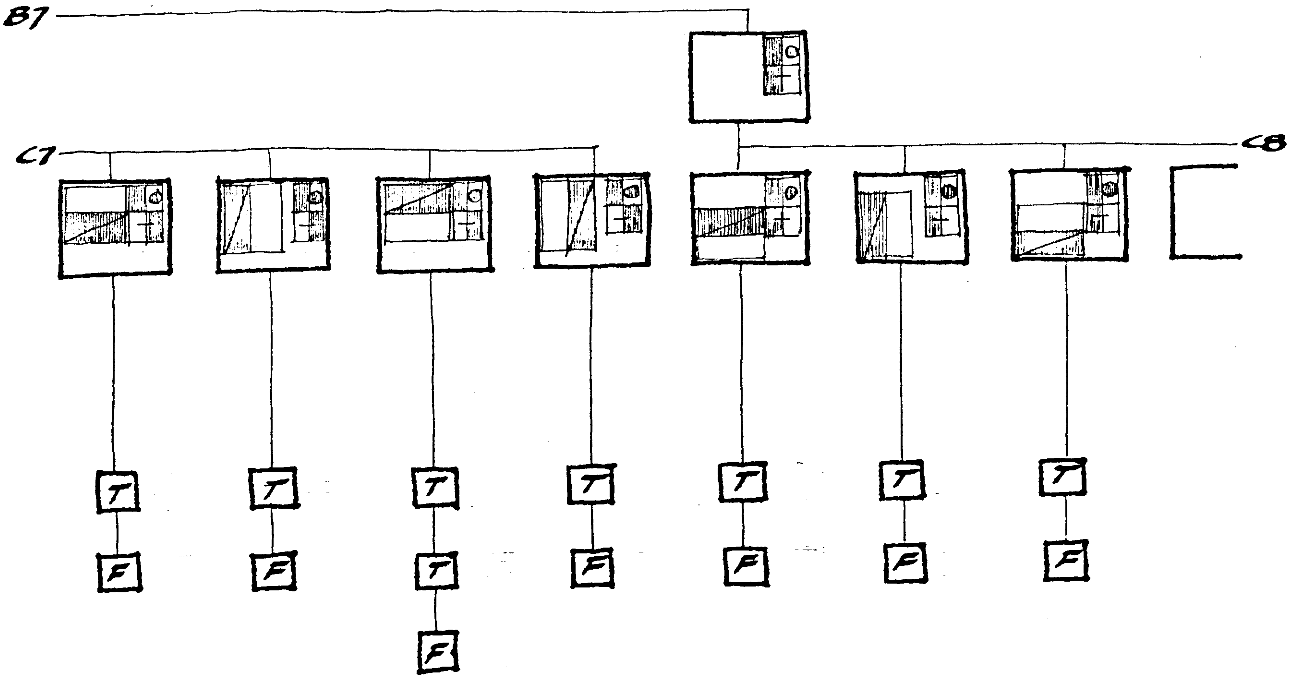

A6
A7
B5
d
C4
T
F
T
F
T
T
F
T
F
T
F
T
F
T
T
F

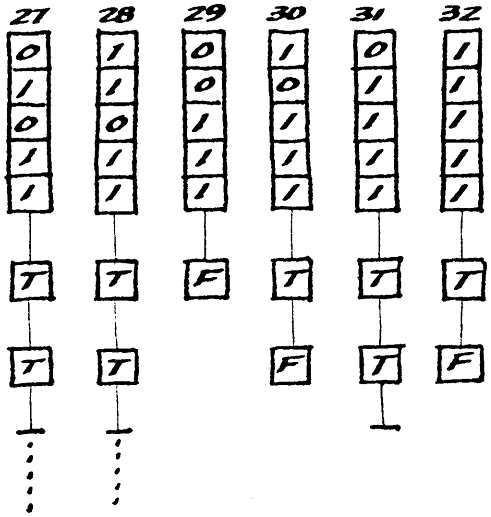

A7
A8
B6
C4
C5
T
T
T
T
T
T
T
F
F
F
F
F
F
F

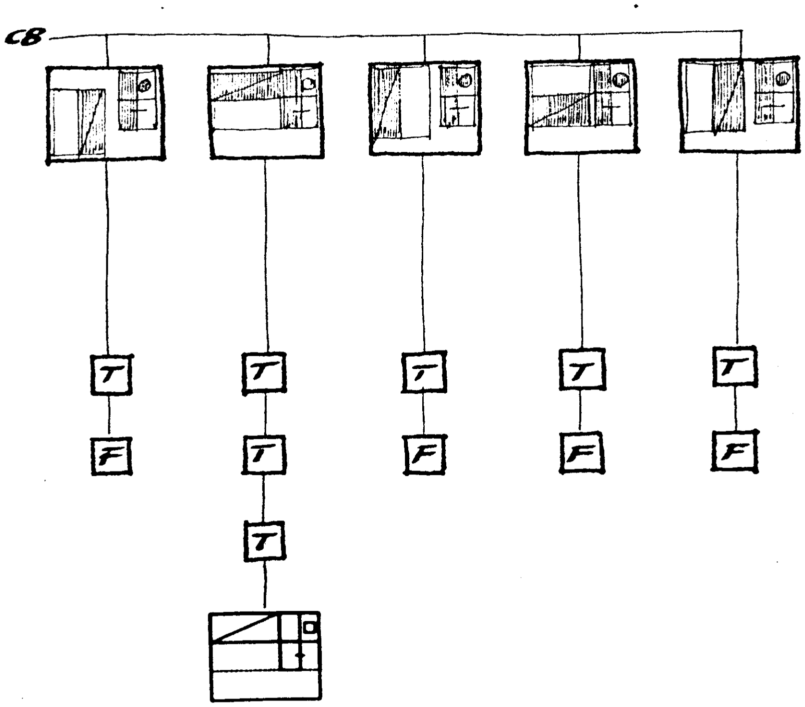


SOLUTION 5:


A9
B7
C6
C7
T
T
T
T
T
T
T
T
F
F
F
F
F
F
F
F


B7
C7
C8
T
F
T
F
T
F
T
T
F
T
F
T
F
T
F
T
F


27
0
1
0
1
1
T
T
28
1
1
0
1
1
T
T
29
0
0
1
1
1
F
30
1
0
1
1
1
T
F
31
0
1
1
1
1
T
T
32
1
1
1
1
1
T
F


CB
T
F
T
T
T
T
F
T
F
T
F

SOLUTION 6: