

LOGICAL GRAPHICS

**Logical Representation of Drawings to Effect
Graphical Transformation**

Peter J. Szalapaj

**Ph.D.
University of Edinburgh
1988**



**I hereby declare that I am the sole author
of this thesis.**

Peter J. Szalapaj

ABSTRACT

A general model for a CAD system is presented, having the following basic components:

- i) A generalised *representation environment* for describing things, offering a range of logical operators for instancing and manipulating descriptions within a hierarchical framework.
- ii) A *drawing machine* which interprets instances of drawings (depictions) into forms recognised by the representation environment, and depicts logical representations of drawings. The drawing machine possesses knowledge of drawing operations in general, including arbitrary modifications and attachments between separate pieces of drawing, and subsequent regeneration of drawing objects. The drawing machine allows the description of drawings in terms of their primitive elements, and the expression of relationships between drawing parts. The manipulation of descriptions of drawings by means of transformational rules preserve the topologies of drawings and allow user-controlled distortions of shapes.

The original work and contributions of the author lie in the partial description and implementation of these basic components, with particular emphasis on the definition of the drawing machine.

Preface

This thesis is concerned with the description and manipulation of drawings using computers. Its central aim is to assist description in accordance with Pye's [Pye, 1978] 2nd requirement for design, viz.

"The components of the device must be geometrically related - in extent and position - to each other and to the objects, in whatever particular ways suit these particular objects and this particular result."

A linguistic approach is investigated here for communicating descriptions of architectural drawings to a logical representation environment. Drawings are described using more formal methods for expressing and dealing with syntax and semantics than is commonly found in conventional CAD systems. After a discussion of the analogical vs symbolic nature of drawings in chapter 1, the assumption made is that drawings can be represented symbolically. Symbolic representations for drawings are those in which graphical primitives have arbitrary significance. This is to be contrasted with iconic representations, where shapes gain their meaning as a consequence of their depictive relation to visual objects.

This thesis has approached the design of a drawing system to be used by architects, in a radically different way from those adopted for conventional drawing systems. There were several objectives in the design of this system.

Firstly, it was to sit squarely within that area of CAD which attempts to build *integrated* design systems, and therefore, although the focus of the thesis was primarily on the description of a design by means of drawings, it was anticipated that this part should fit into a larger whole. The objective of integrated systems embraces such things as verbal descriptions, analysis by means of functions, multiple views, and many other features which in conventional systems manifest themselves as discrete packages, if at all. Such an anticipation was to be reflected to as large an extent as possible in the form of the knowledge representation used.

An essential part of this problem of knowledge representation concerning drawings is that in order for the representation of drawings to be potentially interconnected with symbolic representations of other (non-spatial) properties of depicted objects, they too have to be represented symbolically. A symbolic representation of drawings is one which is independent of the drawing space (co-ordinate values). This leads on to the important point that drawings represented in this way share the same logical structure as representations of

other, non-graphical, objects. This can be contrasted with conventional systems in which separate representation structures (for drawings and for non-graphical objects) have to be known and understood, in addition to thinking about a design object.

Secondly, conventional CAD systems tend to be too tightly bound to particular knowledge domains, such that system operations become very prescriptive in their use. The knowledge embodied within the system presented in this thesis was to be of a very general nature, thus allowing the description of, and operations upon, a wide range of objects.

The body of this thesis consists of a semi-formal description of a drawing machine. The drawing machine is described in terms of a transformational grammar with a context-free base component, and a context-sensitive transformational component. The advantage of such an approach is that the meanings of system operations become relatively unambiguous. This is in contrast to many systems in conventional practice which have been designed without rigorously defined semantics.

The thesis contains a modest amount of theoretical material, and the results are essentially pragmatic. The applicability of this approach to drawing manipulation problems in the context of architectural design is demonstrated.

Organisation

The layout of the thesis is in three parts. **Part One** outlines different aspects that impinged upon the design of the system, including thinking of drawing as language, looking at examples of drawings obtained from an architectural practice (the Scottish Special Housing Association), a study of the properties of conventional CAD systems, and a look at alternative approaches to the development of CAD systems.

Part Two contains the central contribution of this thesis, outlining the different aspects and properties of the system. These include the representation environment, the drawing generation component which communicates with the representation environment, a transformational component which supports transformations and user-controlled distortions of shapes, and a component which supports the attachment of graphical objects by means of logical merging, together with transformations.

Part Three concludes the thesis with a look at unresolved problems and observations arising from this thesis.

ACKNOWLEDGEMENTS

This thesis is the result of several years of hard slog. My first acknowledgement goes to my main supervisor and the guru of EdCAAD, Aart Bijl. His sharp mind and ceaseless energy were responsible for many stimulating exchanges which often gave me great encouragement, occasionally great depression, but always lots of good ideas. I would also like to thank his personal secretary Mrs. Margaret McDougall for her constant concern and sense of humour. I would like to express my gratitude to Sam Steel for giving me the pleasure of working with him during the early stages. He was a very strong influence upon directing the course of future work. I am grateful to David Price of the Scottish Special Housing Association for letting me rummage through his drawings, providing me with valuable source material. Thanks also to Peter Jackson for his critical comments towards the latter stages. I am very grateful to all the people I have had the pleasure of working with at EdCAAD. Last but not least, the people of the Ukrainian community who helped me to survive. This thesis is dedicated to Vasyl Stus, the Ukrainian poet, who died in a Soviet labour camp in September, 1985.

"Perhaps in an attempt at diminution, overawed by the immensity of the transcendental, we have circumscribed, questioned, and doubted too much. Yet, despite all reservations: it did occur."

[*'The Book'* in *Sanatorium Under the Sign of the Hourglass* by Bruno Schulz, 1937]

CONTENTS

Abstract	. 1
Preface	. 2
Acknowledgements	. 4
Contents	. 5

Part One: A Linguistic Approach to the Representation of Drawings

1. Categories of Representation	. 13
1.1. Drawings as Design Objects	. 13
1.2. Drawings for Communication	. 15
1.3. Symbolic vs Analogic Representations	. 17
1.4. Lines	. 20
1.5. Drawing as Language	. 24
1.6. Language	. 25
1.7. The Syntax of a Descriptive Language	. 26
1.8. Summary	. 28
2. Partial Descriptions and Plastic Representations	. 30
2.1. Partial Descriptions	. 30
2.2. Plasticity vs Exhaustiveness	. 30
2.3. Example	. 32
2.4. Summary	. 34
3. Design and Evolutionary Representations	. 35
3.1. Sketching	. 35
3.2. Examples	. 35
3.3. Representing Sketches Formally	. 38
3.4. Maintaining Consistency	. 41
3.5. Logical Representation of Sketches	. 44

3.6. Summary	. 45
4. Minimalist Representations	. 46
4.1. Introduction	. 46
4.2. Language Paradigms	. 46
4.3. Complex Objects	. 47
4.4. Inheritance	. 50
4.5. Context	. 51
4.6. Criticisms of OOPS	. 52
4.7. Summary	. 53
5. Anti-Evolutionary Approaches in CAD	. 54
5.1. Introduction	. 54
5.2. The Limitations of Expert Systems in Design	. 54
5.3. Design is not Problem Solving	. 55
5.3.1. Wholeness of Things	. 56
5.3.2. Differentiation Between Whole and Parts	. 57
5.3.3. Discreteness of Parts	. 57
5.3.4. Prior Typing of Parts	. 57
5.3.5. Correctness of Results	. 58
5.3.6. Alternative Notions	. 58
5.4. Shape Grammar	. 58
5.4.1. Questions Posed by Shape Grammar	. 59
5.4.2. Generalised Shape Grammars	. 60
5.4.3. Evolutionary Shape Grammars	. 61
5.5. Summary	. 61
6. Features of Drawing Environments	. 62
6.1. Introduction	. 62
6.2. Conventional Point-Based Systems	. 62

6.2.1. Point Location	. 63
6.2.2. Sketching	. 64
6.2.3. Grid Patterns	. 65
6.2.4. Optional Grids	. 68
6.3. Construction Lines	. 69
6.3.1. Locating Construction Lines	. 70
6.3.2. Construction Sub-Pictures	. 70
6.3.3. Parameterised Symbols	. 70
6.4. Segments	. 72
6.5. Illustration of how conventional CAD techniques do not support the kinds of operations used in architectural practice	. 77
6.6. Summary	. 85

Part Two: An Environment for the Representation of Drawings

7. A Representation Environment	. 88
7.1. Introduction	. 88
7.2. Primitives for a Representation Environment	. 88
7.2.1. Describing Things in terms of Parts and Properties	. 88
7.2.2. Kinds, Slots, and Fillers	. 89
7.3. Basic Operations	. 91
7.4. Instances and Inheritance: the operational semantics of <+>	. 92
7.5. The Expression of Variant Relationships	. 96
7.6. Deletion	. 97
7.7. Breaking of Inheritance Relationships	. 97
7.8. Copying a Description	. 98
7.9. Merging Descriptions	. 98
7.10. Renaming Descriptions	. 99

7.11. Indirection	. 99
7.12. Naturalness of Representation	. 101
7.13. Summary	. 102
8. A Syntax for the Generation of Logical Representations of Drawings	. 104
8.1. Introduction	. 104
8.2. Definitions	. 105
8.3. Graphical Primitives	. 106
8.4. Properties of Graphical Primitives	. 107
8.4.1. Length and Angle Values	. 107
8.4.2. Status Values	. 107
8.4.3. Symbolic Origin	. 108
8.4.4. Scalability	. 108
8.4.5. Rotatability	. 108
8.4.6. Handling Represented by Sense	. 109
8.5. Symbolic Dimensioning	. 110
8.6. A Context-Free Syntax For Generating Line Drawings	. 110
8.7. Indirection	. 114
8.8. Logical Objects	. 117
8.9. Drawing Machine/Representation Environment Interaction	. 118
8.10. Use of Fixing	. 122
8.11. Use of Binding to express Spatial Relationships	. 123
8.12. Summary	. 123
9. Preserving Object Topology	. 124
9.1. Introduction	. 124
9.2. The Topology Of 2-D Line Drawings	. 125
9.3. Summary	. 127
10. Transformations	. 128

10.1. Introduction	. 128
10.2. How Transformations Work	. 129
10.3. Effect of Basic Transformations on a Simply Connected Shape	131
10.3.1. Changing a Conline Angle	. 132
10.3.2. Changing a Segment Length	. 133
10.3.3. Translating a Conpoint	. 134
10.4. Generalised Transformations	. 136
10.5. Transformations, Binding, and Fixing	. 138
10.6. Use of Fixing to Effect Controlled Distortions	. 139
10.7. Use of Binding to Effect Controlled Distortions	. 147
10.8. Use of Fixing with Binding to Effect Controlled Distortions	. 150
10.9. Treatment of Transformations in Conventional CAD systems	. 152
10.10. Summary	. 156
11. Attachment	. 157
11.1. Introduction	. 157
11.2. Changing Shapes	. 157
11.3. Attaching Two Shapes	. 162
11.4. The Logic of Attachment	. 164
11.4.1. Aggregation	. 165
11.4.2. Cohesion	. 166
11.4.3. Fusion	. 166
11.4.4. Sub-Part Aggregation	. 167
11.4.5. Sub-Part Cohesion	. 167
11.4.6. Parent Aggregation	. 167
11.4.7. Parent Composition	. 168
11.5. Worked Example	. 168
11.5.1. Fixing	. 172

11.5.2. Types of Attachment	. 174
11.6. Knock-On Effects of Qualitative Changes	. 175
11.7. Treatment of Attachment in Conventional CAD Systems	. 176
11.8. Summary	. 179
12. The Flow of Control in the System	. 181
12.1. Introduction	. 181
12.2. Editing	. 181
12.3. Regeneration	. 183
12.4. Transformations	. 185
12.5. Attachment	. 187

Part Three: Conclusions, Observations, and Open Problems

13. Conclusions, Observations, and Open Problems	. 190
13.1. An Instrumental Approach to Design Systems	. 191
13.2. Relation of Drawing Machine to Representation Scheme	. 193
13.3. Depiction-Representation Equivalence	. 194
13.4. Drawing Compositions	. 197
13.5. The Structure of Design Objects	. 198
13.6. Post-Hoc Decomposition	. 199
13.7. Consistency Maintenance	. 203
13.7.1. Recognising Consistency	. 203
13.7.2. Consistency As Contradiction	. 204
13.7.3. Application of Internal Consistency Checking to Design	. 204
13.7.4. The Problems of Internal Consistency Checking in Design	. 209
13.7.5. External Consistency	. 210
13.8. Drawing as Language	. 212

13.9. Modularity of the System Components	. 214
13.10. Extension to 3-D	. 215
13.11. Conclusion	. 219
References	. 220

Part One

A Linguistic Approach to the Representation of Drawings

"In my judgement, there is no mode of exercising the faculty of observation and the faculty of accurate reproduction of that which is observed, no discipline which so readily tests error in these matters, as drawing properly taught. And by that I do not mean artistic drawing; I mean figuring natural objects, making plans and sections, approaching geometrical rather than artistic drawing..... Nothing has struck me more in the course of my life than the loss which persons who are pursuing scientific knowledge of any kind sustain from the difficulties which arise because they have never been taught elementary drawing."

[Huxley, Science and Art in relation to Education]

1. Categories of Representation

A central concern of this thesis will be with representation for the purpose of supporting the activity of designing through drawing, with a bias towards the domain of architectural drawing. It will be claimed that such an activity is linguistic in nature. The investigation will begin by looking at some examples of existing drawing practice. This will lead to some general observations and objectives for computer representations of drawing objects and operations upon those objects.

1.1. Drawings as Design Objects

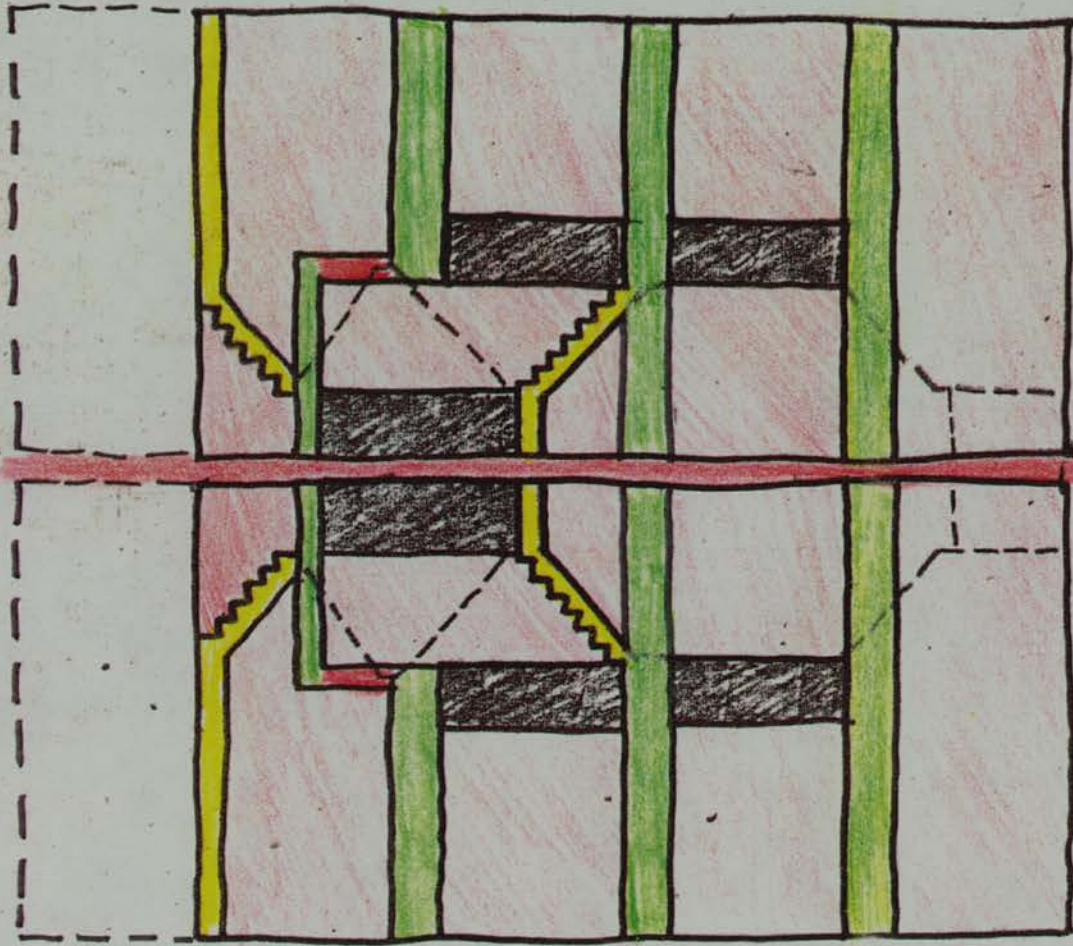
Architects produce drawings from which buildings are built. Just as buildings are seen to be the finished products which emerge from the design process, so too can drawings, which also emerge from this process, be considered as finished products, or *objects*. The difference between the two is that although drawings appear as finished products to different people during the course of a design, such as to quantity surveyors or building managers, they are also *operated upon* by architects when they design.

Figure 1-1 was one of a set of architects' drawings presented to a structural engineer after the architect had made preliminary sketches. The purpose was to seek advice on the suitability of the proposed structures. The set of drawings denoted the primary structural load-bearing walls. Also denoted was the floor material. The coloured areas represent doors (coloured black), structural walls (coloured red), and floors (coloured green and yellow).

Doors are objects within other objects (walls) and are seen to be significant from a structural engineer's point of view because of the fact that things may be bearing on those walls, and consequently the points at which bearings (of loads) can be taken may be affected. The stability of a whole wall may be affected if it has too many openings in it.

As well as those flights of stairs indicated by solid lines, other dotted lines show other flights of stairs. The dotted lines at the top represent the top floor. The structural wall did not necessarily need to go up into the top floor, so the drawing became vaguer, with more dotted lines in that region.

In constructing such drawings, the architect has to be aware of what kind of knowledge the structural engineer possesses in order to be able to distinguish such features as flights of stairs, doors, walls and openings. Both the architect and the structural engineer share a view of the world which assumes the actual existence of objects with certain properties and structural organisation.



SECTIONAL ELEVATION 4

figure 1-1

1.2. Drawings for Communication

The role of drawings in communicating an architect's perception of buildings both to other people, and to himself, makes these objects vitally important to architects. A good exposition of architecture as communication, in contrast to architecture as function, is provided by Eco [Eco, 1980]. Eco also describes how the form of communication invoked by architectural objects is more than just responding to stimuli (e.g. walking up stairs). Rather, it is communication involving 'intellective operations' within a semiotic framework.

Klein argues [Klein, 1987] for the adoption of a Saussurean approach to graphics, the essential feature of which is the arbitrary nature of linguistic signs. Klein makes the following observation:

"A system of graphical signs has to be negotiated ad hoc by participants in the communicative exchange, and can be adapted to the particular purposes of that exchange."

The exchanges that we are interested in are of the following kind. Participant A defines the meaning of some graphical representation. The information provided is sufficient for Participant B to infer (i) what the basic shape elements are, and (ii) how the meaning of complex shape elements is constructed on the basis of their primitive components. These inferred conventions dynamically define a graphical language. As a result, either participant can then go on to create new shape instances within the language such that their interpretations are governed by the just-established conventions. Moreover, new basic shape elements and new composition rules can be added to the language in the course of further interaction."

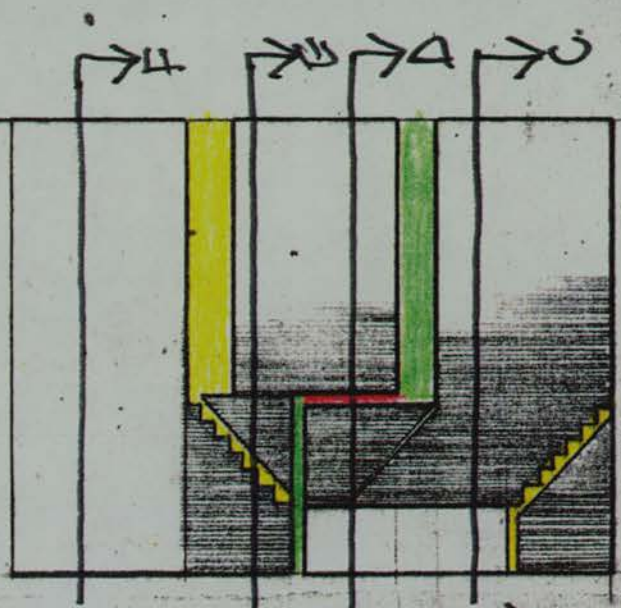
The above is an accurate description of how dialogue is conducted between people such as architects through the medium of drawings. As an illustration, consider again another drawing from the set of architectural drawings obtained from the SSHA. In figure 1-2, the floors appear relatively thick in relation to the walls, making them look out of proportion. Floors and walls were both considered to be structural elements, and therefore coloured appropriately (floors in green and yellow, walls in red). The new shape instances of floors produced by the architect emphasise the property of thickness, which he hopes will be recognised by the structural engineer when he comes to look at this drawing. The thickness of the floors may be denoting floor zones, since a floor zone will include finishes. A structural member might only be 200mm, but 300mm may be considered to be the thickness of a complete floor.

For other people to understand an architect's perception of buildings, they must already know how to interpret architects' drawings. Communication from architects depends on pre-existing knowledge possessed by recipients. If the recipient is a computer, then it too has to know about drawings. Even in the example of dumb drawing systems [Bijl, 1982], in which the computer is not expected to know what is being depicted by a

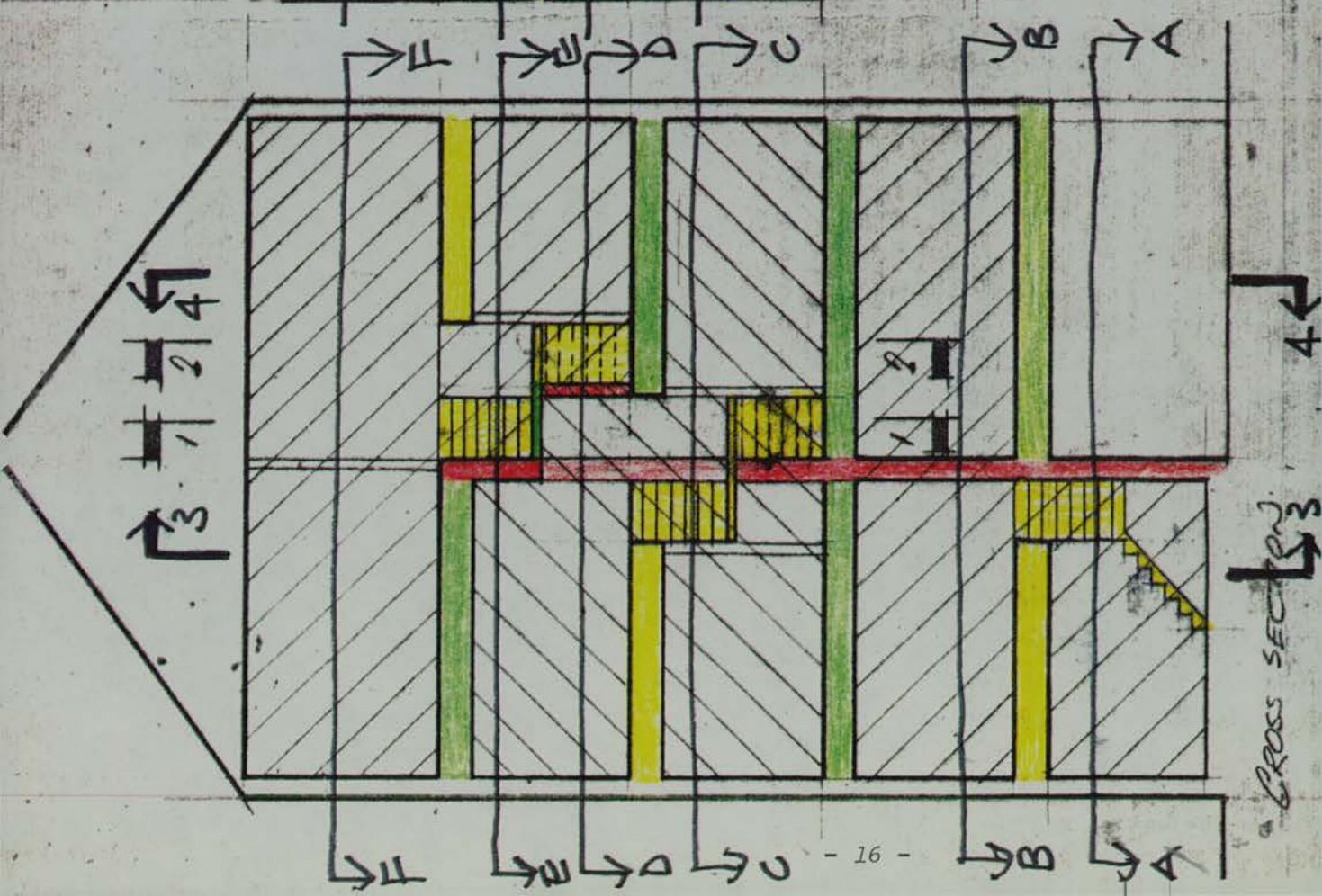


SECTION 2-2

figure 1-2



SECTION 1-1



CROSS SECTION

drawing, a certain amount of interpretation is required in order to understand intended changes to a drawing, in order to construct and edit it.

The system that will be described in this thesis, although also not an intelligent system, goes some way towards being an *integrated system*. The starting point for the development of such a system will be to devise an approach to systematically representing knowledge about drawing operations in a formal way. This knowledge can then be used by a computer program to create representations of design objects as they are depicted.

One can refer generally to any systematic representational method as a *scheme* [Hayes, 1974]. Examples of schemes include logical calculi, some programming languages, the systematic use of data structures to depict a world, musical notation, map making conventions, circuit diagrams, etc. Using Hayes' definition, a *configuration* is a particular expression in a scheme: an assertion, a program, a data-structure, a score, a map, a diagram, etc. Thus one can, formally, *define a scheme to be a set of configurations*.

All of the above configurations are formal in that there is a definite notion of *well-formedness*, from which one can establish whether a particular arrangement of marks is a well-formed configuration. We will come back to the notion of well-formedness later.

1.3. Symbolic vs Analogic Representations

A distinction is often made between representations consisting of a *symbolic* description in some language, and representations which are in some sense more *analogic* models or pictures of the things represented [Sloman, 1975]. Sloman's emphasis on analogical representations is a plea for the consideration of a wider class of languages other than those in which the only semantic primitive is the application of a function to arguments.

However, a *representation* which appears to be an analogic model at one level of analysis, may itself be represented symbolically, so that it becomes impossible to describe the overall representation as purely either one or the other. For example, a room may be analogically represented by a 2-dimensional array of values which denote the occupants of various parts of the room: but this array may itself be implemented as a list of triplets $\langle i, j, a[i, j] \rangle$, i.e. by a sort of symbolic description. It seems essential, therefore, to use a notion of *level* of representation in attempting to make the distinction precise.

Drawings can be seen as analogical representations of things. An architect's drawing, for example, conveys meaning by virtue of the fact that its parts are physically arranged in

some definite way. This physical arrangement of drawing parts is an analogical representation of the way in which meanings of certain parts are compounded to give meanings of larger parts, and, ultimately, the meaning of the whole drawing. From the perspective of constructing representations of drawings within CAD systems, it is important to develop systems which can reflect analogical interpretations of this nature.

One can view any representation scheme for drawings both at an analogic and at a symbolic level. An analogical analysis requires a kind of comparative study between two different media, whereas a symbolic treatment is often more self-contained. The former is vitally important for people such as architects who already use conventional, non-computerised drawing systems, and are interested in how their drawings relate to buildings that are going to be built. The latter, however, is the way in which computerised drawing systems are commonly presented, often at the expense of allowing users of them to express analogical mappings between depicted and logical objects. The approach adopted in this thesis is to aim for an integrated CAD system which places emphasis on supporting the establishment of mappings between graphical objects, and logical descriptions that reflect the structure of the depictions they refer to.

The type of analogical mapping of interest in this thesis therefore, is that based upon a distinction between depictions, and logical descriptions which say something about these depictions. There will be a distinction, therefore, between depicted, and abstract logical objects. This natural distinction reflects alternative modes of expression used by designers. Conventional CAD systems, however, invariably maintain distinctions between depictions and representations of depictions on the one hand, and abstract (non-graphical) objects and alternative representations for these abstract objects on the other. In such systems, there are usually only tenuous links between the two forms of representation (e.g. pointers), which makes it difficult to connect textual descriptions of graphical parts to their depictions (*figure 1-3*).

Conventional CAD System Philosophy

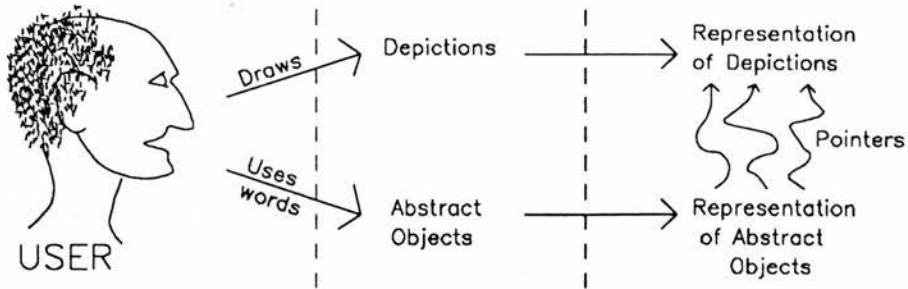


figure 1-3

The proposed strategy for an integrated system would be to support the two alternative modes of interaction with a *common* representation environment (figure 1-4).

Proposed Strategy

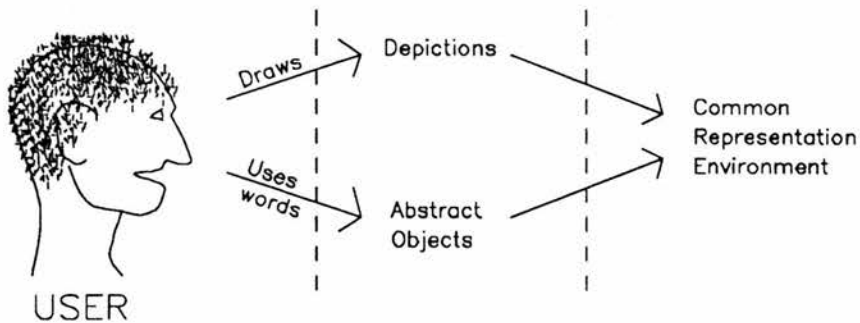


figure 1-4

It has already been indicated how the notion of analogical representation seems to depend upon some similarity between the *medium* in which a representation is embedded, and the thing represented. Consider plan drawings, for example. A plan is an imaginary slice taken horizontally through a building, thus revealing its arrangement of solids and voids. Sections are slices usually taken vertically through a building to reveal the same

kind of thing. What both plans and sections do is to draw attention to spatial relationships seen in the mind's eye. Similarly, a hierarchical logical structure intended to represent a building for example, when depicted as a graphical tree structure, can be seen to have some visual semblance to a drawing of the building in that connectivities between parts correspond in some way.

Figure 1-5 shows part of the location plan of a school. On the same sheet there may also be location sections and location elevations. All such drawings are intended to show the location of the building elements and components. These drawings are typically drawn at a scale of 1:100 and are valuable in that they are usually the first reference in any search for information. A typical location drawing will show all the rooms, walls, partitions, doors, windows, stairs and fittings, and the main dimensions. (Also indicated are the code numbers of all doors, windows, stanchions, and other repetitive features, as well as cross-references to detail drawings.)

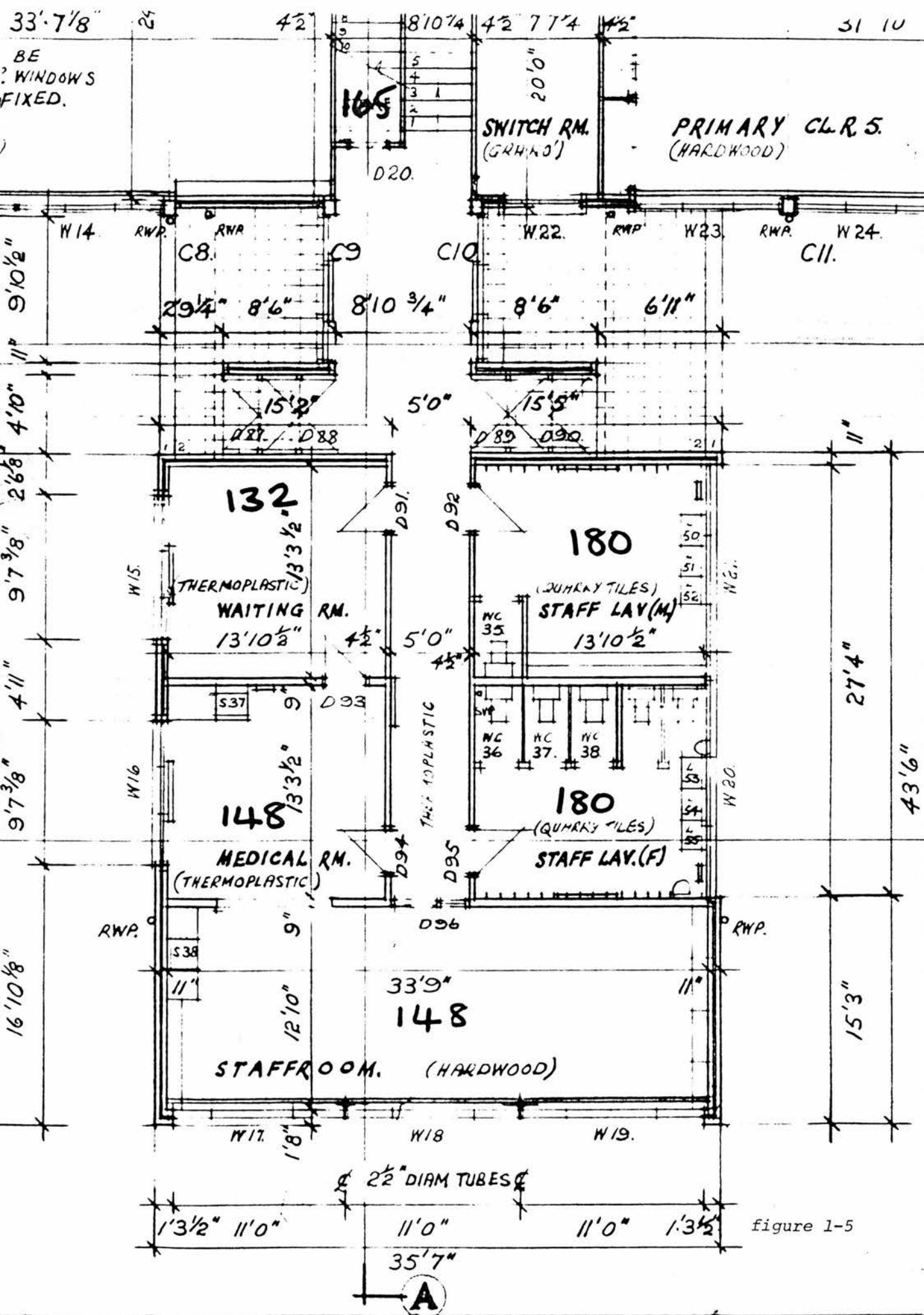
It is apparent from such an example that a plan drawing of a room is an analogical representation of the spatial relationships (in the horizontal plane) in the room, by virtue of the similarity between the 2-dimensional plane of the paper, and the 2-dimensional plane of the floor of the room. The paper is a *direct homomorph* of the room: they are the same sort of structure (2-D Euclidean space), admitting the same sorts of operations (translation, rotation, measurement). The drawing is a simplification of the reality, in the sense that certain properties present in reality (colour, exact shapes, etc.) and certain relations (the third dimension, comparisons of value) are missing in the drawing.

In contrast to plans and sections, elevations come reasonably close to depicting what the eye would see of the depicted object. An elevation is an orthographic projection in which all the features of a vertical surface of a building are projected in horizontal lines onto an imaginary vertical plane. The result is an image of the building as it could only be seen, theoretically, from an infinite distance.

1.4. Lines

One can see that architectural drawings constitute a representational scheme, consisting of a medium in which one can construct configurations of *lines* i.e. arrangements of lines in which relations exhibited analogically in the medium hold between the lines.

Figure 1-6 shows a part of an engineer's drawing of floor plans. The purpose of the drawing is to show all the structural concrete work, which consists of floors and stair flights and landings, and these generally are dimensioned in plan and section so that their



It is interesting to speculate as to how the engineer may have constructed such a drawing using the implements at his disposal. Apart from a drawing board, two of the principal items of his drawing equipment would have been a T-square and a set-square. The former would have been used in conjunction with the board for drawing horizontal lines such as line 1. The set-square would have been used to produce vertical and inclined lines such as line 2 and line 9 respectively. Lines are usually combined in order to construct figures and shapes such as wall A and wall B which represent supporting walls.



The construction of wall A and wall B and their intersection could well have been done as follows:

First draw two faint parallel horizontal lines whose exact length is not important, at the required separation. These would have been done solely with the aid of the T-square and a pencil, and are not visible on the final drawing as they would have been rubbed out. They are the lines that would give rise to lines 4 and 1, and lines 5 and 8.

On the lower line, two points would have been marked whose separation was the same as the distance between the lines, since the final figure is intended to represent the intersection of walls of the same type. Using the set-square, two faint vertical lines are drawn through these points to cross the first two lines. All such faint lines are termed construction lines, and to bring out the required shape they have to be strengthened where necessary. This is usually done by inking-in where necessary, and in this case dashed ink lines are used to produce lines 1 to 8.

Construction lines offer candidate locations for points and line segments of a finished drawing, and any pattern of construction lines may be infinitely variable. Every instance of a construction line is positioned according to some particular anticipation of drawing lines. Construction lines represent a specific anticipation of the geometric properties of real-world objects that are to be represented by a drawing.

Viewed in this way, a regular and uniform grid of lines is simply a set of construction lines arranged according to a loose anticipation of building objects. The more precise the anticipation of building objects, so the grid pattern may become less uniform, as in the case of tartan grids. Irregular construction lines are then an extension of this progression, defining the locations of actual drawing lines. Commonly, construction lines appear as light pencil lines setting out a drawing, prior to being selectively overdrawn with final inked drawing lines.

Because any pattern of construction lines is irregular, it is not possible to create a pattern by using an expression describing regular drawing operations. Unique effort is required to position each construction line. This effort is dependent on knowledge of the building objects that are to be represented by the drawing. Construction lines may be positioned with respect to other construction lines, employing geometric constructions together with distance values.

1.5. Drawing as Language

Returning to the role of drawings in communicating information, it is useful to look at how drawing expressions are constructed and interpreted by people. Analysis of the *form* of expressions sometimes comes into play when people experience difficulty in composition or interpretation. In the case of drawing interpretation, people can recognise drawing parts and then match them to things they have in mind. Conversely, for generation people think of things they want to communicate before embarking upon expressing these things in a particular medium. It is possible to separate out the construction of drawing expressions into the *syntactic form* that such expressions might take, and the *semantic content* in the mind of the person doing the drawing. According to Chomsky, however:

"In general, one should not expect to be able to delimit a large and complex domain before it has been thoroughly explored. A decision as to the boundary separating syntax and semantics (if there is one) is not a prerequisite for theoretical and descriptive study of syntactic and semantic rules. On the contrary, the problem of delimitation will clearly remain open until these fields are much better understood than they are today."

[Chomsky, 1965]

The approach to building a drawing machine described in this thesis, will assume a separation between syntactical rules that can be used to generate or represent depictions, and the semantic interpretation of syntactic structures with respect to the domain that is being depicted. Such an approach resembles syntactic analysis in natural language understanding which assumes the autonomy of syntax hypothesis [Chomsky, 1965]. This is consistent with a view of drawing which presupposes that any notion of correctness of a drawing must lie outside such a drawing machine. A good description of the sorts of problems that can arise in attempting to construct formal theories of the semantics of pictures has been provided by Goodman [Goodman, 1966; 1968]. Of course, such syntactic rules can be provided with an internal semantics for syntactic primitives and combinations thereof. In this case, there is a sense in which one can say that such a drawing machine does capture a notion of well-formedness.

Convergence upon the ambition of treating drawings as language, therefore, can be achieved if the objects that are known to any drawing machine can be defined in terms of drawing primitives from which drawing objects can be composed according to a syntax.

1.6. Language

A language is defined syntactically by a set of *primitive* elements and a set of grammatical rules which define new configurations in terms of old ones. A grammar allows one either to parse or to generate sentences of a language. A model for such a language could be provided by a set of entities acting as meanings for the primitive elements; and, for each grammatical rule, a semantic rule which defines the meaning of the configuration in terms of the meaning of its parts. This is the Tarskian idea of truth-recursion, generalised to this general notion of language.

For a semantic representation to be analogical, it is necessary for each medium-defined relation used in constructing configurations to correspond to a similar relation in the meanings. The representation is then a structural *homomorph* of the reality with respect to these relations. That is, the meanings of configurations must exist in a space which is similar to the representing medium, and the syntactic relations which are displayed directly by the symbol-configurations of the language, must mirror semantic relations of the corresponding kind. The *directness* of an analogical representation lies in the nature of the relationship between the configurations and the reality they represent (it is a relation of *homomorphism* rather than denotation). A scheme is not analogical because of any syntactic features (such as being 2-dimensional), or because of any special qualities (such as being continuous; cf. [Goodman, 1968], pp.159-164) of the worlds it describes. To emphasise how the above account fits practice, formal grammars for simple maps have been constructed, along the lines of Rosenfeld's isotonic grammars [Rosenfeld, 1971].

This thesis claims that drawing operations can be viewed as constituting a language, in which the drawings are expressions. The relationship of these expressions to reality is that the primitive elements (lines) denote features of a building according to some prescribed convention or key, and the positional relationships between elements directly display corresponding relationships between the denoted features. The choice of graphical primitives, therefore, is arbitrary. The graphical primitives described in **Part Two** (chapter 8) of this thesis were chosen because of some interesting properties which facilitate the expression of transformations upon shapes, such that parts may be included and others omitted, thus allowing controlled distortions of them. Such features make for richer communicative possibilities. Other graphical representations could have served just as well in illustrating the communicative nature of drawing, provided they could be fitted into a logical framework. The exact nature of the relationships that we are claiming are being represented will be given a fuller account in **Part Two** (chapter 9) of this thesis. It will suffice here to mention that they will be concerned with the *topological structure* of objects.

1.7. The Syntax of a Descriptive Language

Things drawn on paper can range from symbolic characters that form an alphabet, to pictorial simulations of objects perceived in the physical world. These things are visible elements of verbal and pictorial languages, they are written and read. Note that we are discussing objects as they exist on paper, and the functions of these objects in conveying messages. We start by regarding both symbolic characters and pictorial drawings as drawing objects, with possible classifications of other drawing objects in between, as shown in figure 1-7.

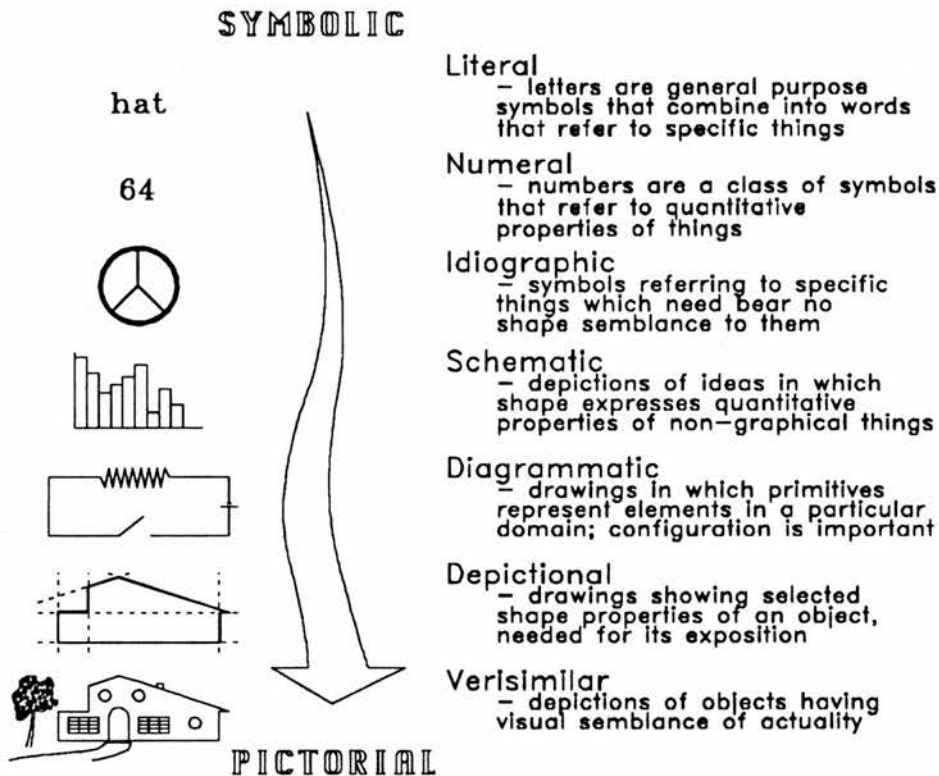


figure 1-7

Such a subclassification is necessary in order to establish the thesis that pictures of certain kinds have a syntactic structure capable of being described to a machine. One possible subclassification is as follows:

i) Literal.

These are objects that combine to form symbols, which in turn map on to specific other things.

ii) Numeral.

This is a class of symbols that map onto quantitative properties of other things.

iii) Idiographic. These are symbols that refer to some specific other thing, but that need bear little or no shape semblance.

iv) Schematic

These depict an idea, such that the shape expresses the quantitative properties of some non-graphical thing. To this category belong those schematic pictures that occur in documents, such as histograms and pie-charts.

v) Diagrammatic.

For any picture in this category, it is evident that there are primitive elements that represent elements in a particular domain. The structure of diagrams is of more importance than the elements from which they are constructed. Examples are electric circuit diagrams, chemical structure diagrams, and flow charts.

vi) Depictional

These show selected features needed for exposition. Their shape properties convey instructions. This class of pictures consists of synthesised images which purport to be representational and thus include drawings such as architectural sketches and working drawings.

vii) Verisimilar.

These contain a visual semblance of actuality. Depicted shapes correspond to shape properties of the depicted thing. This last class of pictures contains such things as photographs of natural objects. This class goes beyond the scope of this thesis.

In both the diagrams and the schematics, it is evident that there are primitive elements that represent elements in a particular domain. Thus in an electrical circuit diagram, the primitive elements are such things as capacitors, resistors, and transistors. These can be considered as being the "words" in schematic drawings. Lines joining elements directly denote, in their topological structure, the electrical connectivities in an actual circuit.

Identifying specific elements in a circuit diagram, however, is not enough to identify all the significant information in the diagram. Of more importance than the recognition of the elements themselves, is the recognition of the way in which they are connected by lines and related to each other by juxtaposition. The interconnection of the elements is the syntactical structure of the diagram. The information content of a diagrammatic drawing is

conveyed as much by its syntactic structure as by the elements within it. It is fairly evident, furthermore, that a syntactic structure does exist, and has the usual properties of a syntactically structured language, viz. that there are allowable as well as disallowed juxtapositions of the primitive elements.

For pictures belonging to the depictional category, the existence of a syntactic structuring is not as evident as it is for schematics. This category was surveyed by Gombrich, an art historian, who was concerned with a variety of forms of graphic art, drawing and sketching in particular. His view was that:

"The phrase 'the language of art' is more than a loose metaphor. Even to describe the visible world in images we need a developed system of schemata."

[Gombrich, 1960]

Gombrich's schemata are what we refer to as rules of syntax, and the point he develops through his study is that the syntactic structure of graphic art, in particular representational art (thereby including drawings and sketches), is determined largely by a set of syntactic rules.

1.8. Summary

The view that the analogical nature of drawings is compatible with their representation in terms of symbolic descriptions has been set out. The symbolic descriptions constitute a language which can be given a formal syntax independent of the semantic content which is to be conveyed by it. The analogic/symbolic distinction is captured by the notion of *levels of representation* which are distinguished by the *mediums* in which they occur. A medium may not be physically directly present, but may itself be represented by configurations in some other medium. The analogical mapping that is seen to be fundamental for the design of integrated CAD systems, and will be explored in this thesis, is that between depictions and logical descriptions of them.

Sloman's arguments for the utility of analogical representations, based on the idea that they are somehow more efficient in use than logical representations is a valid view to take, but not one that is explored in this thesis. It is valid because something that can be expressed logically, may still be more efficient to a user (for some purpose) when expressed in analogic form, e.g. a drawing may be more revealing as a drawing than its symbolic representation. The drawing may reveal things to a designer that are not captured in its symbolic representation. This is the distinction between what can be known to a human designer and a machine. However, the fundamental assumption that will be made in

this thesis is that an analogical representation (a depiction) may be embedded in a medium which is itself represented logically in some other medium (some logical representation environment in a machine). Any discussion of efficiency must take into account the computational properties of the medium. The interest in this thesis, however, is not with the efficiency of one form of representation in comparison to another. Rather, it is in the *integration* of representations of graphical depictions with representations of (non-graphical) logical objects.

2. Partial Descriptions and Plastic Representations

"A model is something to be admired or emulated, a pattern, a case in point, a type, a prototype, a specimen, a mock-up, a mathematical description - almost anything from a naked blonde to a quadratic equation - and may bear to what it models almost any relation of symbolization."

[Goodman, 1968]

2.1. Partial Descriptions

An important fact to take into consideration in any model of design drawing, is that any drawing (a configuration in a representation scheme) is, in general, a *partial* description of a design object. Consequently, the drawing will constrain the form of construction of the object, but will not, in general, uniquely determine any particular object instance. And even if a drawing did uniquely determine a design object (i.e. it was *categorical*), this fact could only be determined by analysis outside of the representation scheme itself. There is no sense in which one could say within any scheme that "this is a complete description".

One can therefore, further specify a design object by adding information ad lib. The added information can even cause the whole configuration to become inconsistent with the object that it is intended to represent. This ability to accept new pieces of information and to gradually accumulate knowledge piecemeal is essential for any representation scheme which claims to model activities such as design. Thus the idea of a *knowledge base* of separate pieces of information, to which new pieces can be added freely without a need to pay attention to control flow or other organisational matters, is very important.

2.2. Plasticity vs Exhaustiveness

This possibility of adding information is one aspect of a scheme's *plasticity*, i.e. the ease with which changes can be made to configurations in the scheme. *Plasticity* is essential for any system working on limited information in an uncertain world, as is the case for a drawing system accepting information from a user working on some design.

One could envisage situations in which one would like to be able to claim exhaustiveness for some particular aspect of a representation. For example, we might want to represent that all the relations of a certain kind, between the entities represented in a configuration, are also represented in the configuration; or that all the facts about some

entity, which are in some sense relevant to some problem or task, are present in the configuration.

According to Hayes [Hayes, 1974], an analogic representation can only be exhaustive if its semantic rules are constrained in such a way that the medium-defined relations of a configuration completely mirror the corresponding relations in the reality. That is, that a medium-defined relation would hold between subconfigurations if and only if the corresponding relation holds in the world between the entities denoted by the subconfigurations. Such a representation would be *strongly analogic*.

It would seem that, in general, such a constraint upon any representation that is intended to represent *design* objects is not feasible, since, in the first place, these objects do not have any existence in the 'real world', other than in the imagination of the designer. We would not want to make any claim of exhaustiveness of any representation of peoples' ideas.

For example, a map is strongly analogic in the sense that the 2-dimensional spatial relationships which hold between towns, rivers, etc. also hold in the map between the symbols denoting them. Maps are also often exhaustive in a stronger sense: all the entities (towns, rivers, etc.) present in the reality are denoted by symbols in the map. Thus we say of a map with a river missing, that it is wrong, not just incomplete.

An example of an analogic representation which isn't strongly analogic is a network. A relation may very likely not be displayed in it. However, a network can also be used as a strongly analogic representations if it is insisted that either all or none of the instances of a certain relation are displayed in it. A family tree is a strongly analogic representation in this sense, relative to the relationships 'child of' and 'married', for example.

Strongly analogic representations are *less plastic* than analogic or Tarskian representations, in that information cannot be accumulated piecemeal in them. To add information to a strongly analogic representation is to alter the information expressed by it. Alterations, as opposed to mere additions, raise problems of their own. This difference between *plastic* and *strongly analogic* representations is a fundamental one.

The trouble with alterations is that the information being altered may have been used earlier as a premiss in a deduction of some kind. Thus, other pieces of information which obtain their support in some sense, from the altered information, are now endangered, and should probably be re-examined. This seems to require the system to keep an explicit record of how it formed its beliefs: a history of its own thinking. This seems prohibitively expensive (of either space or time), due to exponential factors in the amount of material

generated.

Under some circumstances, it may be possible to re-evaluate a belief on criteria independent of its original derivation, but in general, one cannot avoid the problem. This seems like an insoluble dilemma.

More far-reaching alterations to a representation which one can envisage include changes to the basic ontology, to the sorts of entity to which it refers. This would involve a change to the actual grammar.

2.3. Example

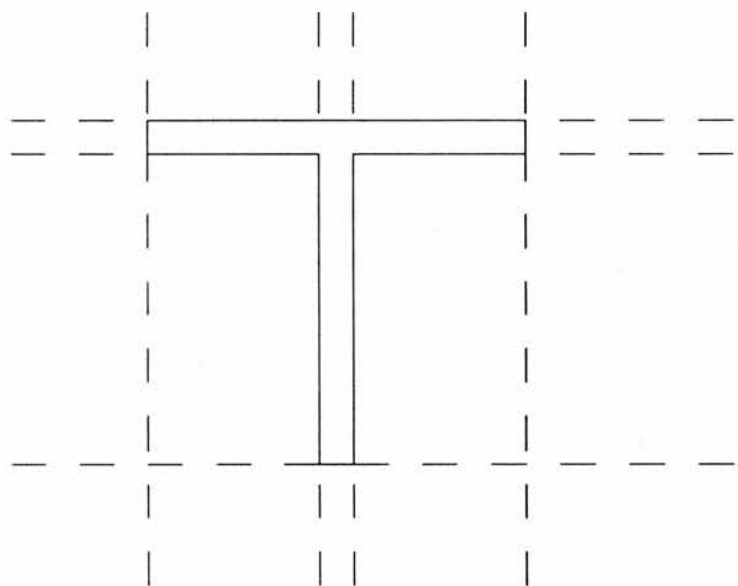


figure 2-1

Given a choice between different representational schemes, one can represent a particular thing in any of them. Consider the problem of assembling a bird table consisting of a single leg supporting a table as shown in *figure 2-1*.

One can think of at least three different ways of doing this:

- 1 By relating the coordinates of the leg and table to the overall coordinates of the bird table.

- 2 The prepositional approach, where coplanarity and "on" facts describing the parts' spatial relation are listed.
- 3 Assembly drawing, where parts of a drawing of the bird table are identified with the parts of the design object representation.

The first approach is the one that has been adopted in most conventional CAD systems, and is the standard way of planting symbols in such systems. It has the drawback that once symbols have been planted, one can no longer access parts of a symbol in a composite picture other than by editing the symbol itself. Such a model discourages partial description.

The problem with the prepositional approach is the linking of high-level predicates such as "touches" and "orthogonal to", to low-level graphical objects such as lines to which such predicates are intended to refer. The problem therefore, is one of translation between two different domains. If one says "The table is on top of the leg", this ought have an effect on the drawing. The problem of deciding exactly what effect is awkward. In the case of the bird table, how could one infer from such an expression which sides were in contact, for example. Answering such a question entails knowing a lot about the objects to which such an expression refers. The translation the other way, from facts about the coincidence of lines to what one might call "prepositional" expressions like "on top of" is equally hard. Either way, it is a difficult gap to bridge. One can envisage having two parallel and independent sets of expressions; one for directing drawing, and one as vocabulary for the data of inference. Although each set of expressions may potentially allow partial description, there remains a problem of integration. The TOPOLOGY-1 system [Gero, Akiner, and Radford, 1983] for example, fits strictly into the category of inference, since it makes no attempt to integrate the prepositional expressions that it deals with, with graphical objects.

The last approach, although low-level, is more flexible than the first, since the parts that can be included in assembly relationships are at the discretion of the user, and not prescribed by the system. It is this approach which will be explored during the course of this thesis, for two reasons. Firstly, because the flexibility of expression gained by such an approach goes beyond the expressive possibilities of conventional CAD systems to express assembly relationships, and is therefore an interesting exercise in itself. Secondly, given one of our objectives is an integrated CAD system, a way of working towards such an objective is to develop a system in which high-level assembly relationships can be mapped directly onto low-level graphical objects and properties. It is crucial to such an approach that the underlying representation should be *plastic*, since assembly relationship

descriptions themselves are likely to be partial descriptions which get modified in response to the behaviour of composite graphical objects whose descriptions contain geometrical and topological relationships. One can envisage an abstract configurational relationship existing in a representation such that it forms part of the description of the lowest-level object whose description contains parts which are terms in such relationships. This should be true whether or not the objects referred to represent graphical objects.

A way of working towards a plastic representation environment is to begin by abolishing the distinction between wholes and parts, since one would want to be able to add to descriptions of objects relatively freely without continually having to revise what counts as a whole and what counts as a part of it. This would be similar to Goodman's dismissal of the type/token distinction in favour of what he calls *replicas* [Goodman, 1968, pp.131].

2.4. Summary

The whole issue of *plasticity* in representation is crucially important for design, since we must be able to modify and improve the representations of knowledge as it changes over time. Plastic representations seem to be the most effective representational structures for the domain of architectural drawing.

3. Design and Evolutionary Representations

3.1. Sketching

Free-hand sketching is often incorporated in otherwise precisely constructed drawings. Sketching is used to convey a visual but dimensionally imprecise impression of objects forming part of a drawing. A common example is sketches of people and trees on elevation drawings. The link between sketched and constructed parts of a drawing is an interesting one. In the early stages of design, sketches form the bulk of architects' externalisations. At some stage, certain sketched objects turn into more precisely constructed drawings, whilst others may retain their impressionistic nature. In this chapter, we will look at sketching with a view to determining properties required of drawing environments that purport to support it.

3.2. Examples

Consider the sketch of the 5-storey block (*figure 3-1*) in which a tree is prominent. The architect intended there to be a row of trees between the building and the main road, in keeping with the local authority's brief. The slight increase in level off the road denotes the pavement. Another increase denotes the zone between the pavement and the building. The tree is in that zone.

The tree has been shown to remind the architect and others of the existence of the row of trees. This has certain implications in the way the root systems are allowed to grow in relation to changes in level and foundations and so on, which will come out in more detail later, and rather differently from that shown. The height of the tree in the sketch is not so significant. A mature tree might be up to 50-60 feet and it might be drawn as being 10 feet. It is important in that it reminds the architect that trees will have roots and they have to be thought about.

Consider the sketches of basic spaces (*figure 3-2*). The intention in these drawings was to see how different groups of dwellings in section might possibly work, and to have a visual note of what this means. For example, one at the lower level has a maisonette. A flat is by definition one floor, and a maisonette is 2 floors, denoted by thicker lines. There are other maisonettes above it, and small dwellings up on the roof. The line drawn across is indicative of the sort of height above street level which could be served by direct access

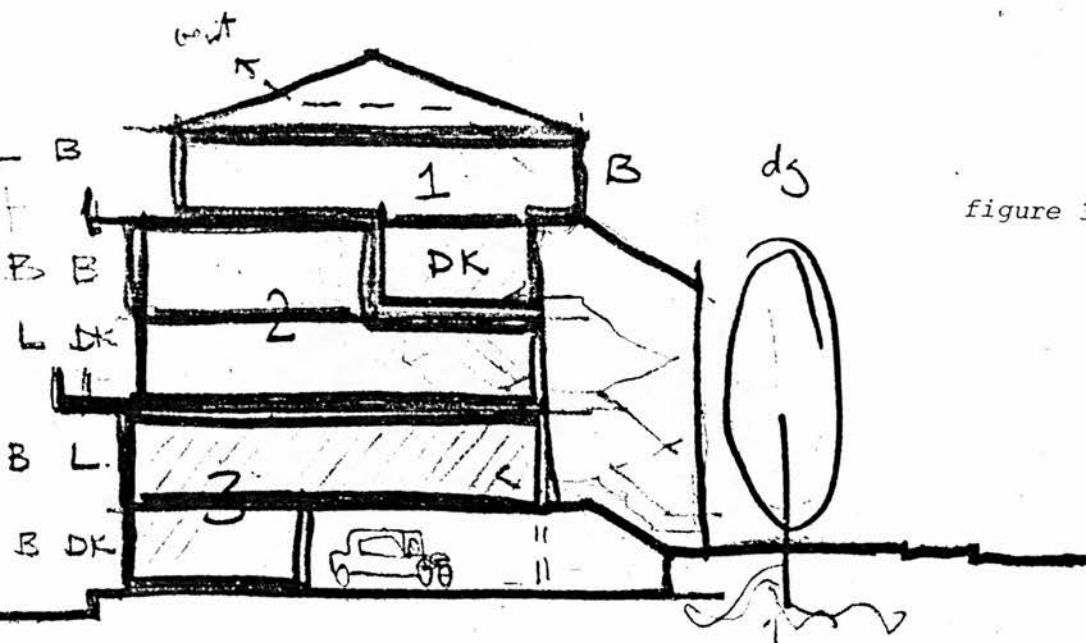
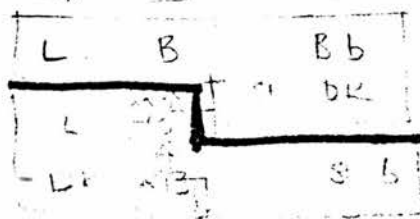
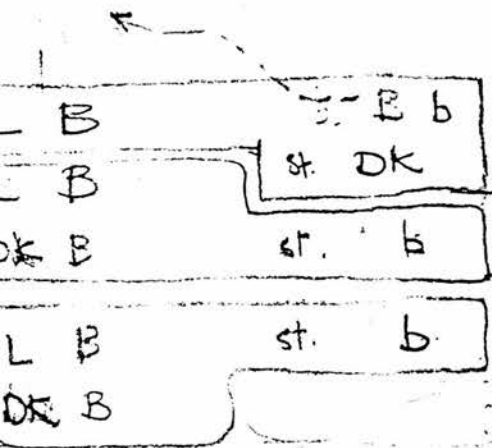


figure 3-1



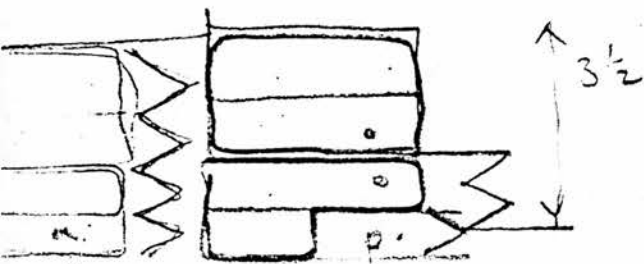
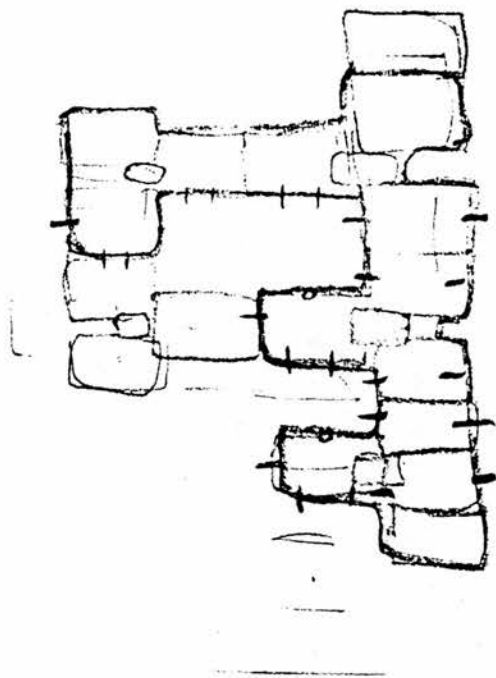
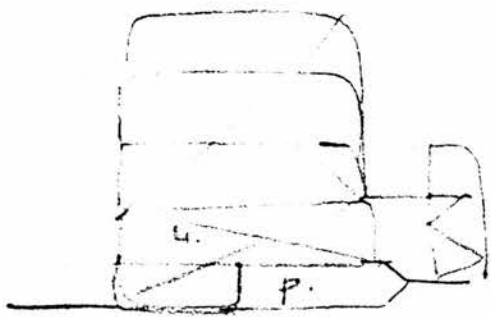
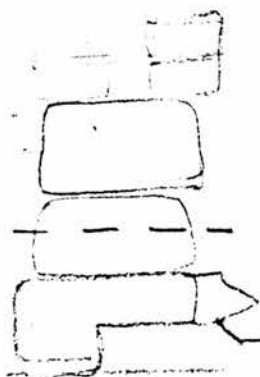
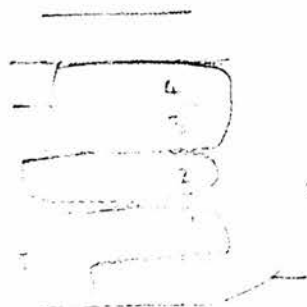
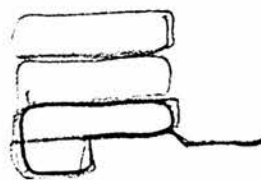
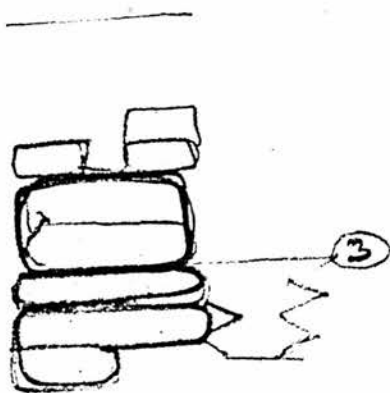


figure 3-2



3 1/2"

without lifts.

Some of the sketch drawings showed the vagueness of early ideas which were later to become more definite. In *figures 3-2 to 3-4*, spaces were shown detached from each other with adjacent zig-zag looking things. The gaps were part of a convention emphasising the boundaries of each dwelling indicating their extent. The zig-zags show the access systems. These zig-zags, having been drawn a little fainter than the rest of the picture, give a vague indication of stairs being within parts of the building. At these early stages, the architects didn't really know where the stairs were going to be - whether they were to be tacked on to the outside of the building, or partly in the building, or completely in the building. The convention used was effective shorthand for:

" there must be a stair there somewhere ".

It seemed to be more important to represent balconies earlier on, rather than doors, windows etc. (*figure 3-4*). This was probably because the basic concept of the solution required, in the mind of the architect, to include a notion of how peoples' dwellings were related to the ground and to private open space, which not only is required for drying facilities, but in relation to ideas about how people should be able to have private open space. The bottom dwelling for example, has a garden which is actually indicated by a thicker line, and an upstand at the end denoting 'fence' or 'end of private space'. The next 3 dwellings up have their private open space provided as some kind of terrace or balcony, whereas it looks as though it is indicated that the very small ones on the top probably don't get it or might have it in some other form.

3.3. Representing Sketches Formally

The preceding section has shown that sketches are the first in a series of drawings which become progressively more refined and detailed, particularly with respect to dimensions. Now we need to look at the particular problems posed by attempting to interpret sketches formally within a plastic representation environment. There are several angles to this problem. There is the problem of re-describing the representation of a sketch with a view to constructing a more dimensionally accurate drawing. Once this has been done, should the old sketch be thrown away and redrawn in its new form? Should the old sketch and the new one exist as separate objects? Should no new object be created, and the additional information 'tagged-on' in some way?

To a certain extent, the answer depends on what one wants to do. Different strategies will be capable of achieving certain goals but not others.

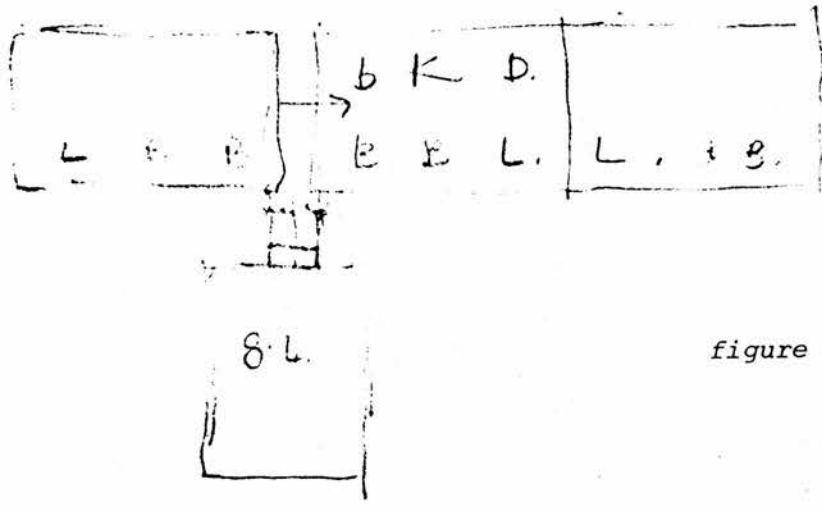
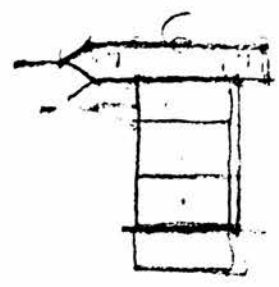
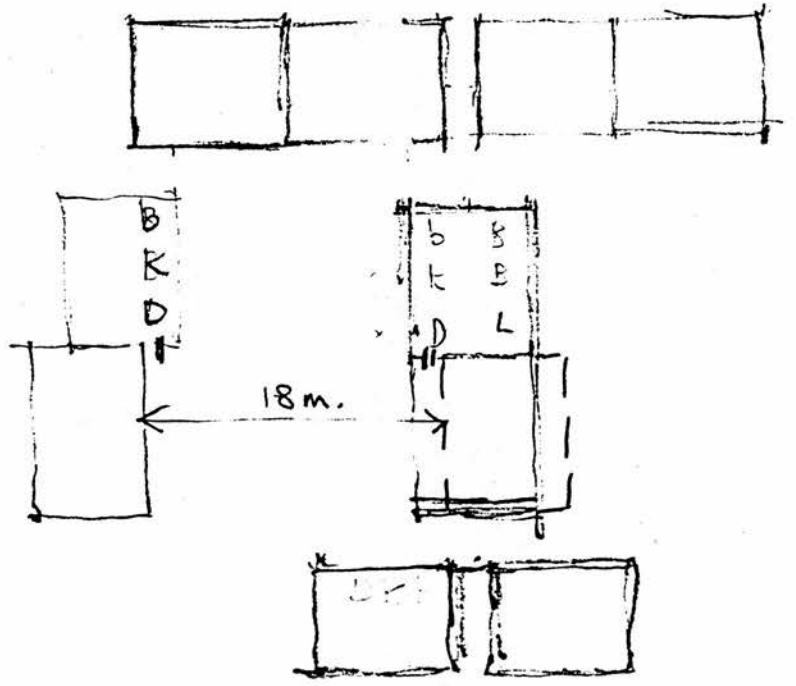
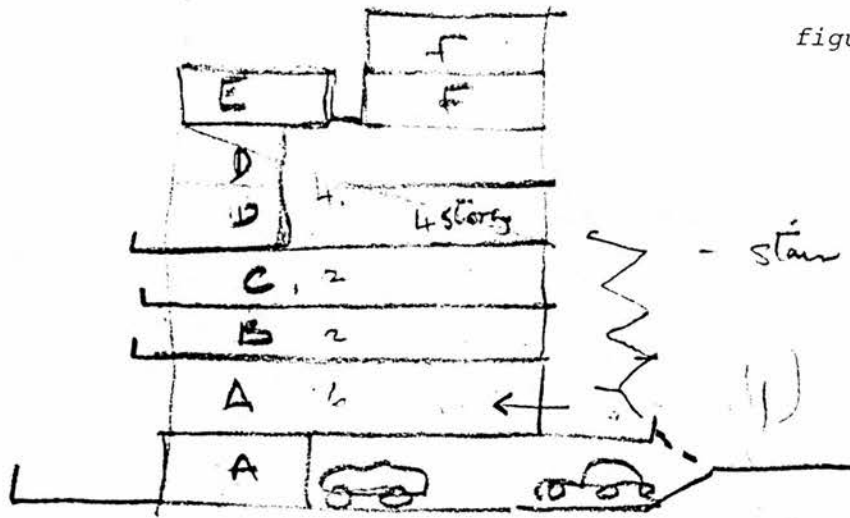


figure 3-3



4, 7

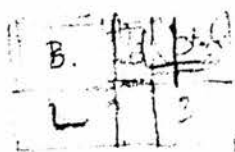
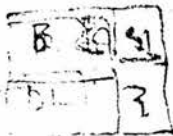
figure 3-4



stäm B C D G } 9. / stäm
B C D G }

6 flak / print

15
12
10
8
6
4
2
1



MS

35.
12 x 6 = 72
12 x 3 = 36
+ 4 x 4 = 16
108
15
40
8 x 5 = 40
20
og. 4,5
278
30
308

3.4. Maintaining Consistency

Given a sketch, and using this as a starting point for the description of a more detailed drawing, one would like at some stage to say something about the actual *dimensions* of certain parts as they are ultimately intended to appear in the design object the drawing is intended to represent. One could envisage that the original architect's brief may constrain the dimensions of certain things. The problem for an integrated CAD system which allows updates to representations both graphically *and* textually, is how to deal with non-graphical updates in which dimensions are being changed. Will the new representations subsequently be graphically realisable?

One approach would be to attempt to integrate dimensions derived by scaling intended design object dimensions, with the physical dimensions of the sketch drawing. This cannot be done by means of a straight one-for-one swap of old lines for new ones, since this would result in all sorts of irregularities such as the breaking of previously connected chains of lines, overlapping of lines, etc.

To deal with this problem in a methodical manner, one would need to know which properties of the initial sketch the user thought important and which should therefore be retained, and which were trivial and could be recalculated thus allowing the new scaled dimensions to fit. Most conventional CAD systems have at any one time a representation of a fully scaled drawing. Each time any change to the representation of the drawing is made, the relevant properties are recalculated. An alternative would be to keep a copy of the initial sketch distinct from a scaled representation of the design object. This may even mean that the object representation has a different topology to the sketch (eg. when it has extra detail that the sketch doesn't have).

The problem that is emerging here has to do with the separation of the *logical representation* of a graphical object from its depiction. In order for any drawing representation to have a depiction, it must be fully dimensioned. It cannot have flexible undetermined parts.

Given an initial sketch drawing, and a non-graphical representation of properties of an intended design object, how can new depictions be generated from such constraints? We can envisage the following possibilities:

- 1) A partially dimensioned design object representation which differs from the initial sketch.

It seems that in this case, if the drawing is a composite one (i.e. consisting of several

sub-parts), the siting of its sub-parts becomes difficult, since not all dimensions are known.

- 2) A design object representation fully and explicitly dimensioned before it is drawn.

The implication of this approach is that either the sketch drawing will have to have original dimensions overwritten (in order for it to be depicted); or, in the case of it being a composite drawing where parts of sub-parts may well be underdetermined (sub-parts being related to each other through parts that are dimensioned), it becomes impossible to draw it. Without any kind of explicit dimensioning, all one can do in such a system is to input the original sketch drawing and assign (by default) its dimensions to be overwritable.

Restricting oneself to a system in which graphical objects have to be explicitly dimensioned, means that it becomes difficult to propagate changes to certain dimensions, to others. In *figure 3-5* suppose the 8 is changed to a 10. What should happen to the dimensions of those lines adjacent to, and opposite to, this line?

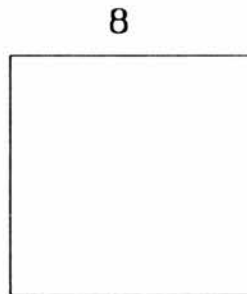


figure 3-5

Creating composite objects by bringing two or more shapes together becomes a trivial pursuit with this approach, since only lines of exactly the same length can be attached to each other. Attachment of lines of different length is impossible, since such a system would not allow for lengths to be overwritten.

- 3) Explicitly dimensioned objects, but with some dimensions fixed and some changeable.

This variant of 2) makes attachment of objects more interesting, in that it now seems to be possible to attach lines of different length, since one of the line's dimension can be overwritten by the other's value. There are other problems associated with this option, however. For example, what happens if one of the lines in the object that is to be attached has to be shortened? Do the lines that previously met this line change in any way? Or do

they just get left behind?

- 4) Draw only lines with fully determined properties, and omit the rest.

This is a strange possibility which will produce odd-looking drawings.

- 5) Draw only objects that, though they do not have all dimensions made explicit, can have them all calculated.

This possibility gives rise to parameterised shapes. In other words the representation of such objects contains explicitly-stated relations between parts of drawings which allow for the propagation of *knock-on effects* whenever changes to dimensions are made. This looks like a more promising alternative. There is no problem in integrating intended design object dimensions with current physical dimensions, since in this set up, design object dimensions which have been scaled become new explicit dimensions.

Consider *figure 3-6*. Suppose the segment labelled "x" can be calculated as 4 before drawing. Then if the 8 is changed to 10, x should be calculated to 6.

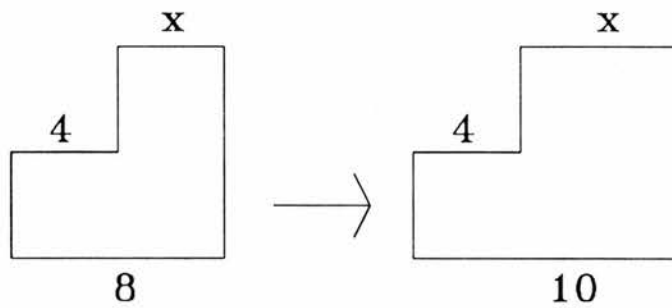


figure 3-6

Adopting this approach, it would be preferable for only the explicitly-stated dimensions to be alterable. It should be impossible to change calculable dimensions. This requirement may be hard to enforce if the calculation is difficult. If the angles of all the construction lines in *figure 3-7* are known, as well as one edge, the figure is fully known. But the calculations to do this will be awkward. To find inconsistency, one must calculate dimensions in all possible ways. How can one abandon some of these checks? Abandoning checks may lead to inconsistency. But if the system hasn't done all the necessary checks, it won't know how inconsistencies arise.

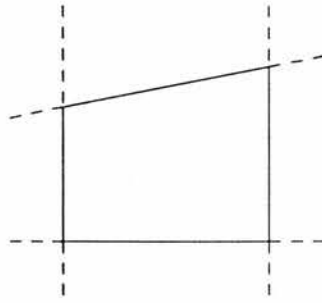


figure 3-7

An interesting question concerns whether one can abandon non-paraxial construction lines. One could handle the angle of a segment in either of the two ways shown in *figure 3-8*. Segments could run between points whose x and y coordinates on a regular grid are derived from angle value properties of the segments themselves. This seems over-elaborate, and also precludes a slanting construction line being used to align several segments. The second part of *figure 3-8* shows the same object drawn from slanting construction lines.

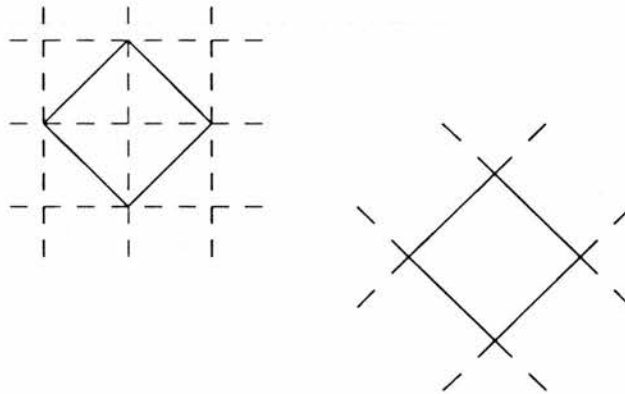


figure 3-8

3.5. Logical Representation of Sketches

By looking at the properties of sketch drawings, and listening to how people talk about them (thus revealing their thinking about them), it becomes evident that the representation environment will have to allow users of it to construct some form of logical parts hierarchy in which things can be described by reference to their parts, so that

collections of parts form descriptions. In keeping with this approach, the part/whole distinction is an unnecessarily restrictive one. Any potential whole should be regarded as a temporal state in that it is a candidate part of some other part. Things should be described by reference to partial descriptions of other things and links between things should be preserved so that descriptions of things can inherit changes to other things, thus forming a variant hierarchy. Similarly, the type/instance distinction is unnecessary. Any instance should be capable of spawning variants as further instances.

One strategy would be to name some thing and then to describe what is meant by the name by naming its parts, bottoming out with names that are unambiguously understood in the knowledge base of the recipient, either human or machine. This is a top down approach. It should also be possible to first describe things and then attach these descriptions to higher level things. This is the bottom up approach.

3.6. Summary

The problems posed for formal systems that purport to represent sketches, give indications to some of the basic properties that will be required of any representation environment for drawings. The examples of the previous sections have focused in some detail on the problems of logical consistency in the maintenance of drawing representations from the sketch stage onwards.

The examples point to a need for the representation environment to allow users of it to construct particular models of things, corresponding to their own perceptions of their world, in this case in the form of sketches. The environment has to possess knowledge of how people express what they think about things such as dimension values, in order to represent any particular instance of description, and to build up a representation from logically related but temporally disjoint sequences of user declarations. In this respect, the representation environment will need to be a plastic one, having a range of logical operators for instantiating and manipulating descriptions. Additionally, the representation environment must provide mechanisms for the expression of parametric relationships between dimension properties of drawing parts.

Models should be nothing more than temporal states of the representation environment, instantiated on demand and representing what users have told the system about their own worlds. Models may be changed as people's perceptions change, and they may be built out of collections of information supplied by different people. The representation environment, therefore, is crucial to the contribution of this thesis.

4. Minimalist Representations

4.1. Introduction

So far we have argued for the general need for representations that allow partial description of design objects and thus support the evolution of a design. If such representations are to be used by designers rather than programmers, then they should be as perspicuous as possible. One way of hoping to realise this ambition is by aiming for a representation which has a minimum of conceptual objects and operations, whilst at the same time supporting a range of descriptive activities.

4.2. Language Paradigms

Potential candidates for such representation environments are object-oriented programming systems (OOPS). Despite the fact that many object-oriented systems exist, however, there is hardly any agreement on what object-oriented programming is and what should be provided by a system that supports it. Object-oriented languages seem to lie somewhere between conventional languages (e.g. C [Kernighan and Ritchie, 1978]) and knowledge representation languages (e.g. KRL [Bobrow and Winograd, 1977], and KL-ONE [Brachmann and Schmolze, 1985]). An interesting property of OOPS is that they only possess one operation, namely, the operation of sending a message.

It is claimed [Anjewierden, 1986] that object-oriented programming is a natural paradigm for building user interfaces, and that the virtue of having only one operation is that any programming interface is potentially small. Having only one principal operation (message passing) and one kind of data type (objects), the claim is that a small amount of code is sufficient to make an object-oriented programming system available externally. Anjewierden [op.cit.], writing about PCE-Prolog, an object-oriented extension to Prolog, states that:

"Where other OOPS revert to special languages with their own syntax (Smalltalk) or a multitude of predicates (50 or so in Loops), PCE simply takes advantage of the main characteristics of object-oriented programming and provides a few predicates for sending messages and creating objects. RISC architectures for both hard- and software must have a good future."

To investigate how a 'RISC' (Reduced Instruction Set Computer, [Patterson, 1985]) approach to the design of a representation environment might work, we need to look in

more detail at the structure of design objects, the relationships between design objects, and the range of phenomena that will arise as a consequence of representing such objects in terms of such relationships. We will show how the representation environment will need to be able to handle such notions as copying, inheritance, and the use of different instances in different parts of a design. In this chapter, we will focus on the particular demands presented by detail drawings.

4.3. Complex Objects

Figure 4-1 shows part of an engineer's production drawing of a roof. It is intended to give a contractor detailed information to build a roof. It may also be used by a roof truss manufacturer to prefabricate roof trusses. *Figure 4-2* shows a part of it in more detail.

The drawing shows truss-girders running between party walls which then support rafters. The architects' intention was to create a space within the roof usable by residents for storage or as ancillary accommodation.

The components involved in the detail of part 4 include a brick cavity wall 270mm wide, and a timber wallplate (100x50) - (a piece of timber which runs along the top of the wall), seen in section, upon which the joists rest. The wallplate is shown in bold outline for clarity. In this drawing, the wall is in section, the wallplate is in section, but the pins are in elevation. The dotted line shows that the ceiling joist goes across the rafter and is nailed to it with 4 nails.

We note at this point that we can talk about the detail drawing in the same terms as the main drawing into which it is supposed to fit. There is no essential difference in the nature of these two types of drawing. From the point of view of the representation environment which will ultimately represent such drawings, they can be seen as being the same kind of object. This ties in with an interesting property of OOPS, namely, that all objects are considered to be *first class citizens* [Rentsch, 1982].

The builders use the roof detail drawing for constructing the main timberwork of the roof. There is sufficient information there to show that everything is in the right place down to the last inch of detail, for example at the eaves. When the builders construct the eaves and do the tiling in more detail, they will then refer to another architect's drawing which will show the tiles, felting, fascia boards, gutters, ledge to keep the birds out, and so on.

According to Anjewierden [Anjewierden, 1986], *complex objects* can be loosely

SIMILAR



FRAMING CLIPS AT
ALTERNATE JOINTS.



4

३

3

125 x 50

1600

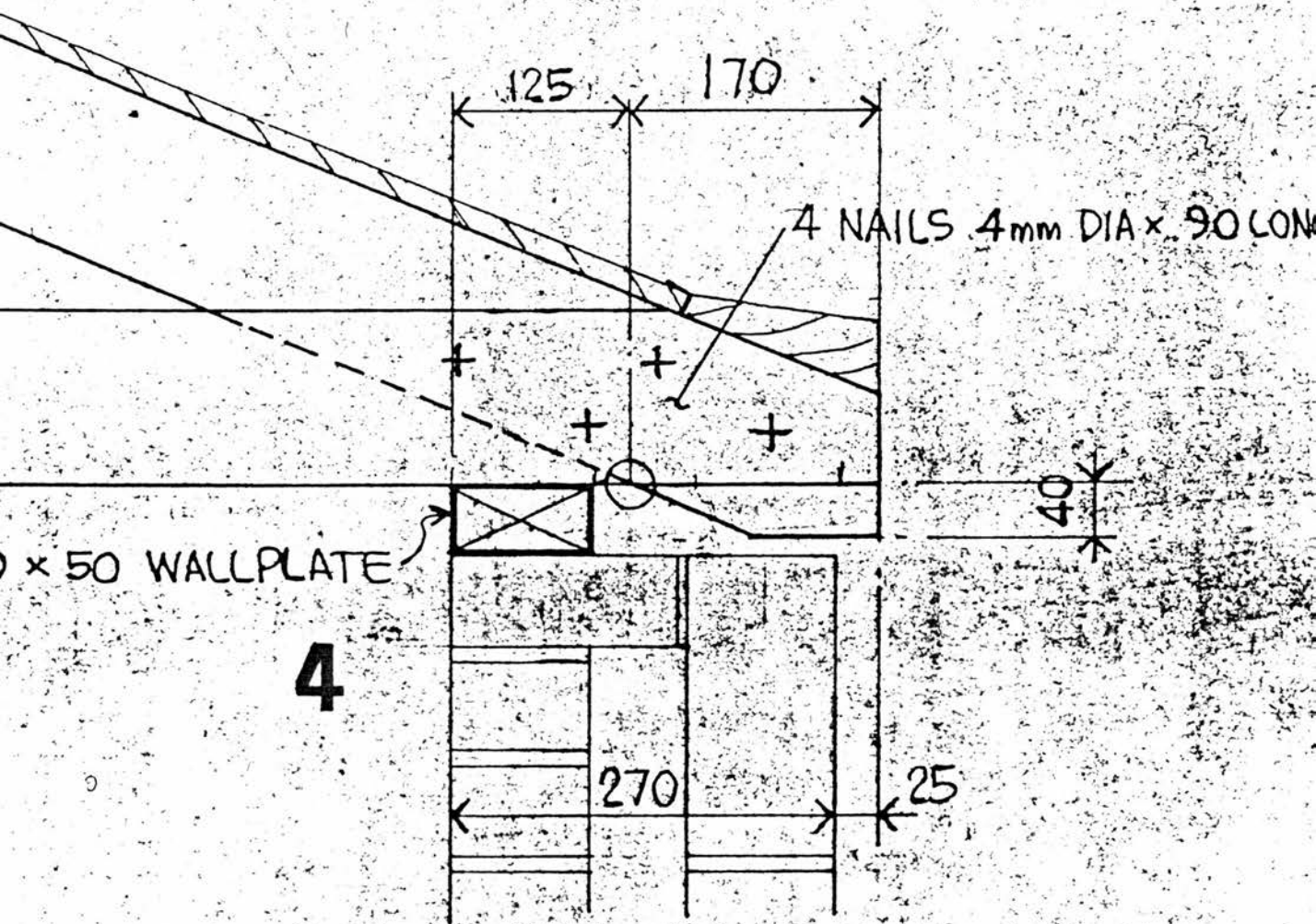
2170

170

figure 4-1

SEE DRG. No. ES/612/14 FOR ROOF PLANS

Figure 4-2



12mm DIA. BOLT - GALVANISED
50mm DOUBLE SIDED TIMBER CONNECTOR

defined as a dynamic collection of interconnected parts. In this example, the roof is a complex object consisting of truss-girders, party walls, and rafters. The detail itself is another complex object consisting of a brick cavity wall, timber wallplate, joist, rafter, and nails. It is clear that the roof is an object, but it is not so clear how the roof can be defined as a specialisation of its parts (or vice versa), since as the example clearly shows, one can always go on to describe things in ever finer detail.

Although the relations between the roof and its parts cannot be defined by specialisation, we obviously still want to treat a roof as an object, and, in a CAD system, move its representation around on a screen. Conventional geometrical transformations on a graphical representation of a roof, for example, are relatively easy to define [Foley and Van Dam (Ch.7), 1984; March and Steadman, 1971] but how can we ensure that all the associated details move as well?

Anjewierden [op.cit.] claims that *delegation* can provide an answer by describing roofs as complex objects with dynamically changing parts, and move behaviour as being delegated to each part in turn. Another example cited is the definition of behaviour to paint all roofs, say, in a certain colour. One can then tell the roof (by message passing presumably) to delegate the paint behaviour to all its parts. Anjewierden admits that PCE does not address the issue of complex objects in a structured way, but also seems to believe that inheritance is not sufficient to describe them in a natural way. This presupposes a particular notion of inheritance as manifested in certain systems, and stems from the apparent absence of procedurality associated with inheritance in such systems. Admittedly, some form of quantification mechanism such as delegation would be desirable in a representation environment. However, before this can be done, of primary importance is the description of design objects in a declarative fashion, allowing a designer to look at and modify his descriptions.

4.4. Inheritance

The function of detail drawings in general is to show the method of fixing, and the dimensions in detail. For example, the roof detail drawing shows the dimensions in relation to the wall, which is fairly critical in relation to the eaves design, and thus the way in which the tiles are put on. This would be too small if it appeared in the main drawing. Very often, it occurs that some of the parts of a detail drawing are replicated in another drawing somewhere else. There will probably be an eaves drawing which shows some of the same information.

In cases when there are natural specialisations of object descriptions as in the above example of a main drawing and its details, but when knowledge still needs to be shared between them, some form of inheritance mechanism is necessary. Inheritance is often used in knowledge representation languages to build hierarchies of related concepts [Brachman and Schmolze, 1985]. Furthermore, in design, in so far as a designer wants to relate drawings with each other, this would inevitably necessitate the *dynamic* creation of relationships across object hierarchies in any representation environment that attempts to model design drawing. Certainly, one would not want to prescribe static relationships between objects, as was the trend in many conventional CAD systems [Bijl, Stone, and Rosenthal, 1979]. Anjewierden expresses his worry that:

"In many practical situations the design and implementation of inheritance hierarchies is extremely difficult and often more a restriction than an asset."

Anjewierden introduces the idea of delegation as being a more general way to specialise classes than inheritance, and claims that delegation allows the creation of more flexible object hierarchies but also seems to introduce unpredictability.

4.5. Context

The representation of part 4 in *figure 4-1* is less detailed than in *figure 4-2*. Because of this coarse/fine distinction, we can always look at a drawing as a whole, or alternatively, we can look at a part of a drawing in finer detail, which is the case when we focus on part 4 in *figure 4-2*. What has happened in moving from the coarse detail of part 4 in *figure 4-1* to the fine detail in *figure 4-2* is that more lines have been added, notably those representing the brickwork and the locations of the nails, more dimensioning has taken place, and the new objects "wallplate" and "nails" have been introduced with explicit names.

Looking at examples such as the detail drawing above, it is difficult to see how the access mechanisms of OOPS can help to access object descriptions in order to update them. We need to be able to access object descriptions directly. But this would clearly violate the information hiding principle of an OOPS.

OOPS enthusiasts believe that the biggest advantage of object-oriented programming over procedural programming, from a software engineering point of view, is the ability to create self-confined objects which may be tested in isolation and when they are proven to work correctly under all circumstances, will continue to do so forever [Booch, 1986].

Such a strategy is counter-productive to systems which require *transparency* such that



the designer can relate properties of one object to those of another (as in detailing).

The example presented in this chapter is typical of how details are drawn, distinct from a main drawing into which the detail fits without the necessity of having a new drawing showing everything assembled in more detail. Another interesting feature of this example is the role played by the brickwork in the detail. Is the brickwork a part of the detail, or is it that a necessary function of detailing is to show the relationship between something that is not part of the detail, and the detail itself? The latter is probably closer to what was intended. In other words, detailing not only involves the addition of extra lines such as the dimension lines in this example, but may also require a change of *context* for the object in question. Relations between the object and other objects may be considered important in one context and not in another. Within the context of a conventional design practice, separate drawings are related to each other by having drawings at different scales. There will be a drawing of the whole thing to a small scale with references on it to show how separate details, which will be on a larger scale, will refer to the overall layout. There is continual detailing from a drawing of a site layout to a particular detail.

4.6. Criticisms of OOPS

Message passing in OOPS is used to customise objects to a certain application. This dynamic specialisation of objects is supposed to make programming OOPS easier and is an example of using message objects for specialisation. However, this also implies that customisation (by a programmer) takes place before a user even touches the system, thus making the end product that the user receives highly prescriptive.

In a traditional OOPS the definition of an object could be created by first defining a class for that object and then instantiating a particular instance of that object. This is a laborious procedure which is very inflexible in that when new attributes of objects need to be added, the whole procedure has to be repeated. We would like to be able to create instances of things on demand, without having to type them first.

Some object-oriented programming systems make it possible for the program to create a new class while it is running, others don't. It is absolutely necessary in design for the user to be able to interactively redefine objects and their attributes.

4.7. Summary

In this chapter, we have looked at some properties of OOPS which may prove useful in design applications, and others which may not. In particular, their usefulness for the representation of structured, complex objects in terms of a minimum of conceptual objects in the representation environment. An essential feature of most OOPS which would be positively harmful in the design context is information hiding. A designer using a CAD system can be considered as being on a par with the programmer of an OOPS. To him, the design representation must be accessible at all times, and therefore, it must be transparent.

Because of the nature of formal systems in machine environments, any user of such a representation environment will inevitably have to reconcile himself to a rationalist way of thinking. There is no reason to suppose however, that such an approach should not also be capable of sustaining the user's own aesthetic position in which the construction of an object points towards an immediate, legible geometric description describable directly in the representation environment. A way of ensuring this goal is to work towards a representation environment which has a minimum of conceptual objects.

5. Anti-Evolutionary Approaches in CAD

5.1. Introduction

The strategy for design advocated in this thesis is antipathetic to design by enumeration of geometric possibilities. In this respect, it is opposed to other strategies exemplified by shape grammar and expert system approaches to design. Having argued for the need for plastic representations to support partial description and design evolution, it becomes difficult to reconcile such descriptive representation environments with the prescriptiveness of shape grammars and expert systems. The essential feature of plastic representation environments is that they are directed towards supporting the *description* of drawing objects with a view to their further manipulation and transformation. Shape grammars and expert systems on the other hand, as used in practice, are invariably based upon highly prescriptive rule definitions. This implies that the objects to which such rules can be applied must themselves have prescribed structures. We will here take a brief critical look at these alternative approaches to design.

5.2. The Limitations of Expert Systems in Design

Expert systems [Amarel, 1980; Amarel, 1986; Lansdown, 1982] rest on the view that human experts commonly employ knowledge that they cannot express overtly in the form of complete step-wise *deterministic* procedures, when working towards some specified goal. This knowledge can be elicited from experts and can be represented in some non-deterministic, *rule-based* machine environment, typically built on inference rules of the form: if--(condition)--then--(action)--. Knowledge represented in a rule-based system can then be applied to new instances of problems, by presenting the system with symptoms and goals. The idea is that expert systems can function in place of human experts.

Design, however, is dependent on combinations of overt *and* intuitive knowledge, and the role of intuition is decisive. When expert systems are imported into the field of design, the knowledge which they are intended to handle is presented as being similar to intuitive knowledge. But there are important differences. The firm goal specifications required by expert systems cannot capture a weak anticipation of design objects. Neither can they accommodate the lack of prior knowledge of the properties that will describe such objects. Consequently, while the external products of the intuition of designers may be conveyed to an expert system, it does not follow that the system will be able to employ those products

as knowledge when dealing with a new instance of design. The role of expert systems in design is inherently limited to discrete analytic subtasks of design and cannot contribute to design synthesis. Thus expert systems cannot function in place of human designers.

Furthermore, we should not believe that it is possible to elicit knowledge from experts which they do not know they possess, then represent that knowledge within expert systems without ourselves having to understand the knowledge, and then expect these systems to use such knowledge to produce meaningful results for people. The argument presented here is not intended as a general refutation of expert systems, but is intended to moderate the ambitions associated with these systems.

5.3. Design is not Problem Solving

Bijl [Bijl, 1985a] has questioned the concepts that have been imported from problem solving fields into the field of CAD, and concluded that design is not problem solving.

A typical problem solving approach can be identified by the following necessary constituents:

- 1 a known state of being, within a single well-defined domain;
- 2 knowledge of procedures that can operate within the domain, by which a given state may be changed;
- 3 a goal expressed in terms that
 - i) specify some new state, including the conditions that have to be met by a solution,
 - ii) specify boundaries to the selection of procedures for changing the existing state.

This definition excludes design only if we add that a problem solving approach has to rely solely on overt knowledge. When we do so we also exclude many other activities not usually associated with design. This observation suggests that advances in design theory are likely to prove significant well beyond particular fields of design application.

For any problem solving process, any instance of problem has to have a start and a finish, and is contained within those boundaries as a whole and discrete thing. Typically, problem specifications and solutions can be undone by moving those boundaries. The wholeness of a problem then gets decomposed into discrete parts, as sub-problems, until parts present the conditions that are required by the available change procedures. To ease

the task of matching part instances with procedures, emphasis is placed on *prior typing* of parts. Sequences of change procedures provide solution paths. Solutions are found by aggregating results, and problems have to have single (or very few) solutions. A match between a solution and a goal can then be recognised as a correct result. This view of problem solving rests on the following notions that are important to the internal functions of problem solving systems:

- wholeness of things;
- differentiation between whole and parts;
- discreteness of parts;
- prior typing of parts;
- correctness of results.

These notions have become widely and firmly established in many fields. The effects on design are evident in CAD systems. What we see is the selective decomposition of ill-defined design practices into well-defined sub-tasks that are amenable to a problem solving approach. So we get programs that perform analytic functions to evaluate thermal performance and energy requirements of proposed designs for buildings. In most cases, these systems contribute nothing to our understanding of design synthesis.

Are the notions that support problem solving valid for design and, if not, can we formulate valid alternative notions?

5.3.1. Wholeness of Things

In general, the start and finish of a design appears to be circumstantial. The context in which design occurs is people and they decide when a design is to start and is finished. There are few overt criteria for recognising that a design is complete, and in the case of buildings there are none.

The boundaries to any instance of design are ill-defined. A design process includes responses to ever more unforeseen considerations. In the case of buildings, these considerations come from an unbounded domain of any arbitrary subset of all people. We do not know how to draw a boundary around the design interest in a building, so that we know we have the whole building.

Typically, the design of a door handle sits in the context of a door, in a room, in a dwelling, in a building, in a locality, in a town, etc. Equally, the door handle sits in the context of fire safety, affecting the ease of escape through doors, with burnt hands, and it

is relevant to social concern for welfare and health provision, etc. And the handle has to be manufactured with available machining technology, marketed and installed. We can say more about the door handle, but we don't know where to draw a boundary around it to define a complete and discrete design interest.

5.3.2. Differentiation Between Whole and Parts

Decomposition of design into parts has the effect that the parts are differentiated according to different domains, with no known relationship between domains that allows results to be aggregated into design solutions. In the case of buildings, we do not know how to add the result of a thermal performance evaluation to a result of a daylight distribution appraisal.

Design has to reconcile different arbitrary and contradictory interests in a design object, and this reconciliation cannot be achieved by overt processes alone. Parts are defined by the perceptions of different people, and their synthesis with respect to any perception of whole is dependent on contributions from those individuals.

5.3.3. Discreteness of Parts

If we recognise the existence of parts, we do not know how to define them as discrete parts. In the case of buildings, a part tends to be defined by the context of other parts. The definition will change when the context changes. It is not practical to work with discrete parts where changes to one part are likely to propagate unforeseen changes to many other parts.

5.3.4. Prior Typing of Parts

In design, part instances typically develop in unpredictable ways and are thus not amenable to prior typing. Attempts at typing either have to be undone as new instances make unforeseen demands on type specifications, or they compel conformity of instances which adds extraneous constraints on a design product.

The difficulty presented here becomes still more instructive if we also question the usual distinction between parts as objects, which may be represented by data, and parts as functions which can be performed by or on objects. It is not clear whether this is a useful differentiation for purposes of a theoretical understanding of design; it is possible to consider functions as parts of descriptions of objects.

5.3.5. Correctness of Results

Lastly, we come to the notion of correctness of results. Design goals generally do not include explicit specifications that allow us to match design products so that we can know that we have correct results. In the case of buildings, it is interesting to note that there is no abstract formal definition of a building that will enable us to tell buildings apart from other things. Nor do we have rigorous classifications of buildings that differentiate between, say, houses and offices among all other instances of buildings. In the absence of such knowledge, goals cannot be explicit and we have to accept uncertainty in our solutions.

5.3.6. Alternative Notions

What other notions can we think of that will be more appropriate to design, and, incidentally, to many other human activities that exhibit properties similar to those of design? The answers could have far reaching implications, but we are at present ill-equipped to provide them. Our thinking is too much conditioned by established concepts associated with successful problem orientated systems. We need to bring to the surface concepts that exist in other theoretical fields, such as mathematics, linguistics, philosophy, and reshape them into a theory that describes design.

Meanwhile, we can identify promising areas of research. We should think of parts as unbounded instances of things, with no notion of whole and independent of types, just parts of parts of parts.... We should try to conceive of a notion of things that uniformly embraces objects and functions, as well as activities and events. We should represent relationships between parts that will enable us to bridge across arbitrarily different domains, to work with divergent interests in design objects. We should aim for the construction of an overt systematic environment that will be able to represent any unrehearsed thing in the head of any designer. Working towards these objectives is essential if computational support of design activity is to succeed.

5.4. Shape Grammar

The view of design adopted by shape grammarians is one consisting of objects, both existing instances and those yet to be instanced, with certain explicitly defined and typically highly specialised properties. Design language then consists of such objects occurring in designs, with relationships between them that can be expressed in an algebraic environment. Formal shape grammars consist of sets of relationships between symbolic

entities plus transformation rules for instantiating new symbolic expressions [Stiny, 1975; Gips, 1975].

Shape representation of design objects in shape grammars are typically expressed as relationships between parts of shapes made up of lines (line primitives). Shape grammars are implemented as sets of *replacement rules*. Within the domain of shape grammar, design refers to spatial properties manifest in drawings. Line objects that are composed into shapes exist without reference to other things in a design domain that the lines may represent. Labels may be attached to shapes to differentiate between similar shapes, and these labels may invoke semantic information in the user, but they are only of syntactic significance within a shape grammar system. Additional design grammars dealing with non-shape aspects of design are envisaged, separate from shape grammars and interacting with shape grammars by means of translations. A design grammar is intended to generate new instances of design, within the bounds of a design language (the class of objects that is recognised as forming constituent parts of designs).

Shape grammars are based on the view that significant properties of design objects can be abstracted from existing instances of designs, and these can be understood in terms of a syntactic structure. Formal grammars can be devised which correspond to perceived abstract structures of designs [Krishnamurti, 1980, 1981]. These grammars can then be used to generate new instances of designs which preserve properties of existing good designs [Knight, 1980; Koning and Eizenburg, 1981]. The prescriptive nature of shape grammar rules reduces users of such systems to passive roles in the design process. In shape grammar systems,

"..... the mode of operation is to identify by pattern-matching opportunities to which the user reacts;"
[Wojtowicz and Fawcett, 1985],

meaning that the user can say yes or no to the system instantiation of design changes. It is this characteristic of shape grammars in use which make them more akin to expert systems than to CAD systems.

5.4.1. Questions Posed by Shape Grammar

Where do the relationships that determine shapes come from? This question is crucial since it is the declared intention of shape grammarians that grammars should perform a generative role, they should be capable of generating new instances of designs. The practice has been to study exemplars of good design, the past works of individual designers who are widely acknowledged to have been good designers, and to infer from

their drawings the abstract relationships between line objects which they are believed to have employed (implicitly or explicitly). These relationships are then interpreted as particular grammars and implemented as rule systems. Thus we have examples of newly designed Palladio villas [Stiny and Mitchell, 1978] Frank Lloyd Wright houses [Koning and Eizenberg, op. cit.], generated by their respective grammars, long after these designers have died. In our present time, it is argued that we should be able to develop grammars that encapsulate the abstract structure of designs of contemporary good designers, and that these grammars could make good designs widely available to others who would not otherwise have access to the services of good designers.

This line of argument is presented as a general and practical justification for the intellectual effort invested in studies of shape grammar, but this particular argument is weak and unnecessary. A test of this argument is to ask whether Frank Lloyd Wright, if he were alive and practising to-day, would still be designing in his style of the first quarter of this century? If the answer is yes, it should then be asked whether his buildings built now would be widely acknowledged as good designs. These questions raise two general issues that are inherent in good design. The first is the designer's reference to the world as he or she sees it, and the evolution of design responses. The second is the world's own reference to itself, how it sees itself, which changes the rules by which it judges good design. Both issues are untouched by the above general argument. This argument of shape grammarians appears to rest on a classicist view of design, a belief in absolute truths, in permanent good qualities of design irrespective of world context.

5.4.2. Generalised Shape Grammars

If shape grammars can detach themselves from particular instances of design such as Palladio villas, then we can envisage how they might be made more generally useful. Objects could be viewed more generally as lines with certain properties: linearity (straight, curved); intensity (thick, thin); continuity (dotted, dashed, solid); length (linear measure). These properties describe instances of lines and may have given values or their values may be parameterised i.e. values found from some relationship to values of other properties. Shapes could be considered as assemblies of one or more lines which are bound together by relationships that exist between them, such as connected lines, angle relationships and distance apart. Relationships instance shapes. In the general formalism of shape grammar, line objects and their relationships can be represented in a symbolic form in an algebraic environment, so that shapes can acquire mathematical properties such as symmetry, and can be subjected to algebraic functions, add, deduct, etc. to effect shape transformations. A given set of relationships targeted at a selected class of line objects constitutes a shape grammar, implemented as a set of replacement rules.

The abstract definition of shape as used in shape grammar refers to lines in 2, 3 or more space, and is independent of any notion of other objects that may be described by shapes, such as planes or surfaces, and polygonal or polyhedral objects. The abstract definition does not rest on Euclidean geometry. Shapes can be considered as representing geometric properties of physical objects, but that is a matter of choice for a system implementor.

5.4.3. Evolutionary Shape Grammars

Without negating the goal of shape grammar, we can adopt a different view. We have already argued that design, including expression of style in the medium of drawings, is essentially *evolutionary* and that any design system has to accommodate unanticipated evolutionary change. The consequence then is that a rule system must enable a user to modify existing rules and add new rules. The abstract grammars from which rules are devised must become a familiar part of the environment in which designers work. Systems then need to be devised which enable designers to formulate their own grammars and add their own rules. Designers will then be able to evolve their own design languages, employing formal definitions that give them access to the power of computers.

This proposal for evolutionary rule based systems requires a non-trivial extension to current work on shape grammars. Single systems that can support implementations of different shape grammars and permit users arbitrarily to change the rules pose fundamental questions.

5.5. Summary

In conclusion, it appears that the evolutionary approach to design first put forward in chapter 2 has advantages over the prescriptive approaches typified by shape grammars and expert systems. We reject therefore, the kind of shape grammar rule application to shapes. The core of this thesis presented in part 2 will be concerned with the manipulation of low-level descriptions of objects such as length and angle values, in a drawing environment whose geometric primitives are conlines, conpoints and segments. In other words, the strategy will be one of allowing a drawing description to evolve by changing relationships between graphical parts. This entails a representation environment which will allow for the expression of required relationships such that salient features of shapes can be maintained. The description of both the drawing environment and the representation environment will be the essence of this task.

6. Features of Drawing Environments

6.1. Introduction

This chapter will describe a range of drawing operations, including those related to grids, construction lines and segments. Each graphical operation or graphical object will be analysed first of all from the point of view of how these operations/objects are tackled in conventional CAD systems. We need to ascertain to what degree the representation of graphical objects in conventional CAD systems reflects what actually happens in architectural practice. The object of this exercise is to begin to outline a range of operations that would be required of any line-based drawing machine.

In designing a line-based drawing machine, it will be argued that the choice of operations should reflect to as large an extent as possible those used in conventional architectural practice, not merely for reasons of user convenience, but also because such operations possess certain characteristics that can be exploited in order to express relationships between graphical parts and objects, such that these relationships can then be exploited inside of a machine. The range of drawing operations, and hence the range of drawing objects upon which they operate, takes us through the process of drawing, from a blank sheet of paper to a completed drawing.

6.2. Conventional Point-Based Systems

In most forms of drawing, lines are typically defined by end points plus something that happens between. In architectural practice, architects make use of continuous straight or curved lines of varied thickness, colour, dotted and dashed lines, and blank lines. It is evident however, that in many cases, particularly in the earlier stages of a design, architects are not very concerned about the exact locations of the endpoints of lines (cf. the sketch drawings of chapter 3). An inherent feature of most CAD systems is that the position of lines with respect to other lines are determined solely by means of endpoints. This is somewhat restrictive in that it makes it awkward to express relationships such as **‘two lines are parallel and displaced by a certain amount’**, or **‘two lines are orthogonal to each other’**. Such general relationships cannot be easily expressed in systems which make use of particular instances of lines with particular endpoint coordinate values. In conventional systems, it is also difficult to maintain the consistency of an intended relationship should the properties of a line change. Typically, the only way to do this would be to redescribe

(by the deletion of existing lines and by the addition of new ones) that part of the drawing in which the relationship occurs. Ideally, one would prefer the drawing machine to maintain such consistency i.e. if the drawing changes, the relationship should still be valid.

6.2.1. Point Location

In conventional CAD systems, point location is the critical beginning of any drawing operation. The problem is how to get points to be where you intend them to be. There are essentially two responses to this question. The first comes from architectural practice, and the second is the one actually used by the majority of CAD systems.

In architectural practice, one first draws straight lines of any length anywhere on the drawing surface and then positions further points and lines by specifying geometric constructs and distance values. The most commonly used constructs are perpendicular and parallel, facilitated by the conventional use of T-squares and setsquares (*figure 6-1*). Drawings produced in this way are influenced by the chosen geometric constructs, but are dimensionally infinitely variable.

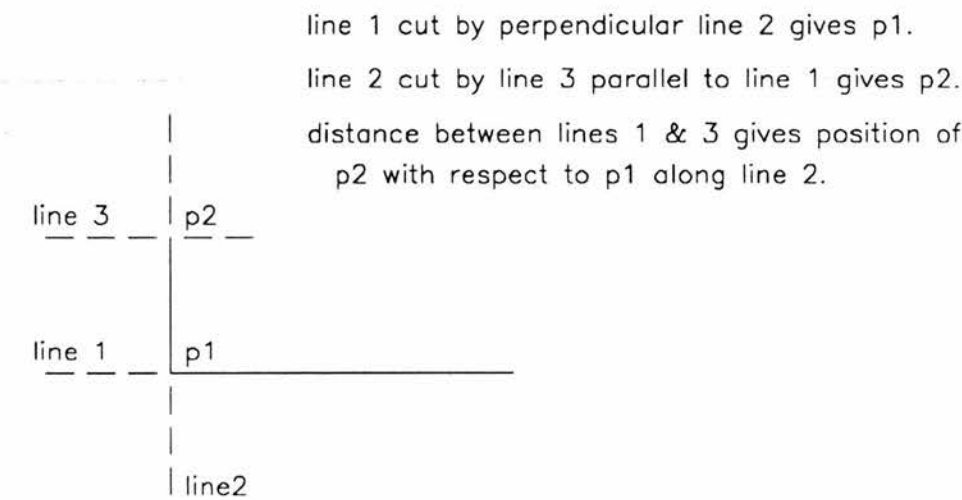


figure 6-1: Positioning points by geometric constructs.

The alternative response used in CAD systems, is to use some form of *grid* of points or lines. The function of a grid is to predefine the parts of a drawing surface that offer candidate locations for points in a drawing. Drawing then consists of planting points on selected grid locations and linking points with lines that are not necessarily on grid. Grids represent some logical ordering of dimensions that may correspond to the physical

properties of objects that constitute a building, such as a structural system or a coordinated component system. Grids may also come about as a consequence of properties of the drawing surface itself, as in the case of computer graphics displays with associated pointing operations. Drawings produced by means of a grid are influenced by the dimensional system of any given grid.

In some CAD systems, these two approaches to positioning points are combined. It is possible to position any point by specifying a geometric construct and distance value with reference to one or more grid points. If the drawing surface favours the use of grids, then positioning points off grid in this way incurs extra effort. This is a disincentive to dimensional freedom.

The virtue of a grid, for the person doing the drawing, is that the dimensional system governing the possible location of points is visible and, by locking a point on to a grid, the point can be positioned exactly. For this virtue to be effective, a grid pattern needs to be regular and, for visual clarity, the interval between grid points or lines as seen on the drawing surface needs to be as coarse as the intended drawing will permit.

There is a distinction between the physical presence of a grid on a drawing surface and the real-world dimensions which the grid represents, as in scaled drawings. The physical grid interval needs to be as large as possible, whereas the real-world dimension of the interval needs to be small enough to provide the dimensional variability appropriate to the building objects represented in a drawing. This tension results in the need to change the grid interval value as the scale of a drawing is changed. As the scale is increased, finer detail becomes visible requiring a smaller grid interval value.

6.2.2. Sketching

Accepting the idea that a grid interval value may change and if the value can be reduced close to zero, it then becomes possible to discuss free-hand sketching on blank paper in terms of grids. Free-hand sketching can be described as conforming to a grid where the interval is determined by how close the drawing instrument allows you to place two separately resolvable points, at the scale at which you are drawing. On paper, using a fairly average pencil, a 1:50 drawing may in effect be using a square grid with a real world interval of 10mm. This effect is more apparent when sketching on a computer graphics display, where the screen's resolution will set limits to the fineness of the drawing grid. By going down to very fine grids you lose the benefit associated with a coarse grid. If you scale up a sketch on a display, you will discover that points and lines are not precisely where you intended them to be.

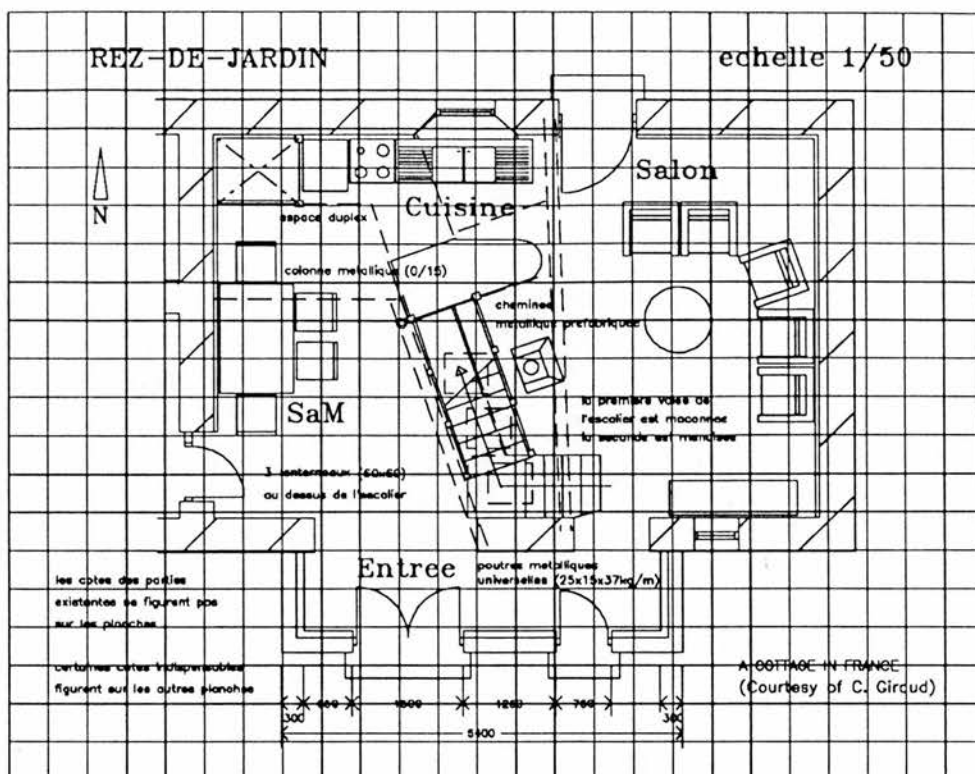


figure 6-2: Regular and uniform grid pattern.

6.2.3. Grid Patterns

Considering grid regularity and size of interval, we get square grids with uniform intervals of 300mm and 100mm for house design (figure 6-2). If the grid were irregular it would bias the possible arrangement of lines in a drawing, and if the interval were finer it would lose clarity at the scales at which general arrangement plans of houses are normally drawn (1:100 and 1:50).

Some building construction systems (e.g. OXSYS [Bijl, Stone, and Rosenthal, 1979]) present regular rhythms of dimensions which can be abstracted and represented as grids. Thus we get tartan grids and, less commonly, non-orthogonal grids (figures 6-3 and 6-4).

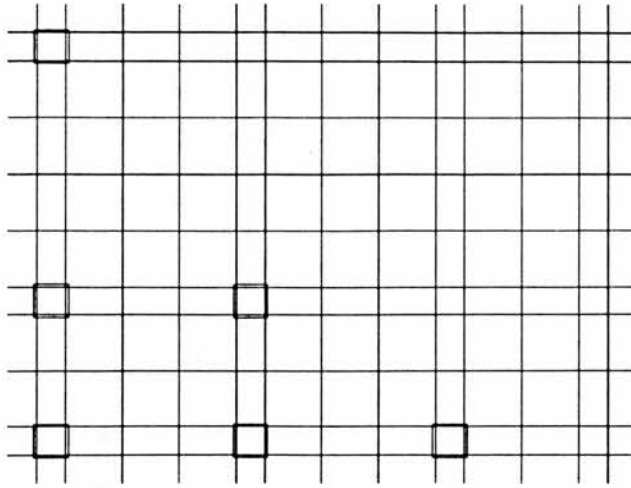


figure 6-3: Tartan grid.

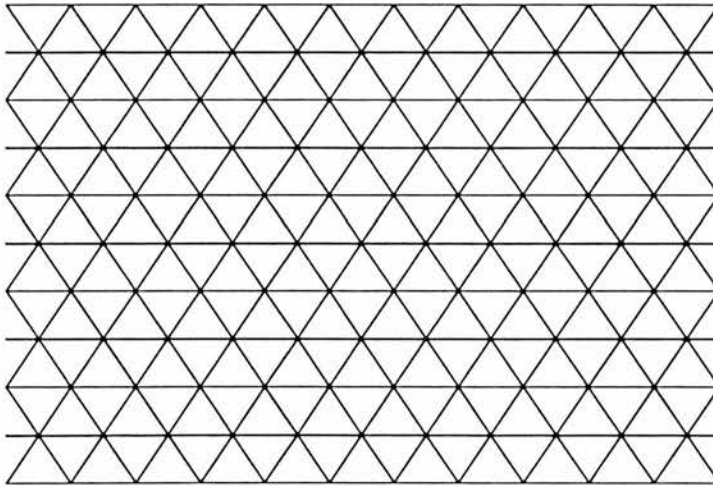


figure 6-4: Non-orthogonal grid.

When different interests are represented in a drawing, such as the structural system overlaid on the general arrangement of partitioning walls, doors and windows, we can get different grid patterns overlaid on each other. Thus we may have a tartan grid overlaid on a square grid where the tartan may be a subset of the square grid (*figure 6-5*).

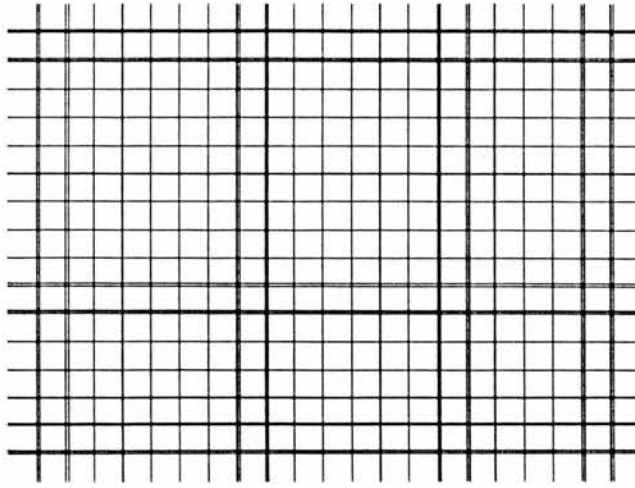


figure 6-5: Superimposed grids.

A drawing may represent a building that has parts with varied structure or geometry that call for different grids. Thus we can get discontinuous grid patterns or grid orientation over the surface of a single drawing (*figure 6-6*).

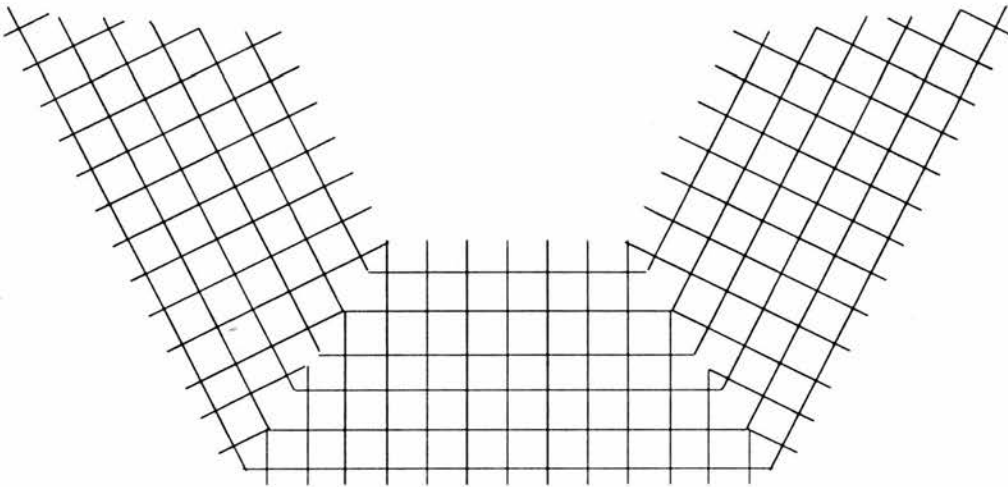


figure 6-6: Discontinuous grids.

6.2.4. Optional Grids

When considering some detail part of a building, detached from a general arrangement drawing, such as the section through a doorframe, it might well be appropriate to call up a finer grid at a larger drawing scale. In practice, given a relatively small number of dimensionally critical lines that constitute a detail, it is common to draw an isolated detail with no grid. Instead, geometric constructs and distance values are used (*figure 6-1*). The lines are thus positioned with respect to each other and to some datum which permits the completed detail subsequently to be located with respect to a grid on the general arrangement drawing (*figure 6-7*).

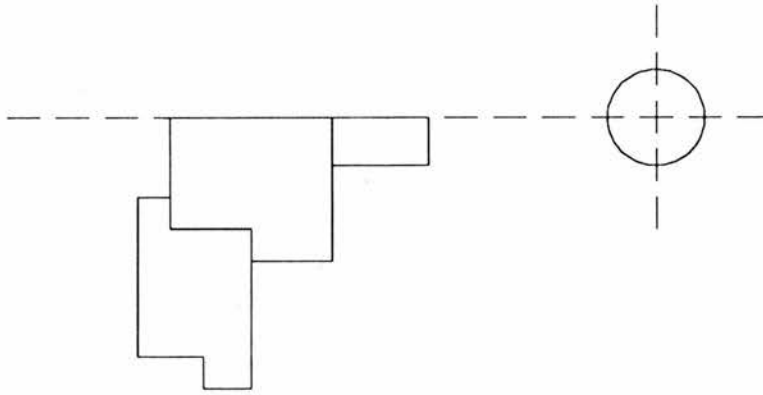


figure 6-7: Isolated detail with locating reference point.

After planting isolated details on a general arrangement drawing, it must be possible to ignore the grid when joining up the separate details, as the drawing is being developed to show complete fine detail. In the example of a doorway in a plan drawing, the section details through the door frames will contain lines that do not lie on the coarse grid, and the end points of these lines will be required as start points for further lines in order to draw the door leaf, to show the completed door.

More obviously, the "grid off" state is also required when combining sketched objects with precisely constructed drawing objects. It is necessary to be able to ignore the presence of any coarse drawing grid.

6.3. Construction Lines

In architectural practice, construction lines function in a similar manner to the grid lines of CAD systems by offering candidate locations for points and line segments of a finished drawing, but any pattern of construction lines may be infinitely variable. Indeed, grids can be viewed as a special case of assemblies of construction lines. Every instance of a construction line is positioned according to some particular anticipation of line segments. Construction lines represent a specific anticipation of the geometric properties of real-world objects that are to be represented by a drawing (*figure 6-8*).

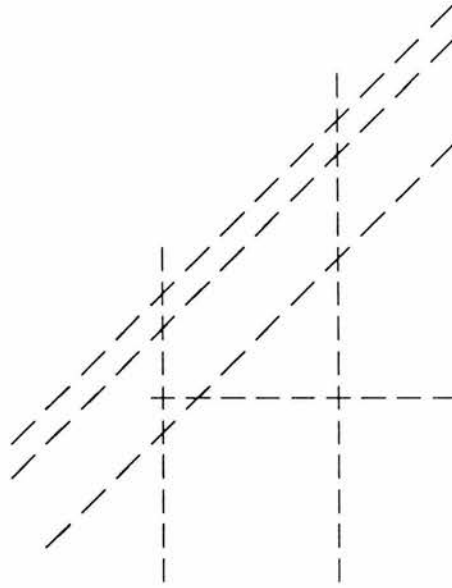


figure 6-8: Construction lines.

Viewed in this way, a regular and uniform grid of lines is simply a set of construction lines arranged according to a loose anticipation of building objects. The more precise our anticipation of building objects, so the grid pattern may become less uniform, as in the case of tartan grids. Irregular construction lines are then an extension of this progression, defining the locations of segments. Commonly, construction lines appear as light pencil lines setting out a drawing, prior to being selectively overdrawn with final inked segments.

6.3.1. Locating Construction Lines

Because any pattern of construction lines is irregular, it is not possible to create a pattern by using an expression describing regular drawing operations. Unique effort is required to position each construction line. This effort is dependent on knowledge of the building objects that are to be represented by the drawing.

The task of drawing construction lines is subject to the same considerations that were made about grids. Construction lines may be positioned with respect to other construction (or grid) lines, employing geometric constructs plus distance values, or they could be positioned directly along selected points of a regular grid.

6.3.2. Construction Sub-Pictures

If knowledge of building objects is purely in terms of their geometry, and if the purpose of the drawing is solely to convey a graphic picture of the building objects, then the location of construction lines can be decided solely with a view to establishing the location of the subsequent segments that will depict the building geometry.

Location of construction lines may be decided one at a time, or construction lines may be grouped and preassembled (anticipating composite geometric properties of the building objects) and the assemblies positioned.

As a simple example, a building may be known to possess walls which in section present a constant geometry of parallel lines with a known distance between them, but with variable length. The construction line assembly may then consist of parallel lines with fixed distance apart and indeterminate length, and subsequent segments would then be defined as segments of the construction lines (*figure 6-9*).

6.3.3. Parameterised Symbols

Commonly, the purpose of a drawing is to say something about a building during its design and construction. The purpose is to explore questions of the form 'what if', 'where is', 'what is the result of this function applied to', 'how much', relevant to the perceptions of architect, client and builder. In reconciling differences in perception, most questions lead back to the "what if" question that is central to design and has a dominant influence on drawing operations.

It is this that has lead to thinking of drawings as constituting a symbolic language.

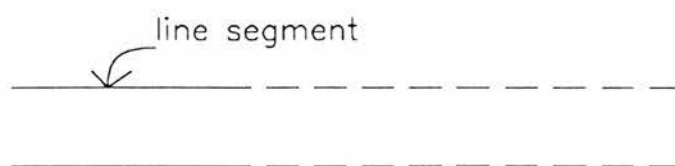


figure 6-9: Stretchable wall symbol.

Lines are formed into symbols that convey meaning and the requirement to convey meaning determines the arrangement of lines (cf. chapter 1). As people's perception of a building develops and changes during the course of its design, so the meaning read from the drawing has to change and the symbolic objects that constitute the drawing also have to change. We get the familiar view of continually changing and never finished drawings.

This view of drawing introduces uncertainty into the task of positioning lines. There is no fixed and correct target. Any line may have to be moved, subject to the need to preserve intended meaning associated with a collection of lines of which it is a part. For example, in parameterised symbols, lines are associated with collections of other lines and the parameterisations are expressed as relative values. A line might be associated with more than one other collection of lines and the relative values may be constrained, to conform to properties of building objects.

If we return to the earlier example illustrated in *figure 6-9*, we may know that buildings in general possess walls whose section can generally (but not always) be represented by parallel lines, but the distance between the lines may vary as well as their length. A parameterised symbol for walls may consist of two construction lines positioned alongside each other, and the symbol requires values to be set for the distance between the lines and (for tapering walls) the change in distance apart over a unit length (*figure 6-10*).

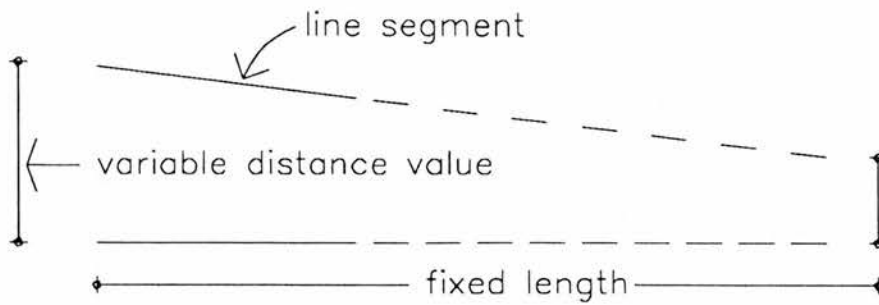


figure 6-10: Parameterised tapering wall symbol.

The values may be bounded if, for instance, we consider that a wall is no longer a wall when its thickness is greater or less than specified amounts, or if its width is greater than its length (*figure 6-11*).

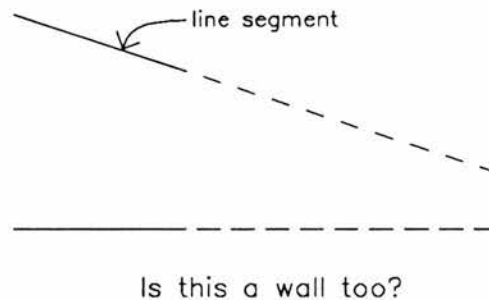


figure 6-11.

If we add to our knowledge of walls in buildings further knowledge of materials from which walls are constructed, we can then set conditional values for the parameterised wall symbol, dependent on specifying choice of material. Thus a brick wall may result in parallel construction lines drawn 300mm apart. Similarly, if we know the implications of where the wall is placed within a building on wall thickness, an inside brick wall may result in parallel construction lines 250mm apart.

6.4. Segments

Grids and construction lines have been described as devices for positioning line segments. Grids and construction lines provide points, lines and assemblies of lines from which points and line segments can be selected to construct drawings. Segments can be

described as objects that represent edges of building objects. Collections of edges may be interpreted as surfaces or volumes and further (non-geometric) properties can be attached to these derived objects to arrive at a useful description of a building.

There is a link between thinking about building objects and drawing these objects. When we consider positioning and assembling segments, this link becomes increasingly important.

Once a designer knows what a drawing is supposed to be a drawing of, the problem of actually positioning segments on a drawing surface is one of knowing which segments are needed, where, connected to which other segments. The answer lies not in the drawing operations but in the designer's perception of the building that the drawing is intended to represent. The drawing operations must then enable him to draw what he decides is necessary, and these operations must permit him to change the drawing as his perception of the building changes.

Returning to the example illustrated in figure 6-10, the representation of a wall becomes a little more complex if we consider junctions between walls of varying thickness. Consider a T junction where the upper left leg, B, is thicker than the upper right, C, and lower, A, legs (*figure 6-12*).

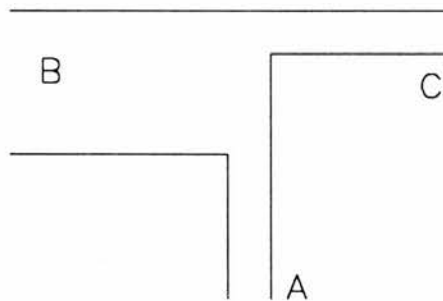


figure 6-12: T junction.

If, for any reason, wall A has to be shifted some distance to the right, what happens to the junction? Does the thicker wall B extend to the right along with wall A, or does wall A simply move along wall C leaving a kink in the new wall B, and what sort of junction or gap exists at the new junction between A and C (*figure 6-13*)?

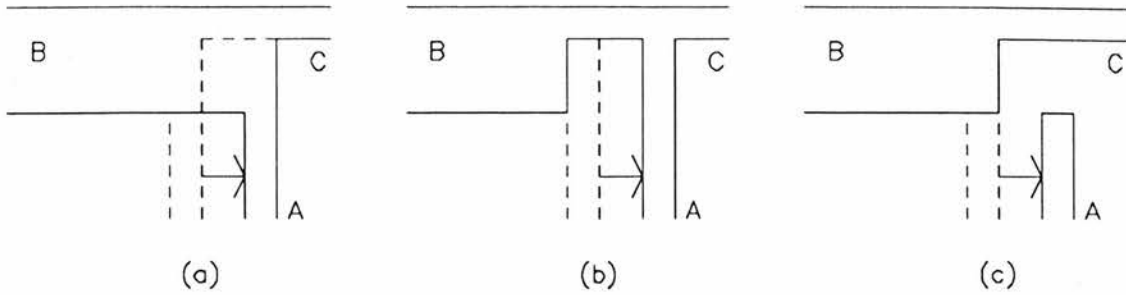


figure 6-13: Possible effects of moving wall A.

These questions can only be resolved by knowing what the drawing means, which may not be evident in the drawing itself. The answers will reflect values which are also not visible in the drawing. For instance, why is one wall thicker, and is the reason at all connected with the other two walls, and what is our tolerance of kinks and gaps? The thicker wall may form part of a support structure which is unconnected with the thinner walls but, since we do not tolerate meaningless small kinks or gaps, the thick wall may extend if wall A is shifted less than a certain amount. Thus we may decide on the first solution shown in *figure 6-13 (a)*, and our decision would have to be reflected in the drawing operations used to draw the junction before and after it has changed.

If in setting up the drawing of the junction we had first employed construction lines (*figure 6-14 (a)*) and indicated that wall A extends to the construction lines passing through a and e and intersecting at b, and if we have indicated that points a, c, and e, d, similarly define the extent of walls B and C respectively (such that B meets C along bc), and if we have also declared that line ab is common to walls A and B, then the desired junction as a result of moving A will appear (*figure 6-14 (b)*).

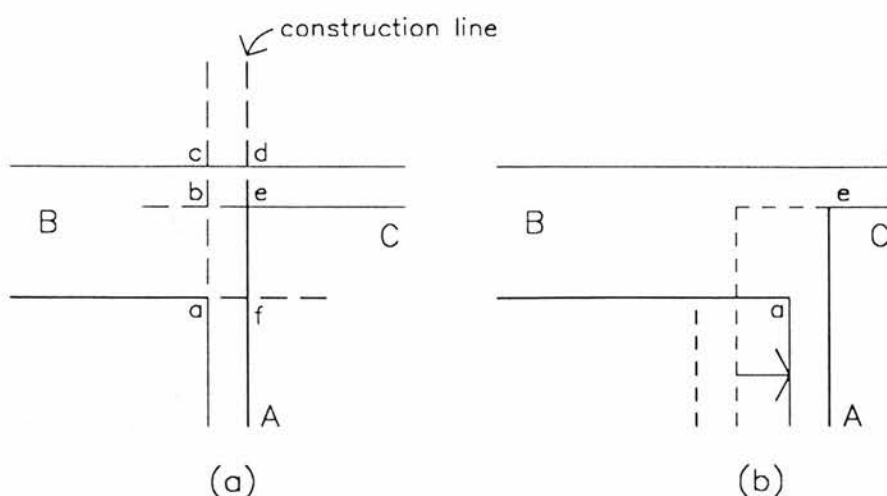


figure 6-14: Recovering construction lines to specify new junction.

Alternatively, if line af had been indicated as terminating wall A and not common to both wall A and wall B, the in figure 6-13 (c) would appear.

We have described the decomposition of the junction as though it has to be decided when the junction is first drawn, and the prior decomposition then determines the effect of any subsequent modification. But, if we regard the drawing as an object with its own integrity, independent of any building object that it may represent, then all the variants in figure 6-13 (and more) are legitimate. Someone altering the drawing, not necessarily the original draughtsman, may require a different result than the one dictated by the prior decomposition. It then becomes necessary to be able to respecify the extent and connectivity between the parts to produce a post-hoc decomposition of the junction.

This somewhat heavy explanation results from an attempt to offer a rational description of conventional drawing practice. A person drawing would resolve these issues spontaneously to achieve a specific goal, following the same logical path. Incidentally, the drawing operations outlined in the T-junction example apply equally to non-orthogonal junctions, where walls join each other at angles other than 90° .

A further step in complexity of drawing operations is introduced when we consider interpretation of collections of segments as derived objects that relate to our perception of building objects. These derived objects may be bounded subsets of the 2-D drawing surface which we understand as polygons, and the polygons may represent 2-D surfaces or section planes through building objects. Further derived objects may be assemblies of segments on the drawing surface which we understand as polyhedra representing 3-D

properties of building objects. We then have to know which segments describe which derived objects, which segments belong to more than one such object, and the nature of attachment of segments to these objects. What happens if we remove, change or add segments? Similarly, what happens if we change properties associated with derived objects that affect their edges and the corresponding segments? Again the answers lie more in the draughtsman's perception of the building than in the drawing operations that are visible on paper. To discover what is happening when a person draws it would be necessary to get people to voice their knowledge of buildings as they draw.

An illustration of the kind of problems that emerge can be found in housing site layout [Holmes, 1979]. The goal of a layout drawing is to map all the real-world objects that completely cover a site so that each object is represented by a polygon, all polygons are completely bounded by other polygons, and all the edges that describe polygons can individually be assigned properties derived from the contents of the pairs of polygons that they bound. Typically, polygons represent houses, garages, footpaths, roads, gardens and communal open space, and lines represent edge details such as kerbing between grass and tarmac. As the drawing progresses, more lines are added forming more polygons that represent more site objects. New polygons may change existing polygons by, for example, sub-division or overlapping. A footpath may sub-divide the front lawn to a house, yet the two halves of the lawn may have to be retained as a single object for purposes of assigning properties. Edges of polygons may overlap or fail to meet, requiring adjustment of the shape of objects to achieve common boundaries. A house is a hard edged object, whereas a path may be viewed as relatively soft edged and a lawn still more so. Thus if there are edge conflicts the lawn will give way to the path and the path will give way to the house. Such combined set operations and dominance ordering is common during drawing operations, even if executed spontaneously by people as they draw. Although this kind of operation is common, instances tend to be executed uniquely depending on the state of knowledge about the objects that are being drawn.

So far, we have described a range of drawing operations related to grids, and the ways in which they are treated in conventional CAD drawing systems. It has been shown how construction lines, when treated as primitive graphical objects, can not only support these conventional operations, but also, because of their function in the logical description of drawings, may be exploited by a reflective drawing system, i.e. a system which models the intentions of the designer using it. We will now illustrate the difficulties of using conventional CAD techniques with reference to the construction of an actual drawing.

6.5. Illustration of how conventional CAD techniques do not support the kinds of operations used in architectural practice

We will focus in this section on a worked example based upon one of the most predominant paradigms used by architects when they design, namely, detailing. Detailing implies understanding of relationships between parts of design drawings, and corresponding parts of detail. This entails knowledge of attachment relationships between parts. The previous chapter indicated the problems associated with the integration of fine detail into already-existing coarse sketches. In a CAD system, one would like to be able to draw a detail, and have it integrated into all the places it ought to occur. Specifying which places these are needs to be done explicitly (in the absence of a general and powerful spatial pattern recognition system). Given a detail, and a place to put it, exactly how to insert the detail can still be obscure. From chapter 4, it is evident that one needs a systematic way of mapping points and lines at one level of detail into points and lines at another level of detail. Here we will concentrate on the actual construction of a drawing of a particular detail, from which it will become evident that conventional CAD systems do not give adequate recognition to the qualities of this prevalent architectural paradigm.

This example will be developed using the Macintosh MacDraw and MacDraft programs [Apple Computer Inc., 1984]. There are of course other CAD drawing systems that offer different and variously more sophisticated functionalities, and it is difficult to select one as being truly representative of all drawing systems. The problems that will be discussed can be illustrated by the use of MacDraft and MacDraw as they are not resolved by other commercially marketed systems.

Consider the drawing of a steel roof truss shown in *figure 6-15*, [Jackson, 1962]. We can put ourselves in the position of a structural engineer intending to produce detail drawings of each of the joints shown for an architect, and focus on the joints at *b* and *c* in particular.

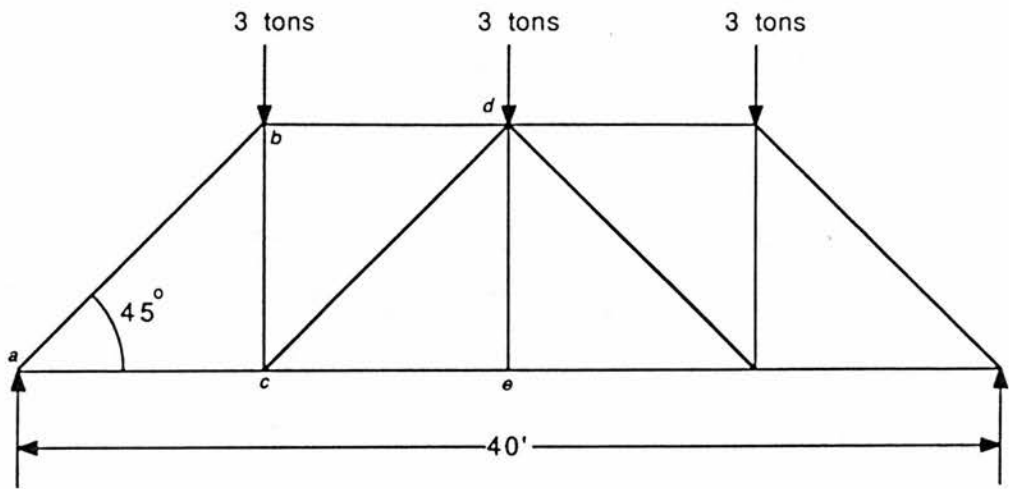


figure 6-15: a roof truss structure

We can begin by drawing a basic symbol for angle beams which are used for all the members. Suppose this is as shown in *figure 6-16*.

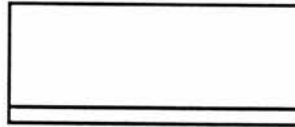


figure 6-16: Angle beam symbol

Because the beams used for different members are likely to be of different sizes, we hope to be able to scale and stretch the symbol accordingly. Having done the force calculations for each member in the structure, we can set about drawing the details for each joint. At *c* for example, four members meet. Structural calculations show that members *ac* and *bc* should both be 2.5" x 2" x 0.25" angle beams. Members *cd* and *ce* are both 3" x 2.5" x 0.25" angle beams, and can therefore be depicted by symbols of relatively greater size. The configuration consisting of the four members meeting at *c* looks like *figure 6-17*.

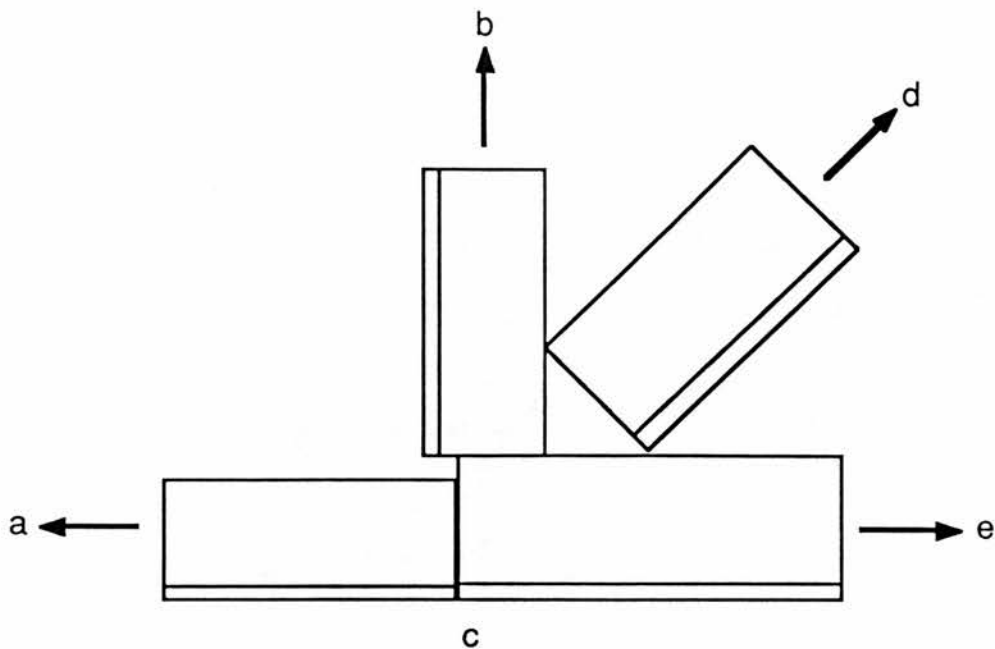


figure 6-17: Junction c

Members *ac* and *bc* are represented by rectangular symbols of the same size. The symbol depicting *bc* can be derived from the symbol depicting *ac* using the MacDraw facility for rotating objects through a fixed angle of 90° . Member *ce* can also be produced using MacDraw by scaling in one direction only in order to provide the different relative size in relation to the size of *ac*. Member *cd*, however, which meets the other members at an angle of 45° , cannot be produced solely from within MacDraw, since MacDraw (version 1.9.5) does not have a facility for rotating objects by angles other than 90° . By using the Macintosh clipboard facility, however, this picture can be transferred to be operated upon by MacDraft which allows arbitrary angle rotation, so that *cd* can be rotated by the required amount. MacDraft does have a disadvantage, however, in that the only scaling function it has is one that scales any given object equally in both *x* and *y* directions. Therefore any scaling in only one direction can be done only by using the clipboard, this time in the reverse direction, such that the picture can be operated upon by MacDraw.

Here we can identify a general problem. Any CAD drawing system offers certain operations on certain collections of lines that form pictures, or sub-pictures. These operations rest on certain assumptions about the representation of pictures. No one system will allow any user to declare any arbitrary operation on any arbitrary collection of lines, that does not conform to its assumptions. The problem presented by existing systems is that their assumptions are built into implementations at too high (or specific) a level of abstraction, resulting in conflict with users' intentions in circumstances where more general operations are required. This points to the need for representations for drawing objects and operations that allow users to describe operations as specialised or as general as is required

in any particular context. Such a representation should be implemented in a manner that remains accessible to users, so that users can declare what operations, to what degree, are to be applied to what collection of lines.

A more demanding test for the Mackintosh drawing programs is presented by the junction detail at *b*, where three members come together. From the force calculations, it has been deduced that member *ab* is of the 3" x 2.5" x 0.25" type. Members *bc* and *cd* are both 2.5" x 2" x 0.25". The drawing of the detail using conventional CAD systems such as the MacDraw/MacDraft systems can proceed as indicated above for junction *c*. The three symbols representing the members *ab*, *bc*, and *bd* can be positioned as in *figure 6-18*.

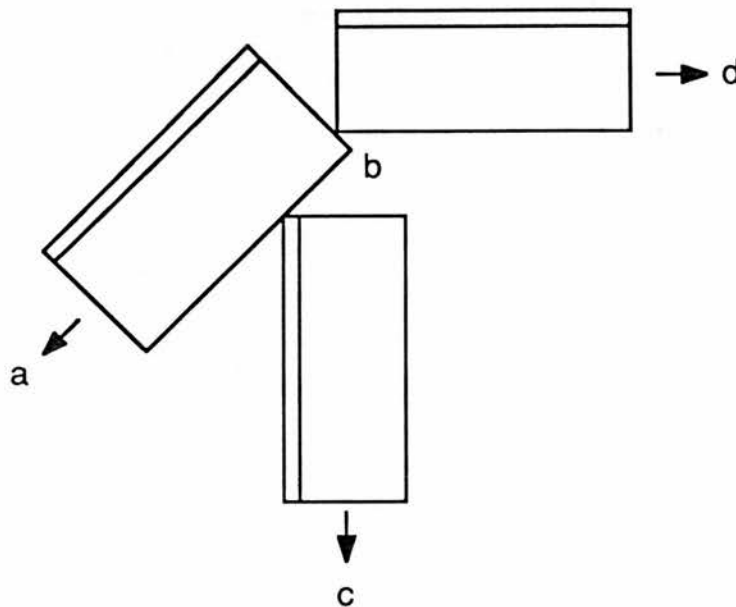


figure 6-18: Junction b

Now suppose that one of the intentions of the structural engineer is to preserve the continuity of the outer structural members, for whatever reason; the reason might be to provide added strength in joints where two or more compression members meet, or to be aesthetically satisfying. This implies that the ends of members *ab* and *bd* will have to be tapered to provide a tight fit. How does one go about depicting such a tapering using conventional systems? One would intuitively like to grab hold of one corner of a symbol and pull it out a little, at the same time leaving the other endpoints of lines that meet at this corner fixed. Once a taper of a certain angle has been obtained on one of the members, one might want to extend the tapered end further whilst preserving both its angle *and* attachment relationships between connected segments. This process could then be repeated for the other member, ensuring that the tapered end is parallel to the tapered end of the first member. A more sophisticated system such as MacArchitron [MacArchitron, 1987] provides a 'move point' function which allows an endpoint of an existing segment to be moved, together with a 'move node' function, which allows a node to be moved to a new

origin by shifting the segments attached to it. However, neither of these functions allows one to further describe other properties that one might want to preserve throughout the transformation (such as an angle or length value for example), nor to selectively include or exclude any segment from the transformation. The intention of moving a node whilst preserving the angle value of one of the segments at this node, and at the same time preserving attachment relationships between segments, is one that is not handled by conventional CAD systems. Such a goal involves a non-standard way of transforming graphical objects. Typically, the members *ab* and *bd* would have to be redrawn as newly defined pictures showing tapering ends.

The general problem that is identified here is the problem of applying transformations selectively to parts of symbols, or to selected constituents of sub-pictures. Systems need to support arbitrary partial transformations. An implication is that the logical representation of drawing objects and transformation operations must allow a user to *describe* an operation with reference to parts that are to move, in a context of parts that are to remain stationary.

In redrawing the members *ab* and *bd*, these had to be drawn as sets of four lines (no longer Mac boxes). For member *ab*, this procedure had to be carried out prior to rotation to its correct position. We then have a new version of *figure 6-18* which is logically different from it. Only now are we in a position to delete the ends that need to be modified, as shown in *figure 6-19*.

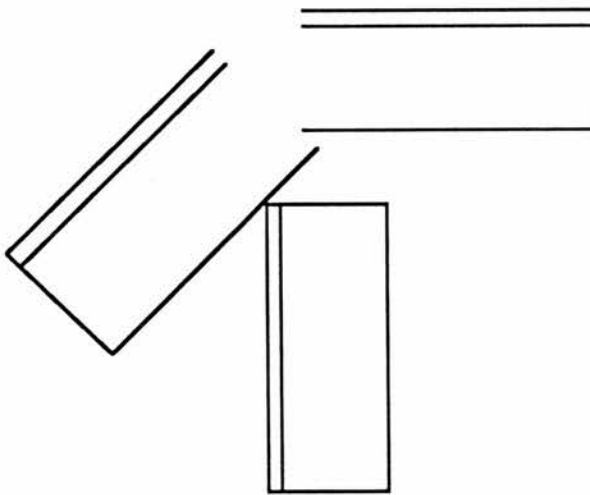


figure 6-19: Junction b redrawn

The question of where to put the new ends is problematic. An apparently obvious place to put them is at the bisection of the angle between the two members. The angle between the former ends of the two members was 45° , resulting in a bisection of 22.5° .

The principle of bisection for drawing tapers is made use of in more sophisticated systems such as MacArchitron and Autocad [Autocad,1987]. In MacArchitron, this is known as the 'L-Join' function, and in Autocad, as the 'trace' command. Suppose, however, that the user realises that machining of a 22.5° angle is very difficult, but that it is known that standard machined tapers are available at intervals of 15° . A possible solution that would allow members *ab* and *bd* to meet as required would be to put a 30° taper on *bd*, and a 15° taper on *ab* (*figure 6-20*).

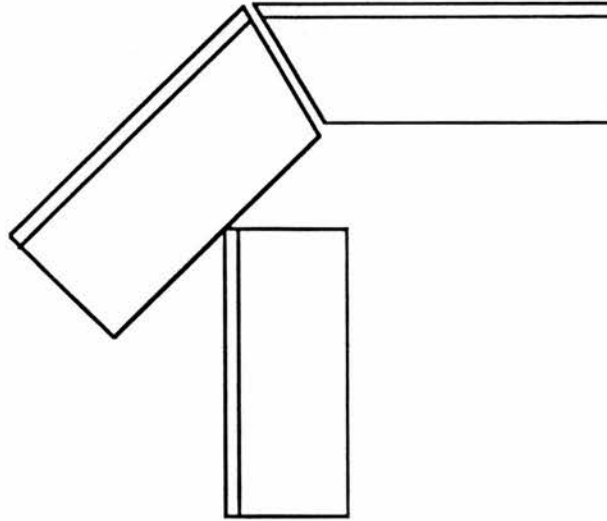


figure 6-20: Tapered Ends

The actual placement of the new tapered ends is problematic using conventional CAD systems such as MacDraft/MacDraw, since these ends will certainly not meet lines that were previously adjacent to them at the same points. The technique that was used in the construction of *figure 6-20* was to first extend in length the outer lines of each member along their current angles until they meet (indicated as dotted lines in *figure 6-21*). The tapered ends can be located by drawing a construction line of screen angle 120° from this intersection point, as shown in *figure 6-21*, after which the inner lines of each member have to be redrawn. To ensure that lines meet, one has to rely upon eyeballing unless locking mechanisms are provided (e.g. the 'L-Join' or 'T-Join' functions in MacArchitron). Finally, the extended lines have to be redrawn at the correct lengths in order to obtain *figure 6-21*.

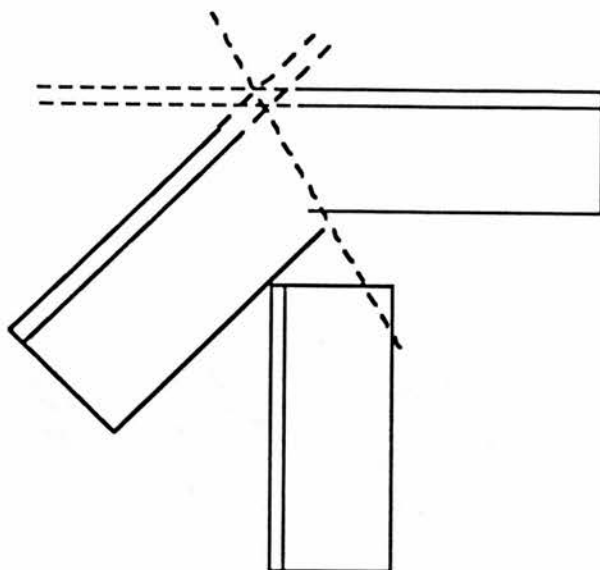


figure 6-21: The Construction of Tapered Ends

By adding detail such as rivets, plates, and lines of force, we get figure 6-22.

Suppose now that the angles of tapers on both members have been evaluated, and we later want to produce the detail drawing for the junction at *f*. It would be useful to be able to make use of the angles that were evaluated at *b*; in other words, to parameterise the angles of tapers at *f* in relation to the angles of tapers at *b*. Parameterisation might also be required at either of these joints, by describing one of the angles with reference to the other, such that when the angle of one tapered end changes, so does the other.

There is a general difficulty in specifying intended spatial relationships in conventional CAD systems, which stems from the type of representation they use. The representation of objects used in most conventional CAD systems consists of lists of Cartesian co-ordinates. Two types of move are allowed: namely translation and rotation. Each of these is described in terms of changes to the *co-ordinates* of points of an object. Objects can be moved around in the reference co-ordinate frame by using combinations of translations and rotations. The problem with a co-ordinate approach to the construction of spatial relations, however, is that spatial relations (understood in some algebraic form) are typically ancestral to co-ordinates i.e. one may know the desired spatial relation (e.g. extent of a line or face, relative angles of lines, contiguity of regions, intersections of lines, interpenetrations of regions, etc.) without necessarily anticipating the co-ordinate location of graphical parts in order that these relations can be satisfied. If a CAD system cannot support the expression of *intended* spatial relations, then it is inadequate in comparison to traditional methods, which, through their provision of a variety of implements (rulers, compasses, set-squares, t-squares, etc.), allow architects to construct such relations.

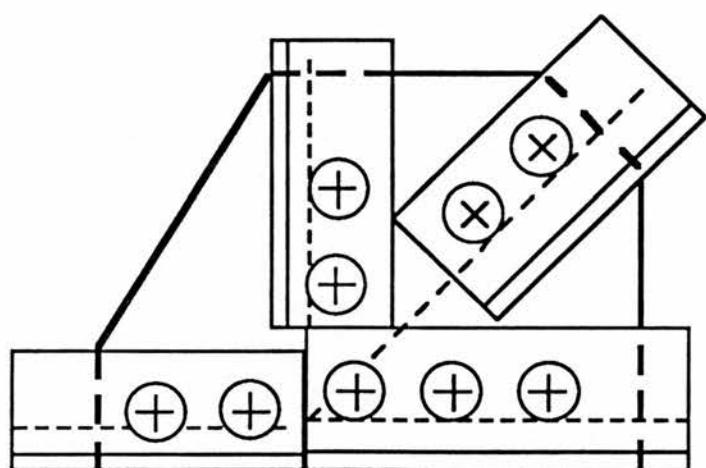
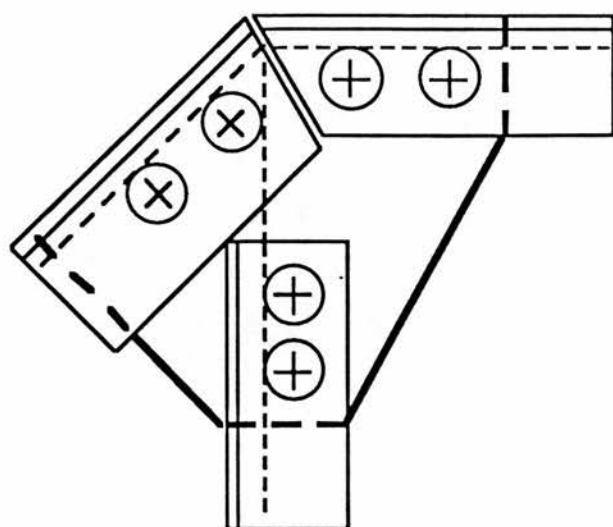


figure 6-22

6.6. Summary

By observing the deficiencies of conventional CAD systems, we can identify requirements for systems that support the kinds of operations used in architectural practice.

The most fundamental criticism of conventional CAD systems stems from the lack of recognition given to the importance of the role of the person using the system - the lack of teleological thinking that lies behind their design. Teleological concepts - such as functions, goals, purpose, choice, and free will - are an integral part of design in general. According to Checkland [Checkland, 1981],

"Man as designer is a teleological being, able to create means of enabling ends to be pursued, and to do so on the basis of conscious selection between alternatives."

Conventional CAD systems prescribe the alternatives available to users when depicting and transforming objects. There seems to be a serious mismatch in conventional CAD systems between depictions and their representations. Of course we can never expect to have a 'complete' representation of any depiction. We can always 'read in to' a picture more than is represented by any formal system. However, given a formal representation, we should expect that users can exercise their teleological instincts by expressing new relationships between represented parts, and to define new types with reference to known objects. The junction detail example can certainly be drawn using conventional CAD systems, but with a lack of correspondence between the drawing of the junction, and the intentions of the user producing the drawing.

Ideally, representations for drawing objects and operations should allow users to describe operations as specialised or as general as is required in any particular context. Specialisation, however, will always be limited by the primitives known to the system. Starting with known primitives, however, generalisation is in principle unbounded. Rather than have a system which provides relatively high-level system defined operations, it is preferable to aim for systems that provide only low-level operations upon low-level primitives, together with means for user-definition of more complex operations upon more complex objects.

Any representation environment should be implemented in a manner that remains accessible to users, so that users can declare what operations, to what degree, are to be applied to what collection of lines.

A CAD system should support the application of transformations selectively to parts of symbols, or to selected constituents of sub-pictures. Systems need to support arbitrary partial transformations. An implication is that the logical representation of drawing objects

and transformation operations must allow a user to *describe* an operation with reference to parts that are to move, in a context of parts that are to remain stationary.

CAD systems should support the expression of intended spatial relationships between parts of drawings. The facilities for the expression of such relationships should be in a form which subsumes, rather than is different from, or in opposition to, those functionalities available in traditional practice. The expression of relationships in algebraic form is one way of achieving this aim. Reliance upon co-ordinate descriptions and relative co-ordinate frames is not.

Part Two

An Environment for the Representation of Drawings

*"To-day we have naming of parts. Yesterday,
We had daily cleaning. And to-morrow morning,
We shall have what to do after firing. But to-day,
To-day we have naming of parts."*

[Henry Reed, *Lessons of the War: Naming of Parts*, 1946]

7. A Representation Environment

7.1. Introduction

This chapter will outline in a general way a representation scheme which will be used to represent design objects. It will be described in as general a way as possible since a multiplicity of implementations of the representation environment have emerged over a period of time, with each implementation itself subject to change. The representation environment was an essential component in the initial development of the drawing machine that will be described in the following chapter [Steel and Szalapaj, 1983]. It was subsequently developed in more detail [Bijl and Szalapaj, 1984; Szalapaj and Bijl, 1984].

Since then, the predominant concern of this thesis has been in the use of the representation scheme to support the representation of graphical objects having particular properties which allow user-controlled distortions of shapes - a powerful feature which I consider to be the central contribution of my thesis work [Szalapaj, 1987]. Parallel EdCAAD/SERC research has included a formalisation of the representation scheme [Krishnamurti, 1986]. The philosophical implications of how a formalism such as the representation scheme of the representation environment can assist or restrict users of any system that depends upon it, are explored in [Bijl, 1989]. Current EdCAAD/SERC research is concerned with the functional specification of an integrated CAD modelling system to support evaluative techniques that are used in building design [Tweed, 1986].

7.2. Primitives for a Representation Environment

7.2.1. Describing Things in terms of Parts and Properties

The quintessential element of the representation environment is the *description*. Things, including graphical objects, get represented by means of descriptions in the representation environment. A description is a set of *parts* each consisting of an *attribute-name*, *part-instance* pair, e.g.

My_house = {roof=Tiled, door=Brown, no_of_bedrooms=3}.

Attribute names are unique within a description. They can refer to logical parts or properties of any kind of thing. Part instances need not be unique and may occur in different parts of a description. A description may consist of any number of attribute-name, part-instance pairs.

7.2.2. Kinds, Slots, and Fillers

Using terminology that is common to certain object-oriented representation systems such as KRL [Bobrow and Winograd, 1977], we can interpret attribute-names in terms of two further basic constituents, *kinds* and *slots*. An attribute-name in general then consists of a kind name plus any number of slot names denoting a path to a part-instance of the kind e.g.:

$$K:s_1:s_2:s_3:.....:s_n$$

where:

- K is a kind name,
- s_1 is a slot name for some attribute of K,
- successive slot names ($s_2,s_3,.....,s_n$) are linked through kind names which are not included in attribute-name expressions.

The entity that an attribute-name refers to is called a *filler* (figure 7-1).

Kind names can refer to any kind of thing, concept, event, function or object. In this context, drawings and drawing primitives can be regarded as objects. Kinds refer to something that is to be described and which may form part of a description of something else. Filler names represent the parts of some thing that, as yet, have no further description, the bottom level of any current parts hierarchy. Slot names name parts of a description and slots are the means for attaching fillers to a kind. Operators are required for linking these elements in different ways to build up descriptions of whatever a designer may have in mind.

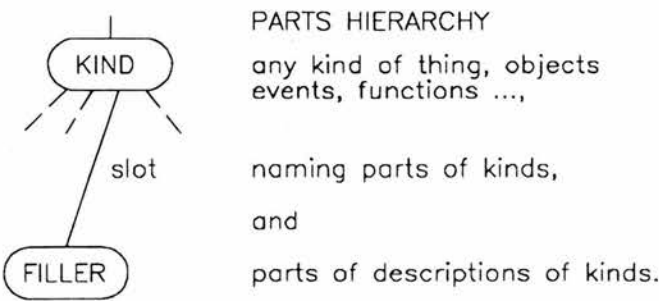
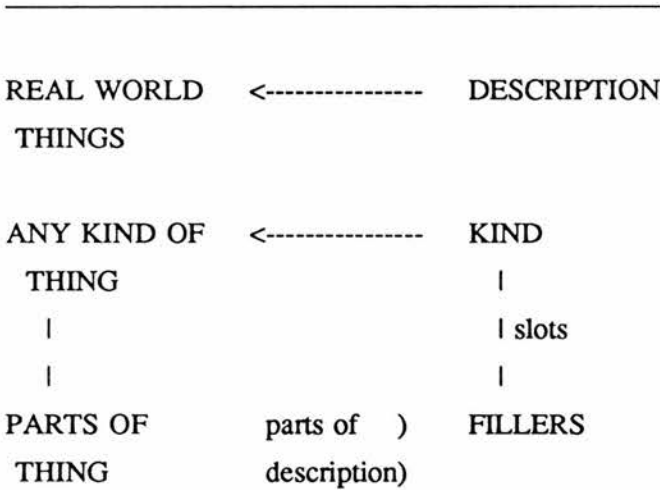


figure 7-1: Constituents of the Representation Environment

The notion of description that has been defined above employs a notion of parts that all belong to a single entity type, represented by kinds. The descriptive power of the representation environment relies wholly on relationships between part-instances. In other words, there is no reliance upon any form of entity typing that corresponds to typing of parts in some user domain.

Descriptions may arbitrarily have slots and fillers added, replaced and deleted. Further operators are needed in order to reveal names of existing kinds, fillers and slots, and the structures of existing descriptions. More complex operators, such as those for copying descriptions will be described in a later section.

PARALLEL STRUCTURES OF THINGS AND DESCRIPTIONS



The naming convention is to use upper case names (i.e. names beginning with upper case letters) for kinds and fillers, and lower case names for slots. Kind and filler names may be defined by the user or system-generated. Typically, names will be assigned to empty fillers when they become kinds and receive their own further descriptions, taking the name of the slot that names the part which is the filler. Kind names are automatically given instance identifiers which are incremented whenever their descriptions are changed (see § 7.4.). Slot names are always declared by the user. Operators are generally represented by symbols, such as <+ for add to, <= for replace with or <~ to make a variant of.

7.3. Basic Operations

Given the primitive constituents of the representation environment as described above, it is necessary to define precisely the operators that will work upon them to produce representation structures. In the simple case, things (represented by kinds) are described in terms of parts and properties (fillers of slots). Typically however, descriptions of things are relative to descriptions of other things ("John's coat is like Chris' but has more holes in it."). Relative descriptions can refer to specific individual properties ("The colour of Chris' door is the same as John's."). Such interdependence of description ought to be reflected in any representation environment. We will take the ability to be able to express such descriptions as a starting point for the definition of basic operations within the representation environment, and use diagrams to illustrate their effects.

We start from a world description of two kinds A and B, as in *figure 7-2*.

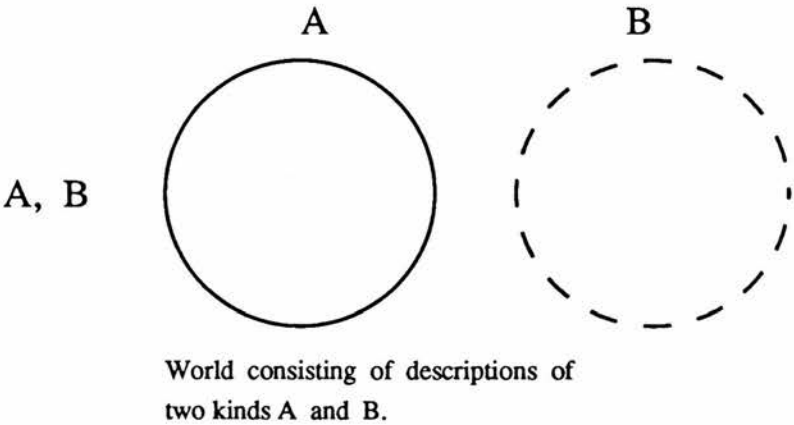


figure 7-2

One of the most basic operations that can be carried out is shown in *figure 7-3*.

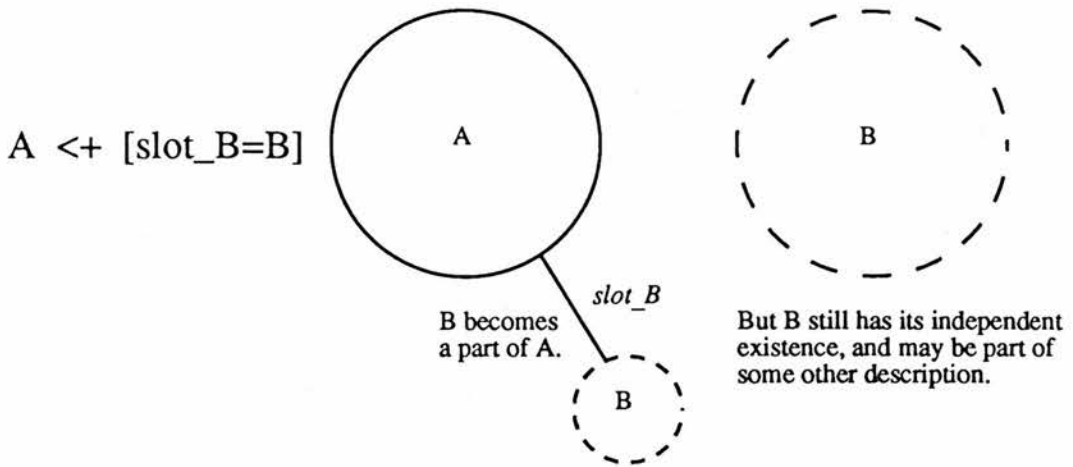


figure 7-3

<+ is the operation of adding a part to a description, and can be interpreted as meaning the addition of the right to the left. As well as adding an explicitly specified part as shown in the example (A <+ [slot_B=B]), this operator can be used to add unassigned attribute-names (A <+ [b]), or any combination of the two (e.g. A <+ [b, c=C, d]).

The example given above (adding an extant part to a kind) necessitates the explicit specification of a name for the part to be added. Using the <+ operator to update the description of a particular attribute-name rather than the description of a kind, can be achieved by constructing an attribute-name from the kind that is having a part added, followed by the name of the part that is being added to it. This attribute-name can then be used on the l.h.s. of the <+ operator, followed by a kind name on the r.h.s. (or some other attribute-name which refers to a kind). Thus A:slot_B <+ B is equivalent to A <+ [slot_B=B].

7.4. Instances and Inheritance: the operational semantics of <+

The operator <+ , has been referred to as the *add* operator. This is a reasonable description whenever the l.h.s. of the operator is a kind, and the r.h.s. consists of slots and fillers. The effect of the <+ operator is effectively to *link* the description of B to A via some explicitly specified slot name. It would be wasteful to have to copy over the whole of the description of B into A's description, and preferable therefore for the operator <+

to assert that A's B is a new *instance* of B, whose description is elsewhere. A's B continues to inherit B's description, but itself becomes part of A's description. The most common occurrence of instances arises when a new kind is being introduced into the representation environment, and is given the description of a kind which already has a description.

Instance relationships are implicit *inheritance* relations established within the representation environment. If an inheritance relationship was set up as a consequence of using the <+ operator to link the description of one kind to the description of another (e.g. A <+ [slot_B=B] as in the example above), then subsequent changes to sub-parts of any instance so created have the effect of taking precedence over corresponding inherited sub-parts. We can label this form of inheritance as *instance inheritance*.

With instance inheritance therefore, a kind appearing on the r.h.s. of the operator <+ becomes a new instance of its former self (initially receiving the same system-generated instance identifier). The new instance becomes the *child* of its *parent* instance. The child instance continues to inherit the description of its parent, but any change to any inherited slot will have the effect of superseding that inheritance. This means that if the filler to a slot of an instance is changed, then the changed filler takes precedence over the equivalent inherited filler. This remains true even when the parent's description changes.

To ensure that changes which are intended to be local to particular kinds are not propagated to other kinds, the representation environment operates a general rule which says that:

For any change to a part-instance that is accessed by an attribute-name beginning with a kind name K, all kind names along the path described by the attribute-name will have their *instance identifiers incremented by one*.

The effect of this is that the description of K and only K will include a changed description which retains the same attribute-name but receives a new part-instance. A simple illustration of this process is shown in *figure 7-4*.

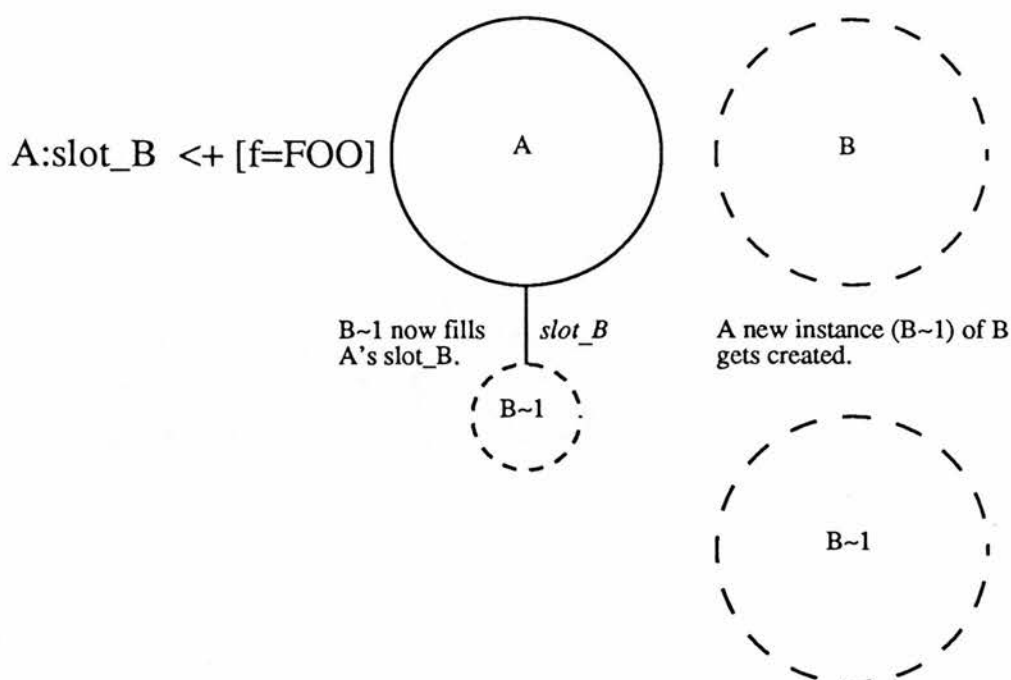


figure 7-4

Assuming that B was initially made a part of A by means of `A <+ [slot_B=B]`, suppose that a part of B with attribute name f, was now changed *through* A from some initial value to FOO. In order to ensure that this change is local to A, a new instance (B~1) of B needs to be created having the changed filler, and replacing the old instance.

Any part of a description of a kind can be passed along a chain of successive child instances, provided that each successive instance has been declared using the `<+` relation. However, nothing that happens locally to a child instance can affect descriptions of its parent instance or any successive grandparents.

Thus, if a particular instance of kind B is to be included in the description of kind A, and this instance already exists in another description, say C's q part, and it is desirable for B's q part to be modifiable through B itself (rather than by having to access C), then this can be achieved as shown in figure 7-5.

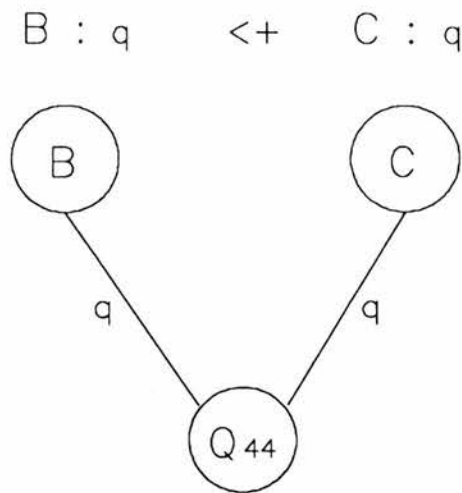


figure 7-5

If the system knew of C's q part as instance Q_{44} , then it would make B's q part Q_{44} as well. In the case of instance inheritance, it is preferable for a user not to have to know about such things as instance identifiers. The same effect as the above expression could therefore be achieved by:

$$B:q <+ Q.$$

This expression basically says that B's q is Q (i.e. that Q which exists as part of C's description), letting the representation environment take care of instance identifiers. B's and C's instances of Q would initially have the same description consisting of further slots and fillers, until B's q part gets changed. B would continue to see any subsequent changes to C's Q, provided those changes are not preceded by local changes to B's Q. *Instance inheritance* therefore is affected by the *subsequency* (and hence supersession) of local changes to fillers of inherited slots.

A further important observation to be made about instance inheritance is that any new instance can have only one parent instance, or a single linear chain of antecedents. We now look at a common way of viewing inheritance relationships, and show how this can be implemented in terms of instance inheritance.

7.5. The Expression of Variant Relationships

Instance inheritance is an implementational mechanism which supports the inheritance of properties between object descriptions. From the psychological point of view, it is often convenient to think of objects as being in some ways like other objects, or, in the old KRL terminology [Minsky, 1975], "seeing-as". For example, a "POT" can be seen as some sort of container, or as a shot in snooker which has the effect of pocketing a ball, or as some other thing. If "POT" had yet to be defined within the representation environment, but "CONTAINER" and "SHOT" had, then it should be possible to define "POT" in terms of both "CONTAINER" and "SHOT". In other words, "POT" is a variant of "CONTAINER" and of "SHOT", and can inherit properties from both.

Variant relationships therefore, are explicit (and possibly multiple) inheritance relationships which need to be expressed as such by a user of the representation environment. *Variant inheritance* allows for multiple views, and is therefore concerned with the *selection* of attribute-names from a potential multitude of ancestral bequeathers of identical slots. The representation environment has therefore, when supporting a change to the filler of a slot of a kind, to provide a mechanism for identifying where the slot is: whether it is local to the kind itself, or within the description of one of its ancestors. This is the case since the same slot name may exist as part of the local kind description or part of the descriptions of any of its ancestors.

There are several possible ways of providing such a mechanism, the least complicated of which is to do it in terms of the already existing instance inheritance mechanism. This can be done simply by creating new child instances of parent kinds, and then attaching these newly created part-instances via appropriate slot names to the required kind. In the "POT" example, with "POT" inheriting properties from both "CONTAINER" and "SHOT", suppose both "CONTAINER" and "SHOT" have a "unit_of_volume" slot filled by "CUBIC_CENTIMETRE" and "DECIBEL" respectively. What will the "unit_of_volume" slot of "POT" be filled by? There are two solutions, each dependent upon the context of description that is being looked at. It has already been shown how contexts can be described by means of attribute-names. For example, something like "POT:as_container:unit_of_volume" would refer to "CUBIC_CENTIMETRE", and "POT:as_shot:unit_of_volume" would refer to "DECIBEL".

In conclusion, therefore, we can see how the psychological mechanism of variance can be implemented in terms of instances.

7.6. Deletion

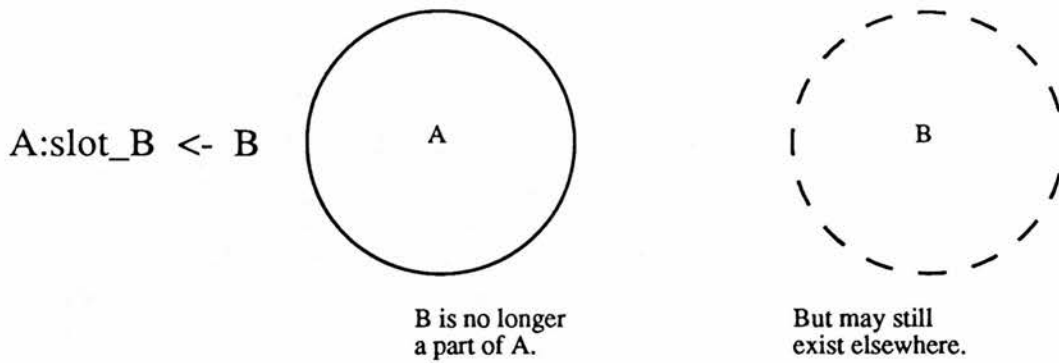


figure 7-6

Equally basic is the operation of deleting a part from a description, as shown in *figure 7-6*.

\leftarrow means delete the right from the left. At the simplest level, this is used to remove parts from descriptions, e.g. $A \leftarrow [c]$.

7.7. Breaking of Inheritance Relationships

Users have the power to break inheritance relationships.

\triangleleft - left is no longer to be an instance of right e.g. one kind is to discontinue inheriting changes to the description of another kind ($A \triangleleft B$).

When an inheritance is stopped, the currently inherited description is copied to the inheriting instance which can then continue to exist as an independent kind with a new kind label.

7.8. Copying a Description

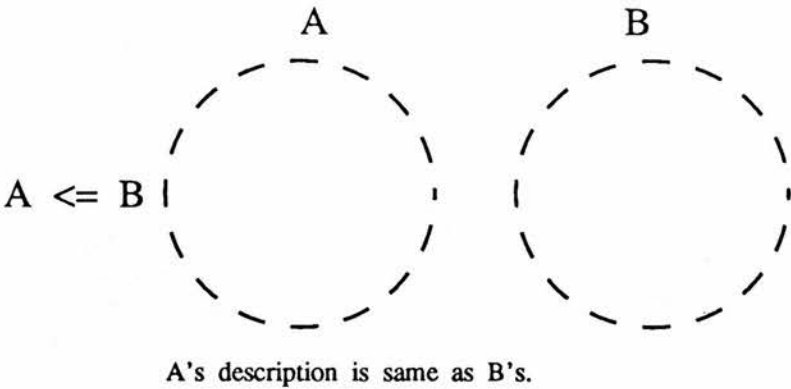


figure 7-7

Make the description of the kind on the left the same as that of the kind on the right. No inheritance takes place. A new kind (A) is created and B's description copied into it.

7.9. Merging Descriptions

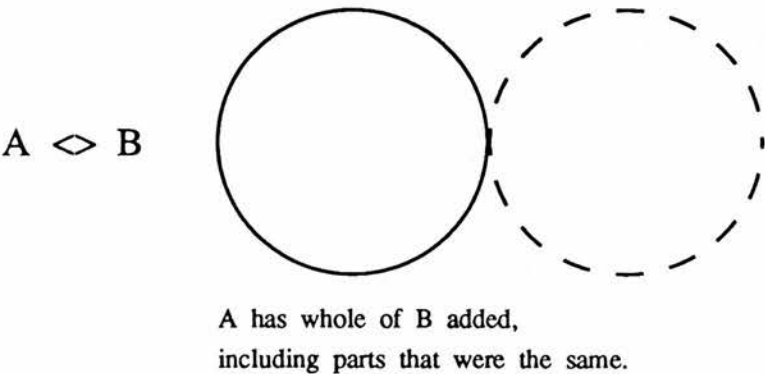


figure 7-8

The kind on the left becomes an aggregation of its own original description together

with the description of the kind on the right. In this case the whole of B's description becomes merged with that of A, and the kind B disappears. B may still be referenced within other descriptions, but the description of B will be empty (until redescribed). The difference between \diamond and $<+$ is that with the former, no new part-instance relationship is created. The usefulness of \diamond will become more obvious in chapter 11.

7.10. Renaming Descriptions

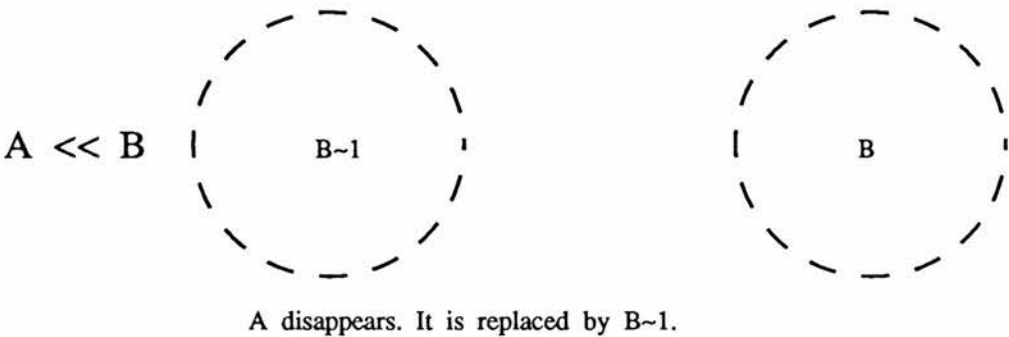


figure 7-9

$<<$ - within the context of the description on the left hand side, replace every occurrence of the named kind on the left with the kind name on the right. The changed kind becomes a new instance of the kind on the right. Again, the usefulness of $<<$ will become evident in chapter 11.

7.11. Indirection

Indirection is an instruction to *view* an instance of a part of another kind. It is a mechanism which will be used frequently in setting up relationships between graphical parts (see following chapter).

An indirection is invoked in the representation environment by the symbol $>>$, which establishes a relationship in which no change can be specified from the viewing kind - this is a *view only* slot. An indirection slot ensures that its kind sees only the viewed kind, and goes on seeing any changes that might subsequently be made to the viewed kind.

Returning to the earlier example, we now want kind B to see another kind and its description, C's Q, but without any intention to change the description of the viewed kind from B. This can be done as indicated in *figure 7-10*.

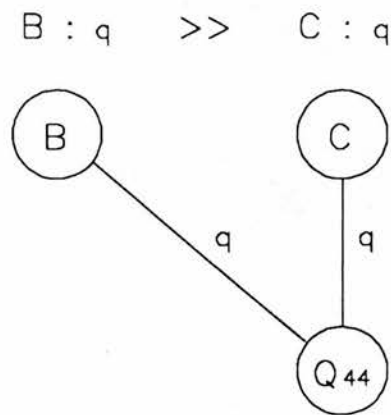


figure 7-10

The system's interpretation would be the same as before, but in this case any change to the description of Q can be made only from C. For example, in adding D to the description of Q, as in *figure 7-11*.

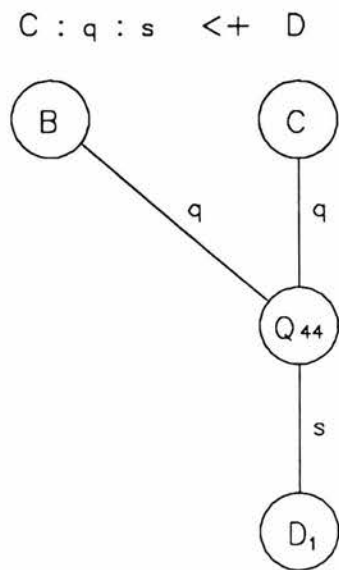


figure 7-11

Such a change would mean that the representation environment would then know of B:q:s, and that the filler of this slot was also D.

7.12. Naturalness of Representation

The most natural representation for any problem domain can never be determined objectively, but varies with different people, and even with different states of mind. Typically, the most natural and uniform representation for any system is always the most psychologically convenient one. According to Goodman [Goodman, 1966], the degree of naturalness does not affect the legitimacy of any system, but may affect efforts to determine what Goodman calls *the criterion of extensional isomorphism*. Basically, the criterion of extensional isomorphism is concerned with whether or not a reference to something in terms of explicitly named entities refers to the same thing as that referred to by an alternative description obtained by replacing these entities by their definitions. In other words, this criterion is satisfied when there is an equivalence between extensional and intensional objects.

Extensional isomorphism would only become problematic in our system if it allowed the expression of intensional statements which did not get evaluated. For example, suppose the user tries to set

Kitchen7:floor:covering <+ linoleum.

One can see that a user may want to construct such expressions, and one can envisage how such an expression might be preserved in its intensional form by storing it in the knowledge base in exactly the same way as it is expressed. Suppose Kitchen7 is then given a floor, Floor64 say, which is given a covering of linoleum. This is done by means of two separate evaluable expressions, an evaluable expression being one which consists of a l.h.s. which is a kind (rather than a part-name), and a r.h.s. which is a list of slot-filler pairs. The evaluation of an evaluable expression entails the assertion of facts about kinds, slots, and their fillers.

Kitchen7 <+ [floor=Floor64]. Floor64 <+ [covering=linoleum].

In this case, everything is fine, since the stored intensional description is isomorphic to the extensional objects that have been asserted as kinds, slots, and fillers. Suppose now, however, that Kitchen7 is given a new floor, Floor11, say, which happens to be already covered in tiles.

Kitchen7 <+ [floor=Floor11].

In this case, the old extensional references are deleted, and replaced by new ones referring to Floor11. The intensional description, however, remains, and contradicts the extant extensional objects. Furthermore, a system which considers all expressions to be intensional would be even more inconsistent since new intensional expressions are likely to contradict earlier ones.

The system as it stands then, makes no use of intensional expressions other than to resolve them with extensional references, which in turn can be overridden by new extensional facts. This same principle can be extended to those cases in which intensional expressions occur on the r.h.s. of expressions. In this case, at the time of stating the expression, intensional expressions may become part of the description of some kind, but only if the intensional expression is first evaluated, and found not to be inconsistent with extant descriptions. If subsequently, the extensional referents for these intensional expressions are changed or deleted, the intensional expressions still remain as part of the description of some kind, and will refer either to changed values/kinds, or, in the case of deletion, to nothing at all.

Additionally, a user of the system described so far can effect the extensional identity of kind names, by *binding* the same value to the two different names (more about this will be said in the following chapter). Similarly for the intensional identity between two kind names, which is achieved by binding each of the two kind names to some intermediate part-name which refers to some kind which has a value bound to it.

7.13. Summary

A list of representation environment operators together with their descriptions has been presented in this chapter. The aim was to provide abstract operators that are generally useful in creating descriptions. The set of operators evolved and changed over a period of time as it became clear that certain operators were redundant and could be replaced by other, more general ones. In some cases it was necessary to introduce new operators. It should be mentioned here that this is an ongoing process which was frozen for the purpose of this thesis. We are not claiming here therefore, that the set of operators presented here is a definitive one.

The personal view of the author is that the richness of any representation lies not only in the range of things that it can represent, but also in the scope that it itself offers for exploration of its limits. In this latter activity, even what appear to be only minor changes

to the system will often lead in unexpected directions, the result of which may well be a radically different representation. The strategy was to select a potential representation environment, and study its behaviour in coping with a range of problems, watching out for evidence pointing towards a more appropriate alternative structure.

We can think of this strategy as one of working with an evolving representational formalism. We have shown how, from the basic operations of adding something to a description and deleting something from a description, it becomes possible to represent things in a simple static way. Following on from this, we have added certain operations which can do additional useful things with this static structure. These include operations for copying, where the semantics of such an operation are clearly understood. Copying may or may not involve the creation of new object instances. Sometimes it is necessary to establish a variant relationship between kinds such that inheritance relationships can be expressed in a *controlled* way by a user.

It can be seen that, given a basic representation environment which supports the representation of parts hierarchies, and a limited number of conceptual ideas that work upon the hierarchic structures, such as adding to a description, looking at a description, copying a description; one finds that each of these permit a great amount of variation, but in limited directions, and on a limited number of themes. According to particular needs in particular domains, the representation environment can *evolve* appropriately to deal with them. The representation itself, has no concept of completeness, all descriptions being considered as partial. This is a vital feature for the support of design evolution.

8. A Syntax for the Generation of Logical Representations of Drawings

*"black black on white white
not vague darkness
black defined, black concentrate
crystal-pointed white*

*rigging, a line of land
nails, wires
lines alive, acts of language
constellations of black*

*counter to 'unlived life' (passing
repassing, drooping,
senselessly reviving)*

*energy, gay, terrible, rare,
a hope, man-made."*

[INK DRAWINGS by Denise Levertov (Here and Now)]

8.1. Introduction

Given the representation environment outlined in chapter 7, we now aim to use it to represent the general structure of drawings in the domain of architectural drawings. As drawing takes place, we would like representations of drawings to be created autogenously i.e. without explicitly stating what it is that is being drawn. To do this, a drawing machine has to be able to *communicate* with the representation environment in some way. It is proposed to do this by means of a grammar whose semantic rules refer to general knowledge about drawings. The semantic component, and in particular its topological content, will be given a fuller account in the following chapter. The syntactic rules are aimed at invoking corresponding representations in the representation environment autogenously as drawing takes place. The activity of drawing, therefore, invokes this grammar, which in turn invokes descriptions of *logical objects* (i.e. logical equipollents of drawing objects) in the representation environment.

This chapter contains a description of a proposed drawing machine which is capable of generating logical representations of 2-D straight line drawings, without restricting drawing objects to be polygonal. The general motivation for such a system is to support the types of drawing operations that are to be found in conventional architectural practice. A specification of the rules required to draw specific types of graphical objects is presented in the form of a context-free syntax with a compositional semantics.

8.2. Definitions

Here we make a distinction between:

1 *Depictions*

- arrangements of physical objects in a drawing space; the objects that depictions depict are referred to as *depicted objects*. The objects from which depictions are constructed are *drawing objects*.

2 *Representations*

- logical equipollents of drawing objects, replicated in a symbolic representation environment. The objects from which representations are constructed are *logical objects*. The concern of this chapter is the generation of both the depictions *and* the representations of 2-D line drawings, but not to transform the representations so constructed. What is being described in this chapter, therefore, are the operations essential to the *editing* of object descriptions. It will subsequently be possible to modify these representations in two distinct ways:

- a) firstly, by means of *transformations* which transform representations into new representations, preserving the constituents of descriptions but not necessarily their properties (chapter 10); and
- b) secondly, by means of *attachment* between two or more objects. Attachment takes place when two or more objects come together and implies the removal or addition of graphical constituents, having the effect of combining objects together or of decomposing an object into parts. This way of achieving changes to object descriptions will be described in chapter 11.¹

There is a clear demarcation between drawing machine knowledge and knowledge about corresponding logically determined objects in the representation environment. The former, although concerned with communicating graphical depictions to the representation environment, is also concerned with the production of graphical depictions on physical devices (and in this latter respect can be supported by available graphics production

¹ The two ways of modifying object descriptions suggested here, and presented in chapters 10 and 11 respectively, are presented in this way for historical reasons only. The transformations of chapter 10 will be shown to be distinct from the edit rules described in this chapter. It will become evident in chapter 11, however, that attachment operations have a great deal more in common with the types of edit rules described here. In fact, they can be said to be an extension of them.

systems, e.g. window management systems). The drawing machine developed in this thesis was supported by a graphics extension to C-Prolog known as Seelog [Pereira, 1982]. Seelog depends upon co-ordinate values for the production of lines and the delimitation of regions. The drawing machine, however, in order to carry out its communicative function, has also to have knowledge of the *structure* of graphical objects, together with a facility for the expression of *spatial relationships* between them. An example is the structure provided by construction lines from which points are defined, which in turn define segments.

Note that graphical output is only a temporal depiction of the representation environment's objects. As a user draws using the drawing machine, graphical objects with certain properties are instantiated. The drawing machine in turn invokes structures of logical objects in the representation environment. Consequently, the historical development of a drawing becomes significant in determining subsequent depictions. This will become more evident through looking at the examples both of this chapter, and those of chapters 10 and 11.

8.3. Graphical Primitives

Architectural drawings are essentially 2-D composite objects (*compositions*)² irrespective of whether they depict projections of 3-D objects. The pencil lines that appear in architectural drawings (see chapter 1) can be considered as construction lines (*conlines*) that are used to position the ink lines (*segments*) that appear in finished drawings. In a drawing machine, these could be presented as effectively infinite lines visible across a screen. The intersection of any two conlines will produce a construction point (*conpoint*) which may be used subsequently to delimit a segment. Any segment will lie between any two such conpoints.

Conlines can be viewed as a more general case of grid lines, in which the grid pattern is infinitely variable. Every instance of a conline is positioned according to some particular and possibly tentative anticipation of the final segments.

Conlines can either be horizontal, vertical, or inclined, reflecting the traditional way of producing drawings in a 2-D plane. Once a set of conlines has been constructed, it is then possible to traverse them, picking out segments to determine the desired shape of a drawing object.

2 Bracketed words denote the graphical primitives known to the drawing machine.

Segments and conlines are the primary graphical objects from which graphical representations of depicted objects can be constructed. It is primarily the boundaries of shapes (which can be represented using segments) that define them [Steel, S. & Szalapaj, P.J., 1983]. Segments can represent edges (which is the usual interpretation of segments), slices (non-physical boundaries of regions), and notional lines such as dimension lines and centre lines. The logical object representations in the representation environment, of graphical objects constructed from segments and conlines, can then constitute *partial descriptions* of depicted objects, in that they can be further described, both graphically and textually.

8.4. Properties of Graphical Primitives

8.4.1. Length and Angle Values

Any line, drawn in the X-Y plane, which is intended to represent an edge in 2-D space, has two design object dimensions associated with it; *length*, and *angle* relative to a bearing. Conlines, however, are effectively of infinite length, but do have the attribute of angle. In architectural drawing, it is typically conlines that are drawn first, and they therefore determine what angles segments shall have. Consequently, we associate angles with conlines and lengths with segments.

Upon completion of the segments representing parts of a new shape, the default design object dimensions of the lines contained in it are those that can be calculated from corresponding screen values, applying a current scale factor. Subsequently, it should then be possible to overwrite these values by intended design object dimensions.

8.4.2. Status Values

The fact that dimension values other than those of the screen co-ordinate system can be given to segments, implies that segments should have a status with respect to whether their lengths can/cannot be overwritten. Such overwriting may be done either explicitly by a user, or as a consequence of other drawing operations such as attachment (see chapter 11). This property of segments can be represented by an attribute of a segment which states whether or not the dimension is *fixed* (i.e. cannot be overwritten), or *unfixed* (i.e. can be overwritten). This status value might be conditioned by further intentions such as transformations. Status values can also be given to conlines with respect to their angles.

8.4.3. Symbolic Origin

A set of conlines has no logical origin. An origin is needed in order to establish the co-ordinates of conpoints in order that graphical output can be produced. If a logical origin were to exist, then the lengths of segments could be expressed in terms of the distances of their endpoints from the origin. However, it is preferable for lengths to be expressed in terms of distances between the endpoints of segments. If distance is expressed in this way, and a graphical object consists of a connected chain of segments, then it is necessary to have *directed* line segments or vectors. If direction is a property of segments, then any point can be taken as a temporary origin, and the other points located relative to it. The co-ordinate of a single origin would then be sufficient for a drawing machine to generate any drawing consisting of connected chains of lines. The origin is an attribute of a graphical object which can refer to some symbolic point. This point will need to have a co-ordinate value for purposes of the drawing machine, but not necessarily for the representation environment. Whenever an object needs to be drawn, the drawing machine can generate co-ordinates for all of the conpoints in the representation environment's description of that object, in order for a picture to be produced. The drawing machine can do this using the knowledge it has about lengths of segments. Providing a different co-ordinate value for the origin will effectively translate the object.

8.4.4. Scalability

Since the drawing machine will require drawing input within a co-ordinate system, and since upon completion of a drawing, screen values can provide inferable design object dimensions, it makes sense to scale the co-ordinate system in order to provide as meaningful values as possible until they need to be refined at some later date.

The drawing machine, therefore, should be capable of scaling its symbolic descriptions. Just as it is possible to express the co-ordinates of points relative to an origin, it should also be possible to scale an object relative to a particular segment length. Consequently graphical objects have a scale attribute which can refer to a segment whenever a scale operation takes place.

8.4.5. Rotatability

The drawing machine may also need to rotate its graphical objects, so a rotate attribute is a further part of their description. This will typically refer to a construction line whose angle is used as the reference angle at the time of rotation, such that all other angles

are expressed in terms of it; previous numeric values for angles can be expressed parametrically in terms of the rotated construction line.

8.4.6. Handing Represented by Sense

Having observed how a single origin would suffice to re-generate a shape consisting of a connected chain of lines, it follows that an additional property of segments is their *sense*, giving the direction of the segment with respect to its endpoints. A segment drawn from the origin along a construction line of angle 45 degrees, say, may be drawn either north-east or south-west. Sense is typically one of two values "+" and "-", depending on whether a directed line segment is on the clockwise or anti-clockwise side of a north-south axis (*figure 8-1*).

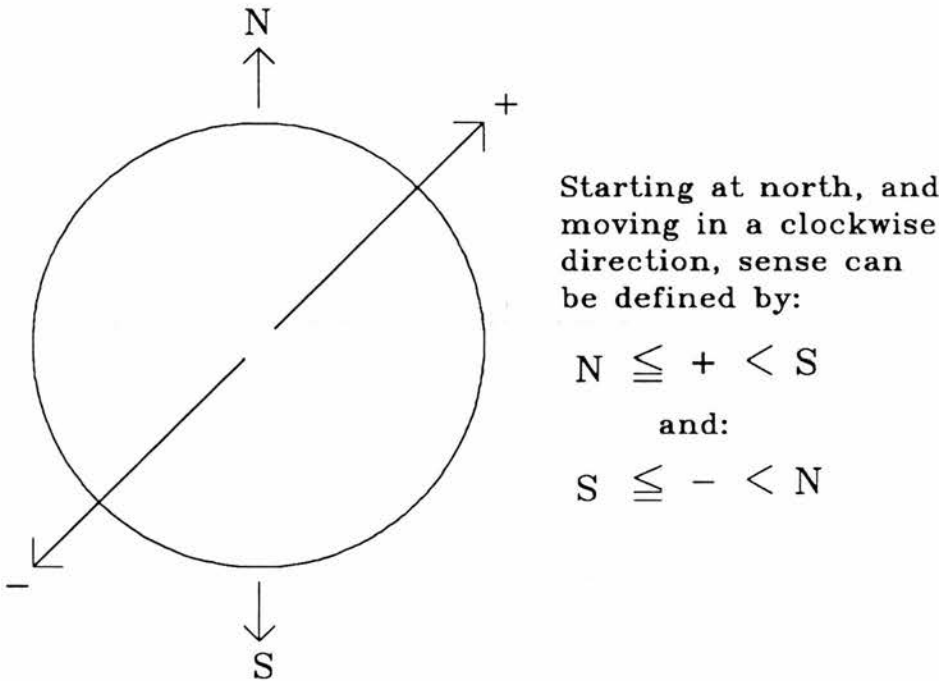


figure 8-1 : Sense

The role of sense is limited to the task of recording the direction in which a segment is drawn, from its first conpoint to its second conpoint. Sense can be affected by rotation that may result as a consequence of transformation (see chapter 10). Such changes have no direct effect on any sequencing of segments within a shape. 'Handing' of shapes is effected by binding angle values (chapter 10), and then specifying a reversal of sense for a particular segment.

8.5. Symbolic Dimensioning

We would like the dimensions of segments to be expressible in terms of the dimensions of other segments. Parameterised shapes require only a minimum number of dimensions to determine all the line dimensions for a given shape.

Part of the description of conlines and segments is an attribute which contains status information with respect to whether their dimensions can subsequently be overwritten (§ 8.4.2.). Typically, parameterisations are relationships that need to be maintained and hence the values of the fillers of their fixed/unfixed slots are usually "fixed". Wherever such parameterisation information exists in the description of an object, the appropriate dimensions need to be calculated in order to depict it. Where dimensions have been declared as unique numeric values they may have either "fixed" or "unfixed" status. Values obtained directly from the drawing machine will have "unfixed" status.

Apart from explicit user-defined parameterisations, there are also implicit system-generated ones which occur during the course of scale/translate/rotate/round operations, and which also result in symbolic expressions in place of actual numeric values for dimensions.

The advantage of symbolic dimensioning lies not so much in its use for stretching and shearing independent shapes, a feature which is used in some working CAD systems such as CADRAW [Hamilton and Scoins, 1980], and CAD systems such as CAM-X [Murray, 1982], but in its use in the composition and decomposition of objects, where it is applied to the conline/segment representations that we have already discussed.

8.6. A Context-Free Syntax For Generating Line Drawings

Using the above primitives, it is possible to devise a context-free syntax for generating representations of 2-D straight line drawings in the representation environment. In defining the base component of the syntax, object descriptions at certain levels will typically be decomposed in non-enumerable ways, i.e. there may be many segments to a shape, or construction lines (conlines) meeting at a construction point (conpoint). This is indicated by the recursive nature of the rules. The rules are presented in tables where the left hand side denotes a syntactic rule (or rules, depending on whether a particular graphical object needs to be defined recursively), and the right hand side a corresponding (brief) semantic description of this rule (or rules). In this section, the semantic constraints are expressed verbally, but will be treated more formally in the following chapter.

<p>Segment → Conpoint1 + Conpoint2 + <Length, Fixed, Bearer, Sense></p>	<p>A segment is effectively a line drawn between any two conpoints. A segment cannot exist between a conpoint and itself. A segment drawn <i>from</i> an existing conpoint, or <i>to</i> an existing conpoint, does <i>not</i> form a new conpoint at the point it is coming from or going to. Rather, the existing conpoint becomes a <i>common</i> conpoint. Consequently, the logical representation of such points changes to accommodate new attributes (e.g. additional conline) that the new segment might bring with it. Common points can be avoided by indicating the termination of a segment before the beginning of a new one, by introducing a new conpoint instead of starting (or terminating) at an existing one. The bearer is the conline that bears the segment.</p>
---	--

<p>Conpoint → Conline + Conline Conpoint → Conpoint + Conline</p>	<p>A conpoint is the intersection of two or more conlines. Given an existing conpoint, further conpoints can be generated by extending segments from the existing point along the conlines that bear them, then generating the conlines that belong to the segment's other end.</p>
---	---

(continued overleaf)

Conline \rightarrow <Angle, Fixed>	A conline is an infinitely long straight line, and therefore, for purposes of generation, can be described by the general equation for a straight line $ax + by + c = 0$.
--------------------------------------	--

Length \rightarrow { an element of the REALS }	Length is described by a numeric value for the distance between the two conpoints.
--	--

Fixed \rightarrow YES NO	The fixed statuses of angle and length values are indicated by truth values.
------------------------------	--

Sense \rightarrow + -	Sense is the direction of a segment between its conpoints.
---------------------------	--

Angle \rightarrow { an element of the REALS }	The angles of conlines are given in degrees.
---	--

From the above set of rules, it can be seen that an attempt at a definition of "shape" has been made. A shape is a collection of segments with an origin. We need to be able to distinguish between one shape and another in the representation of the syntactic structure for a composition, and this is achieved by means of attaching natural number subscripts to each shape instance. These names are system-generated, and allow textual, as well as graphical reference to shapes. Similarly, segments are so differentiated, as are conpoints, conlines, and compositions. Each segment has a unique length value and fixed status for its length, and hence these attributes do not need to be distinguished from one segment to another. Similarly for conline angle

values and fixed statuses for angles.

An important feature of the drawing machine is that any graphical object or its attribute (i.e. composition, shape, segment, conpoint, conline, length, fixed, sense, or angle) can be described by indirect reference to another graphical object or its properties. For example, for shapes with more than one segment, at least one conpoint must also belong to another segment's description, and is described by the mechanism of indirection (see following section) through the other segment. Under an attachment transformation (see chapter 11), a conpoint may take the name of a conpoint in another shape, or be described by indirection through another shape.

8.7. Indirection

An *indirection* is effectively a path through a parse tree generated by the syntactical rules, denoting that a description in one part of the tree takes the description of another part. It is typically of the form:

Constituent_name:constituent_1:.....:constituent_n

where "*Constituent_name*" is the name of a particular graphical object instance, and the constituents are of different types, with suffixes distinguishing between instances. Un-suffixed constituents will typically appear as the last elements in such indirection chains, since they cannot be further decomposed according to the base rules. Indirections of the latter type indicate terminals within the syntax.

The mechanism of indirection is relied upon heavily in order to be able to access parts of a description which are then transformed in some way. It also provides the means for testing structural descriptions w.r.t. their adequacy in having transformations applied to them (see chapter 10).

In generative grammar, the corresponding mechanism is to use labelled bracketings, whereby a structural description is represented by means of embedded lists in which each bracket has a corresponding label [Bresnan, 1976]. Such a schema requires that there be a complete structure for transformational rules to be applicable e.g. a sentence or a relative clause. In drawings, however, there is no concept of completeness, and transformations should therefore be defined upon only those parts of a description that are involved. It may turn out that the effect of a particular transformation upon a part is to propagate changes to all other parts with similar descriptions e.g. all segments within a picture. In such cases, we need a notation for

expressing such effects. Consider the shape shown in *figure 8-2*:

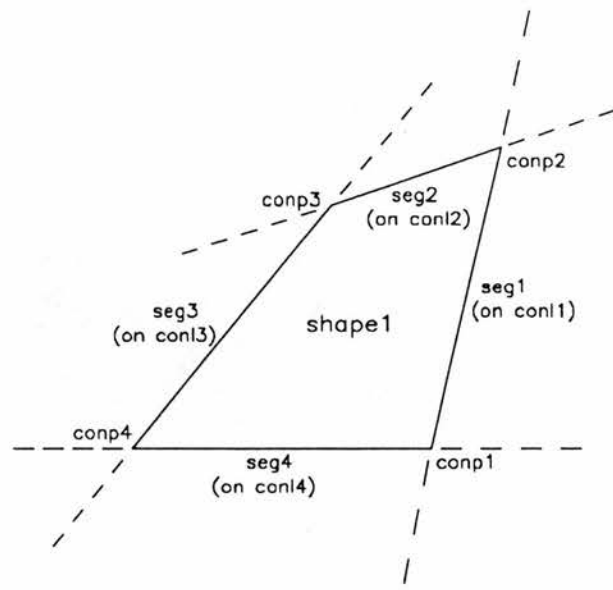


figure 8-2 : shape1

Using the base component rules, a parse tree for such a shape can be generated. The general form of descriptions of shapes and compositions is as shown in *figure 8-3*. The convention used to obtain the structure of *figure 8-3* is that the first endpoint of a segment is described within this segment's description. The second endpoint's description is referenced by means of indirection to another segment which contains the instance of the description. Similarly for conline descriptions within conpoints.

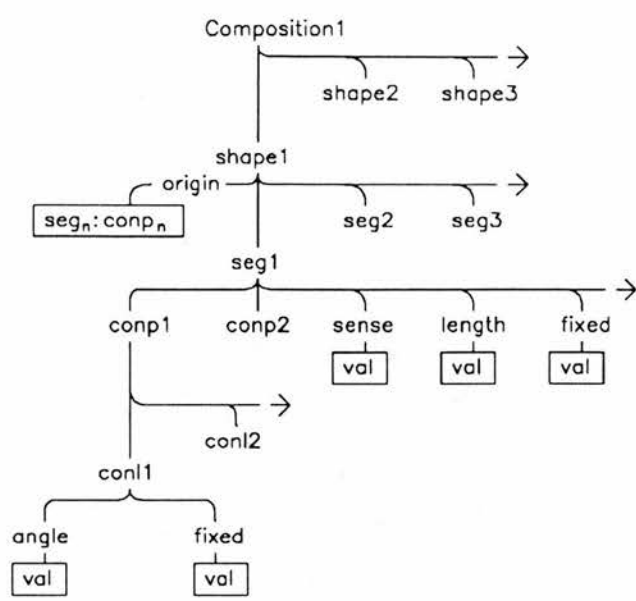


figure 8-3 : Logical description of shape

The rules defined above are unordered. The nature of the constituents referred to by the rules, however, imposes an ordering on the possible steps. Conlines need to be drawn before conpoints can occur, and conpoints must exist before segments can be placed between them. This implies a bottom-up parsing strategy for generation. In linguistics the parsing strategy is typically top-down, since there is a notion of a "complete" top-level object in the form of a sentence. A sentence consists of a linear sequence of words which are retraced once the sentence boundaries have been recognised. In drawing, however, there is no analogous notion of completeness, and what may start out initially as a group of lines, may end up as being the elevation of the west facade of a building, for example. In other words, one can go on defining the drawing at progressively more abstract levels, and hence bottom-up processing is the natural choice for generation of logical representations of drawings.

Regeneration is slightly different, however. To regenerate a graphical depiction of a logical representation that is already known about in the representation environment, one already knows what the top-level object is. This object may itself consist of sub-parts which are logically separate, each having a graphical depiction. Each depictable object should have its own symbolic origin. Starting from this origin, one can draw the conlines that pass through it, then recursively extend segments along these conlines to locate further conpoints and conlines. For each graphically separate object therefore, the process of regeneration is again a fundamentally bottom-up one, although the pre-

processing of logical objects that precedes it is top-down.

8.8. Logical Objects

Once a drawing has been produced using the rules and primitives described, the representation environment can associate it with a kind name forming part of the description of some logical object. A graphical object can be thought of as being an instanced logical object in that its structural description reflects properties of the drawing it is intended to represent. The representation environment's logical representation of a drawing will include automatically assigned slot and kind names, such as SEG-25 for a segment, C-15 for a conline, or P-37 for a conpoint, and these names can subsequently be linked with non-drawing parts of a description.

Typically, drawings are constructed by a combination of operations: making variants of already extant graphical objects; copying extant objects; adding further primitive objects; describing attachment relationships between graphical parts. Subsequently, further naming of abstract parts can occur in order to map them onto parts of the drawing. This may lead to renaming of system-generated logical objects to correspond to the newly created abstract parts to which they belong.

When a variant of a logical object is created, all the conlines, conpoints and segments of the variant are described systematically by means of relative naming. Thus the variant "SHAPE52" of a "SHAPE" with segments "S1", "S2", "S3", "S4", has parts that can be referred to as "SHAPE:seg1", "SHAPE:seg2", "SHAPE:seg3", "SHAPE:seg4" respectively at the time of creation. If subsequently the length property of "SHAPE52"'s "S3" changes, then the part name "SHAPE:seg3" within the context of SHAPE52, can no longer be referred to as such, and is instead replaced by a complete description of the new seg3 local to SHAPE52, and the new logical objects created within this new description are added to the database.

As well as being able to access objects graphically via the drawing machine, it is also possible to access them textually through their logical names. This is particularly useful in the case of dimensions (angles and lengths) which are typically updated by text input.

8.9. Drawing Machine/Representation Environment Interaction

It should be noted that the attributes that describe constituents of drawings, whose values are numeric values, truth values, +/- conditions, or instances of other constituents of drawings (referred to by means of indirections), map onto *slots* (parts/properties of objects) in the representation environment. Constituent names and instances map onto *kinds* and *fillers* respectively. For example, "segment1 of a particular shape" constitutes a part-name to a part-instance of segment. The base component of the syntax is defined with reference to constituents which map onto slots in the semantic representation. A syntactic tree consists of constituents with constituent names or values.

Conlines are represented as kinds with slots for angle values, their fillers referring to some angle datum (conventionally 0 for horizontal). Conpoints are described in terms of conlines, and become parts of a description in the representation environment only when they are used to delimit segments. The coordinate value of a point is initially taken from the space in which the drawing first occurs (the display screen coordinate, suitably translated into some real world value), but thereafter coordinate values are calculated by the representation environment from information it knows about a current (modified) description, from segment lengths and conline angles, related to some coordinate datum. Finished line segments have slots for their end points, length value and sense (direction from first point).

Conlines and finished line segments are the basic drawing machine primitives in the sense that these are used to build up further descriptions of objects. Point coordinates in the drawing machine are used only to give a single value to position a drawing shape, and to calculate initial values for segment lengths when segments first occur in a drawing; coordinates are not used to manipulate a description. Conpoints are inferred from conlines and segments, and are manipulated by the user saying things about conline angles and segment lengths.

An illustration of an instance of the representation of a graphical primitive in terms of its properties is given by the example in *figure 8-4*.

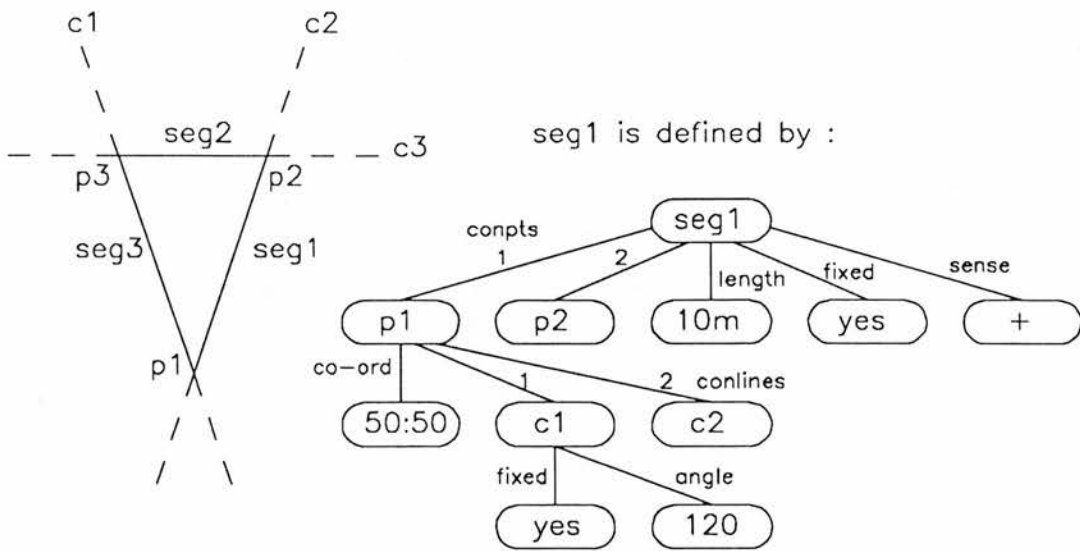


figure 8-4 : The description of a segment instance

Since we define shapes in terms of conlines and segments, it is not always obvious what is the set of conlines that belongs to a particular shape, and not to another, or whether there are unused conlines (those that do not define conpoints or are not the bearers of segments). Shapes can only be displayed with those conlines that are necessary to position all segments, and hence an object drawn initially as in *figure 8-5a* will be regenerated as in *figure 8-5b*.

A shape is adequately defined if only those conlines that describe conpoints of segments are presented, and the drawing machine will have no way of knowing how to place conlines that are not bearers or do not pass through conpoints; it will have no way of deducing distance information in order to place them. These conlines are not lost therefore, but just cannot be displayed as parts of a current shape.

Conpoints are determined by the intersection of conlines. However, several lines may pass through each other to produce many intersections. If new points were to be created whenever two conlines crossed, then there would be redundant conpoints.

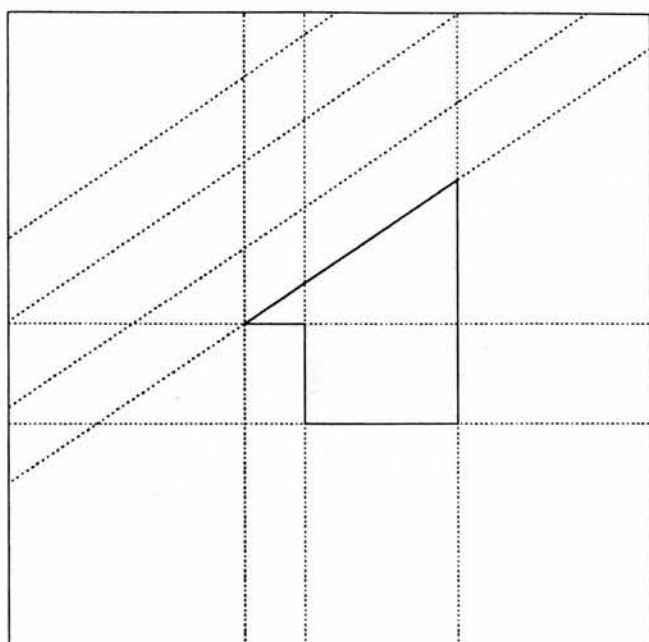


Figure 8-5a: An object drawn with conlines

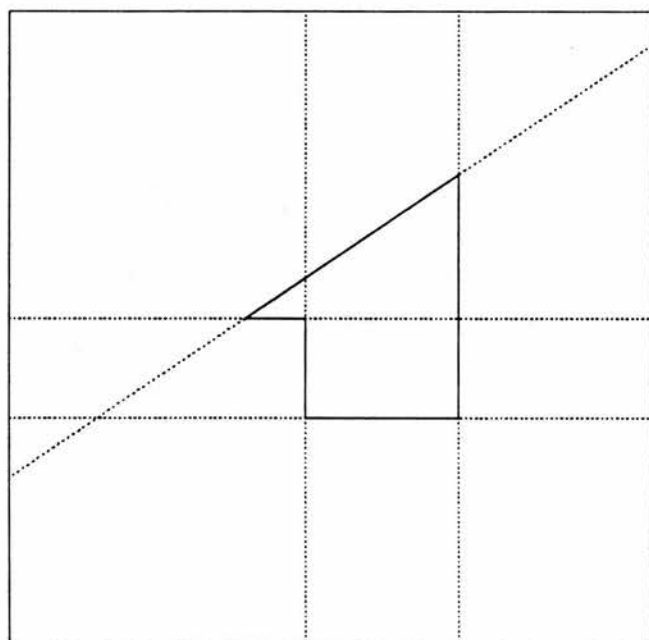


Figure 8-5b: A necessary and sufficient graphical presentation

Consequently, the drawing machine only generates conpoints when they are defined as being the end points of segments.

Further problems can arise when different shapes make use of the same conlines or conpoints. This becomes problematic when a dimension of one of the shapes later changes. Is the change propagated to the neighbouring shape, or does the latter retain its original dimensions? Often sharing of dimensions is necessary, but this is not always the case. The solution adopted is to let the user explicitly state such relationships using attachment operations (see chapter 11).

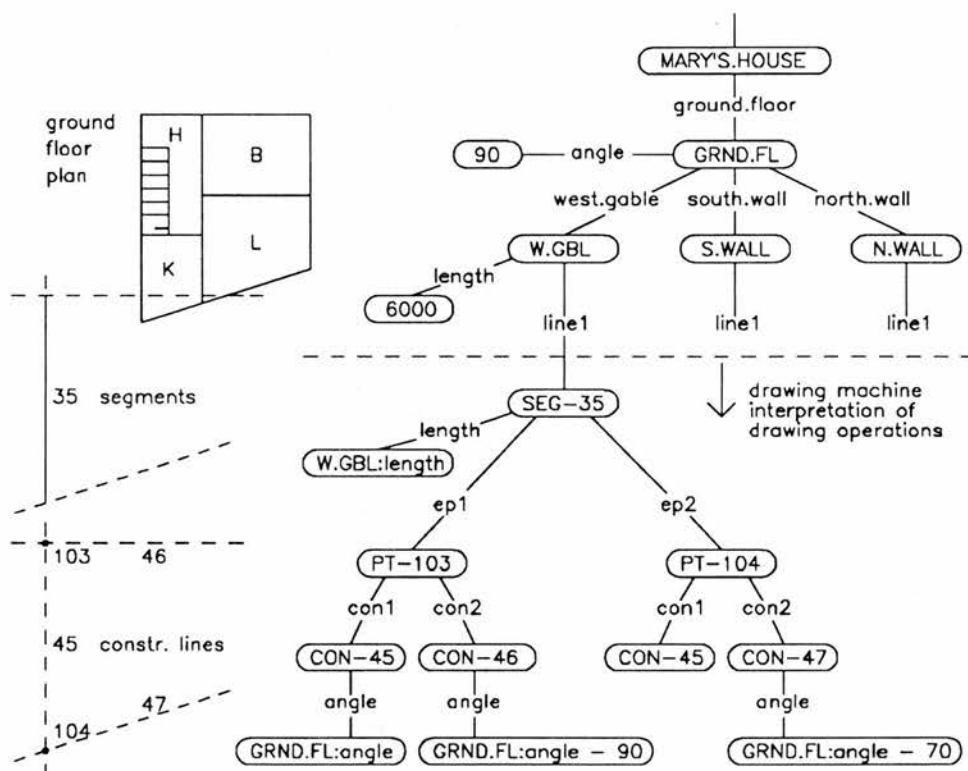


Figure 8-6: Logical Structure

Figure 8-6 illustrates the relationship between a description in the representation environment and a design drawing. For convenience, the representation is presented as a logic diagram of a *virtual* structure of parts of Mary's house. This is a virtual structure in the sense that it is not a physical structure stored in the computer. The structure is created by the system from information contained in the separate declarations of the user, in response to the user invoking the "virtual" operator, and the structure disappears when the user no longer wants to see it. The diagram exists outside the system and its purpose is solely to illustrate to the reader the relationships

between parts of a description inside the system.

The drawing is shown in the process of being constructed (from the bottom upwards), with conlines drawn first and then a segment added, on the way to a ground floor plan of a house. The segment forms part of the west gable wall description. The representation environment builds up a representation of the thing that the particular drawing describes, and the information it holds is sufficient to regenerate the drawing.

In this illustration the description below the dotted line is generated automatically by the drawing machine as drawing takes place. The sequence of logical expressions that are communicated by the drawing machine to the representation environment in the description of the ground floor west gable wall are as follows:

```
W.GBL    <+ [line1=SEG-35].
SEG-35   <+ [length=W.GBL:length,ep1=PT-103,ep2=PT-104].
PT-103   <+ [con1=CON-45,con2=CON-46].
PT-104   <+ [con1=CON-45,con2=CON-47].
CON-45   <+ [angle=GRND.FL:angle].
CON-46   <+ [angle=GRND.FL:angle-90].
CON-47   <+ [angle=GRND.FL:angle-70].
```

Note that angle values are established with reference to a user-declared angle value for the ground floor (so that if the completed ground floor part is later reused in different orientations, relationships between the drawing parts of the ground floor description will hold true), and the length value for segment 35 is overwritten by a user-declared length value for the west gable wall (affecting the position of lines that will appear on a subsequent regeneration of the drawing on the display screen).

8.10. Use of Fixing

Dimensions are typically overwritable by other dimensions, and problems arise when certain dimensions need to be kept fixed during the course of a design. When "fixed" dimensions clash with "unfixed" ones, you would expect the latter to get the former values. When two "unfixed" dimensions are to be matched, there should be some system convention which will resolve the assignment of dimension values. When basic shape transformations are being executed, (see chapter 10), the convention adopted is that the location drawing

provides the dimension for the component that is going fit into it. For a complicated shape which has both "fixed" and "unfixed" dimensions, it is not immediately obvious as to which of the "unfixed" dimensions should be calculated first, and the order of calculation will typically provide different final solutions. The best way of overcoming this problem is for the user to have control over such inferences. All dimension values are initially assigned "unfixed" status values. The user may subsequently wish to modify these statuses, as will the system when trying to satisfy structural conditions prior to executing transformations (see chapter 10).

8.11. Use of Binding to express Spatial Relationships

Values of fillers may be bound to the values of other fillers by means of indirection. The effect of an indirection as the filler of a slot is to tell the system to look for the value at the end of the path described in the indirection. Binding allows for the expression of relationships such as "parallel to", "orthogonal to". More on this will be said in chapters 10 and 11.

8.12. Summary

This chapter has presented the kind of knowledge needed by a drawing machine to communicate structures of drawing objects produced by a user, to a logical representation environment. This allows one to subsequently execute operations upon the logical representations of drawings, and to regenerate modified drawings. This knowledge can be divided into knowledge of the way in which the drawing machine works, in order that it can converse with a user in terms of drawings, and knowledge about drawings in general which includes topological knowledge (about which more will be said in the following chapter). The way in which the structure of graphical objects is understood by the representation environment has been described. This knowledge is reflected in the properties of the graphical primitives used in the drawing machine, and the ways in which operations on these primitives are interpreted by the system.

The strategy of regarding drawings as objects that describe other things, means that the representation environment can 'understand' drawings in the same manner that it understands any other things and therefore an integration between drawing and text descriptions which refer to design objects can be achieved.

9. Preserving Object Topology

"Let us consider some substance which is being used by man for his own purposes. However plastic it is to his designs, whatever transformations he makes it undergo, there remains something which he cannot alter, and which seems indeed to dictate the limits within which his transforming power over the substance shall extend."

[Sydney Herbert Mellone]

9.1. Introduction

The drawing machine described in the previous chapter, although capable of generating and regenerating graphical object descriptions, is inadequate, since although it can represent graphical objects with various topologies, it has no facility for the preservation of these topologies in the light of further changes to object descriptions. March and Steadman [March and Steadman, 1971] provide the following categorisation of commonly used mappings, or more precisely, transformations, found in architectural drawing:







mapping	invariant	position	length	angle and ratio	parallelism	cross-ratio	neighbourliness
identity		o	o	o	o	o	o
isometry			o	o	o	o	o
similarity				o	o	o	o
affinity					o	o	o
perspectivity						o	o
topology							o

figure 9-1: Transformations used in architectural drawing

From this table, it can be seen that different transformations arise as a consequence of a progressive relaxation of constraints (position, length, etc.). The least constrained, and therefore most general transformations are *topological*. Topological transformations preserve 'neighbourliness' (see below for definition). Further constraints upon such transformations can give rise to any of the other transformations mentioned.

A central objective of this thesis work was to devise a drawing machine which would allow for the expression of transformations upon drawings and drawing parts in a *controlled* way by a user of the drawing machine, as distinct from an approach in which transformations are already *prescribed* within the drawing environment. A natural way of achieving this goal, therefore, is to think of transformations as being essentially topological in nature (i.e. as unconstrained as possible). One should then be able to allow any user of the drawing machine to progressively *constrain* such transformations to achieve an intended result.

To proceed with this approach, we need to take a more formal look at how topological ideas relate to the domain of 2-D straight line drawings.

9.2. The Topology of 2-D Line Drawings

Topology is the study of those properties of geometrical objects which remain unchanged under continuous transformations of an object. A *continuous* transformation is one in which elements 'close together' to start with are 'close together' at the end of the cycle of transformation, such as bending or stretching.

The basic objects of topology are *topological spaces*. Intuitively, they can be thought of as geometrical figures. Mathematically they are *sets* endowed with some extra structure called a *topology* which allows one to set up an idea of continuity. A topological space is just a set of arbitrary elements in which a concept of continuity is defined.

The idea of continuity is based on the existence of relations, which may be defined as local or *neighbourhood* relations - it is precisely these relations that are preserved in a continuous mapping of one figure onto another. A topological space therefore, is a set in which certain subsets are defined and are associated to the elements of the space as their neighbourhoods. Usually, these subsets are open sets. Thus the neighbourhood of an element x in a topological space is any subset of the space that contains an open set that contains x . Depending upon which axioms the neighbourhoods satisfy, one distinguishes between different kinds of topological spaces.

In terms of the constituents of the syntax described in the previous chapter, we define a *simplicial complex* (or *complex*) as consisting of a collection V of conpoints together with a prescribed collection S of nonempty subsets of V , called *simplexes*, such that every conpoint is a simplex, and every non-empty subset of a simplex is also a simplex. The *dimension* of a simplex is one less than the number of conpoints in it; that of a complex is the maximum dimension of any simplex in it. In these terms, a 2-D line drawing is a complex of dimension 1 or 0. A segment is a 1-dimensional simplex. A complex is 0-dimensional if and only if it consists of a collection of conpoints, but no segments or other higher dimensional simplexes. Aside from these "totally disconnected" drawings, every drawing is a 1-dimensional complex.

We can define the *neighbourhood of a conpoint of a 2-D line drawing* to be the set union of all simplexes of an arbitrary simplicial decomposition of the drawing which contain the given conpoint in their interior or on their boundary. Thus neighbourhood is defined in terms of the set of segments that meet at a conpoint. This definition will therefore use closed sets rather than open ones (figure 9-2).

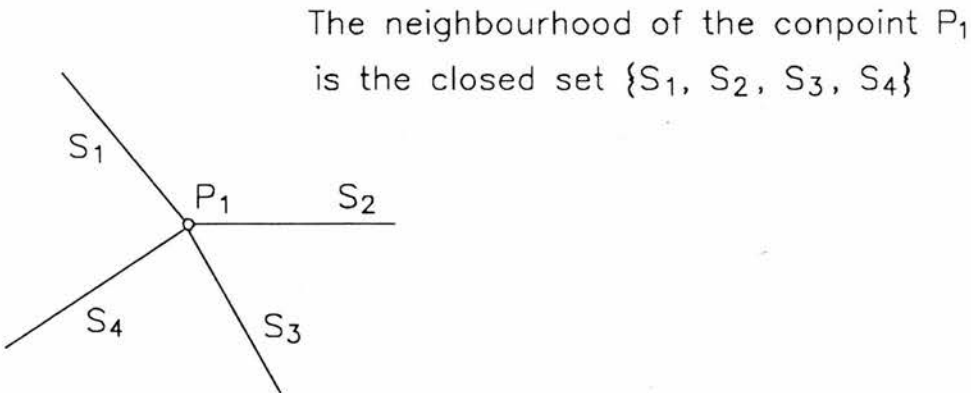


figure 9-2: The neighbourhood of a conpoint

Using this notion of neighbourhood, we can define *continuity* for 2-D line drawings. A mapping f of the topological space of 2-D line drawings X onto a (proper or improper) subset of this topological space Y is called *continuous* at the conpoint x ($x \in X$), if for every neighbourhood $U(y)$ of the conpoint $y = f(x)$ ($y \in Y$) one can find a neighbourhood $U(x)$ of x such that all segments of $U(x)$ are mapped into segments of $U(y)$ by means of f . If f is continuous at every conpoint of X , it is called continuous in X [Alexandroff, 1961].

We are primarily interested in those 2-D line drawings that possess the topological property of *connectivity*. Intuitively, a connected space is a topological space which

consists of a single piece. A connected 2-D line drawing is one which is not composed of several disjoint 2-D line sub-drawings. Connectivity is perhaps the simplest, yet most important property a topological space may have. Any continuous mapping will always take one connected space into another. Each transformation, therefore, will have the property of preserving connectivity. More precisely, a *connected space* is a topological space X which cannot be represented as the union of two disjoint non-empty sets. In analytic topology, these sets are open sets. Since we are effectively using algebraic topology, closed sets will do [Alexandroff, *op. cit.*].

Two topological spaces are *topologically equivalent* if we can pass from one to the other in a continuous way, and also come back in a continuous way. In terms of set theory, topological equivalence between two topological spaces A and B demands a function $f: A \rightarrow B$ such that f is a continuous bijection, and its inverse function is also continuous.

The following chapter will attempt to outline the way in which the properties of graphical objects presented in chapter 8 might be exploited in order to be able to invoke topological transformations which may be additionally constrained in a controlled way by a user of the drawing machine.

9.3. Summary

We have defined in this chapter those properties of drawings that will be preserved in transformations. The connectivity of a 2-D line drawing will affect the outcomes of transformations applied to local neighbourhoods. This feature can be used within a drawing machine to reflect intended relationships between parts of drawings.

10. Transformations

10.1. Introduction

Given the need to be able to carry out topological transformations upon shapes using the primitives of the drawing environment (conlines, conpoints, and segments), we need to find a way in which the properties of primitives can be exploited in order to effect such transformations. Firstly, let us recall what these primitive properties are.

In the case of conlines, they have the property of angle, expressed in degrees. Angles of conlines can be related to or *bound* to angles of other conlines by means of indirection. Angle values are by default obtained from screen values. In addition, conlines have status values to denote whether their angle values can be overwritten. If they can, then they are said to be *unfixed*, if not, they are known to be *fixed*. A conline is always part of at least one conpoint.

A conpoint has at least 2 sub-parts which are distinct conlines. A conpoint is part of at least one segment.

Segments have the property of length expressed in length units. Lengths can be *bound* to lengths of other segments by means of indirection. Length values are by default obtained from screen values. Just as with conlines, segments also have status values to denote whether their length values are *fixed* or *unfixed*.

Each segment has two endpoints (conpoints), a sense and a bearer. A segment is always part of at least one connected chain of segments (a shape). The origin of a shape can be any conpoint of any segment within it.

Transformations therefore, can be defined upon those and only those parts described above. Changes to object representations, with the intention of preserving their topology, are effected by qualifying any parts that should take part in such changes.

10.2. How Transformations Work

Changing the properties of the graphical primitives described above is the essence of graphical transformation. Changes to segment lengths and conline angles provide us with basic transformations. The effect of such changes on the representation environment is to establish replacement fillers to slots of kinds. Such destructive updates are achieved using a sequence of two expressions, the first one containing the logical operator '<-', followed by a second expression containing the logical operator '<+'.

The transformations of interest are those in which the shape before the transformation is topologically equivalent to the shape after the transformation (irrespective of what happens in between - see §10.6.). If a particular shape is not topologically equivalent to a known shape object, then this could have been produced only by an *edit* rather than a *transformation* of the object.

If the end product of a transformation is a topologically equivalent shape, then any change to a property value will unavoidably have knock-on effects upon adjacent (connected) parts. An obvious transformation of this nature is one in which a conpoint of a shape is moved to a new location. This transformation could be invoked in a variety of possible ways, each producing a different outcome:

- (i) Firstly, when rotating the angle of a conline about a conpoint. Besides changing the angle value of the conline, such a change may affect the length of any segment lying along this conline, as well as the lengths of segments which have endpoints that meet this conline.
- (ii) Secondly, when changing the length of a segment. In this case, the direction in which the change takes place has to be indicated. The endpoint that moves may pull with it any other segments that meet at this conpoint, thus affecting both their lengths and the angles of their conlines.
- (iii) By moving the conpoint itself. The lengths of all segments that meet at this conpoint are liable to change, as are their conline angles.

Other candidates for transformations include moving a segment to a new location. We can envisage such a transformation occurring when the endpoints of a segment are mapped onto two new endpoints. Furthermore, such a transformation can be directly extended to a set of connected lines that get mapped onto some other set of conpoints.

Note that each of the examples cited refers to *local* changes to some part of a shape, which may in turn cause changes to adjacent parts. For such knock-on effects to occur, we

can expect that transformations can be compounded, i.e. certain shapes can be explained only by assuming that they have undergone several transformations in sequence. As far as a user of the drawing machine is concerned, the general strategy will be to indicate a change to be made, and then to indicate the segment which is to receive the change parameters. The drawing machine should then receive this information and automatically apply change operators to the shape name and execute the consequential changes to the shape description. Different change operations can be used in combination, to effect required changes to one or more shapes.

For a transformational rule to be applicable to some part of a shape, it is necessary to define local conditions upon the part structure, under which the transformation could conceivably take place. We can express structural conditions in terms of indirections, which are implicit conjunctions of basic predicates constructed from categories of the syntax. A transformation T is an ordered pair (C,M) , where C is a structural condition, and M is a transformational mapping [Bresnan, 1976], i.e.

$$T = (C,M)$$

For a transformation upon a structure to be possible, the condition C must be true of that structure, and the mapping M must be defined upon it. A transformational mapping is composed from a set of elementary operations, each of which can be defined in terms of the representation environment's own change operations. A transformational mapping

$$M = \{o_1, \dots, o_n\} \quad ,$$

consists of n elementary operations. Suppose a particular ox ($1 \leq x \leq n$) is defined with reference to a kind K , having an attribute $R1$, which may or may not have a current value. Then ox can take one of the following forms:

$\{Ti, R1, V1\}$	- identity	(implemented as a meta-logical comparison between the evaluation of $K:R1$ and the evaluation of $V1$)
$\{Ta, R1, V1\}$	- adjunction	(implemented as $K:R1 <+ V1$.)
$\{Td, R1, V1\}$	- deletion	(implemented as $K:R1 <- V1$.)
$\{Ts, R1, V1, V2\}$	- substitution	(implemented as $K:R1 <- V1$. followed by $K:R1 <+ V2$.)

where V2 is a new value that replaces the previous value V1.

10.3. Effect of Basic Transformations on a Simply Connected Shape

In this section, we will illustrate the effect of applying basic transformations (of which there are three: rotate online, change length, translate point) upon a simply connected shape (a rectangle). We will assume for the purposes of this example that each segment meets each segment adjacent to it at a *common* point. The definition of both structural conditions and transformational mappings will be given here in terms of indirections. By convention, each segment description contains an explicit description of one of its endpoints, which in turn contains an explicit description of one of its conlines (typically, the 'bearer'). The other endpoint is indirected to its description in the context of another segment. Any other conline is indirected to its description in the context of another segment's conpoint.

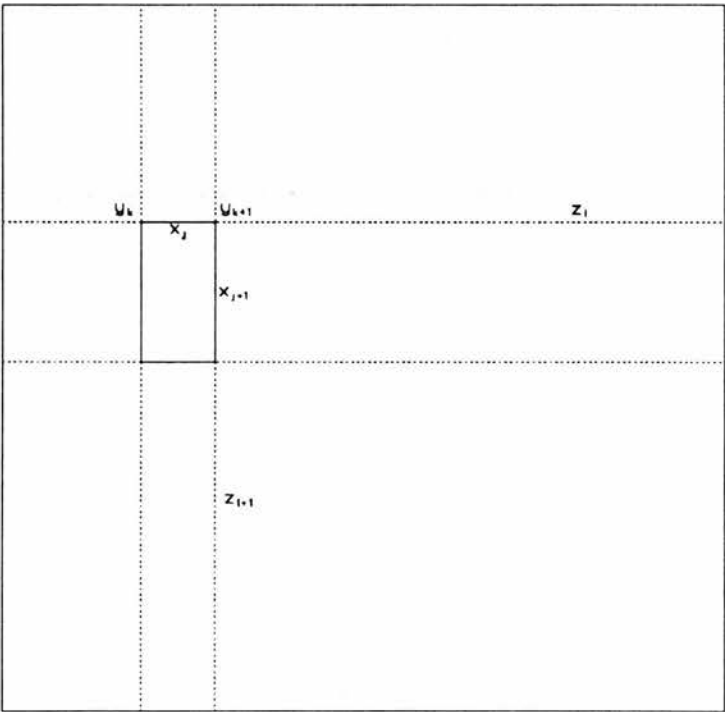


figure 10-1: An initial shape

The series of examples in this section is intended to illustrate how parts in the logical representation of shapes can be accessed to support the execution of transformations.

10.3.1. Changing a Conline Angle

Suppose we want to rotate the conline z_l about the conpoint y_k by some value specified in degrees. Local structural conditions need to be identified for the connectivity of the shape to be preserved:

O1, consisting of the following element, requires the conline angle in question to be variable:

{Ti, Shape_a: seg_x_j: conpoint_y_k: conline_z_l: fixed, NO}

O2 and O3 identify the status of the length of the segment lying along the conline, and similarly for the following segment. Suppose the values returned are:

{Ti, Shape_a: seg_x_j: fixed, NO}

{Ti, Shape_a: seg_x_{j+l}: fixed, NO}

Given these conditions, the transformation can be executed by means of the following three mappings:

Firstly, O4 which changes the angle of the conline:

{Ts, Shape_a: seg_x_j: conpoint_y_k: conline_z_l: angle, Angle}

Next, O5 and O6 have the effect of re-evaluating segment lengths:

{Ts, Shape_a: seg_x_j: length, L1}

{Ts, Shape_a: seg_x_{j+l}: length, L2}

A possible end product of such a transformation is shown in *figure 10-2*.

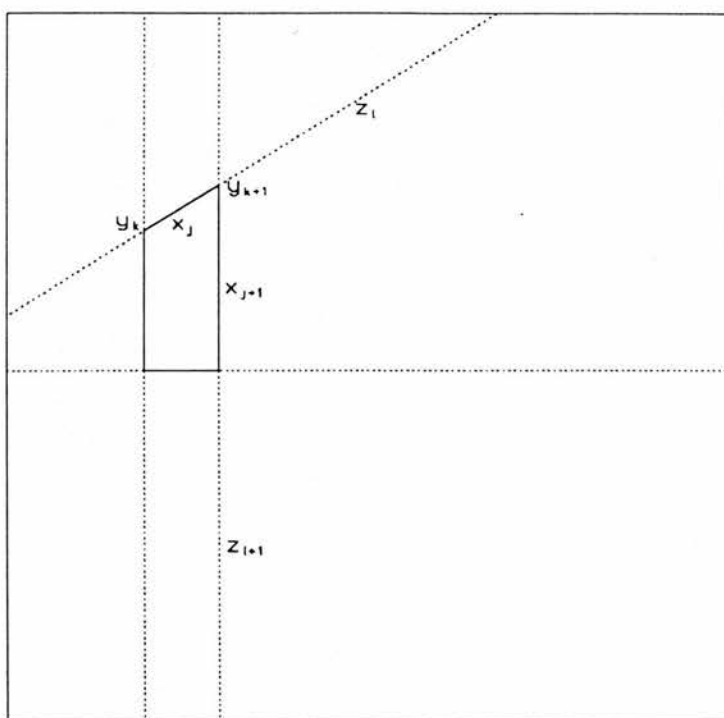


figure 10-2: The shape after rotation of conline z_l about y_k

10.3.2. Changing a Segment Length

Suppose instead, starting from the initial shape shown in *figure 10-1*, we want to increase the length of the segment x_j by an amount specified in the direction y_k to y_{k+1} . The structural conditions are:

First, O1 requiring the segment length in question to be variable:

{Ti, Shape_a: seg_ x_j : fixed, NO}

Next, O2 and O3 identify the status of the length of the adjoining segment, and similarly for the angle of the conline upon which this segment lies. Note here that fixed statuses are typically one of two values. The choice in these examples has intentionally been to assume that values are variable, and to illustrate the kinds of outcome that will arise in this case. The way in which fixed statuses are handled will be described in § 10.6. Suppose that in this example, the values returned are as follows:

{Ti, Shape_a: seg_ x_{j+1} : fixed, NO}

{Ti, Shape_a: seg_ x_{j+1} : conpoint_ y_{k+1} : conline_ z_{l+1} : fixed, NO}

Again, given these three conditions, the following three mappings can be executed:

First, O4 consisting of a single mapping which changes the segment length:

$\{Ts, Shape_a: seg_x_j: length, L1\}$

O5 and O6 have the effect of re-evaluating the length of the adjoining segment along with the angle of its conline:

$\{Ts, Shape_a: seg_x_{j+1}: length, L2\}$

$\{Ts, Shape_a: seg_x_{j+1}: conpoint_y_{k+1}: conline_z_{l+1}: angle, Angle\}$

A possible result is as shown in figure 10-3.

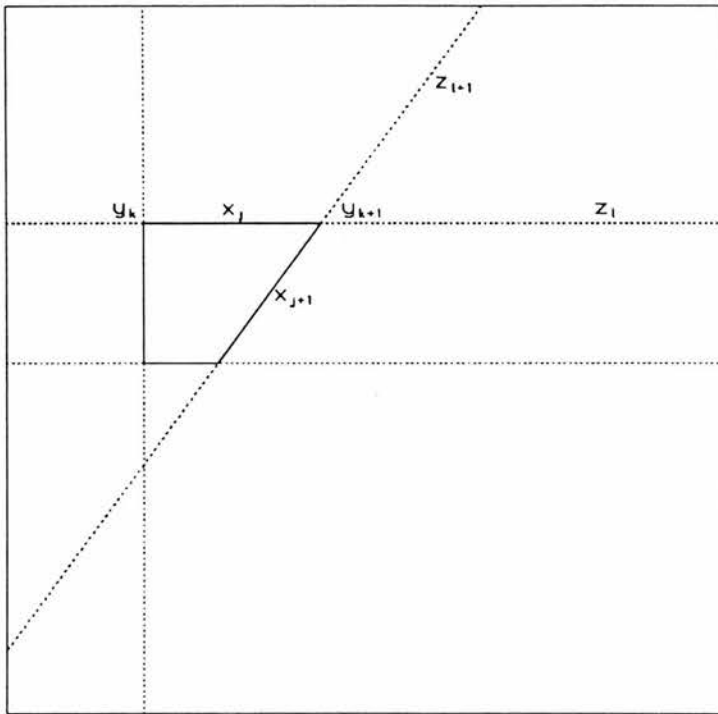


figure 10-3: The shape after increasing length of segment x_j

10.3.3. Translating a Conpoint

From the initial shape in figure 10-1, suppose the conpoint y_{k+1} needs to be moved to a new location. The structural conditions are:

Firstly, O1 and O2 consisting of two elements, which identify the statuses of the following two conline angles:

$\{Ti, Shape_a: seg_x_j: conpoint_y_k: conline_z_l: fixed, NO\}$
 $\{Ti, Shape_a: seg_x_{j+l}: conpoint_y_{k+l}: conline_z_{l+l}: fixed, NO\}$

O3 and O4 identify the statuses of the lengths of the segments lying along the conlines:

$\{Ti, Shape_a: seg_x_j: fixed, NO\}$
 $\{Ti, Shape_a: seg_x_{j+l}: fixed, NO\}$

These conditions having been satisfied, now allow for the following transformation to be carried out.

O5 and O6 change the angle of the conlines:

$\{Ts, Shape_a: seg_x_j: conpoint_y_k: conline_z_l: angle, Angle1\}$
 $\{Ts, Shape_a: seg_x_{j+l}: conpoint_y_{k+l}: conline_z_{l+l}: angle, Angle2\}$

O7 and O8 have the effect of re-evaluating segment lengths:

$\{Ts, Shape_a: seg_x_j: length, L1\}$
 $\{Ts, Shape_a: seg_x_{j+l}: length, L2\}$

This gives us *figure 10-4*.

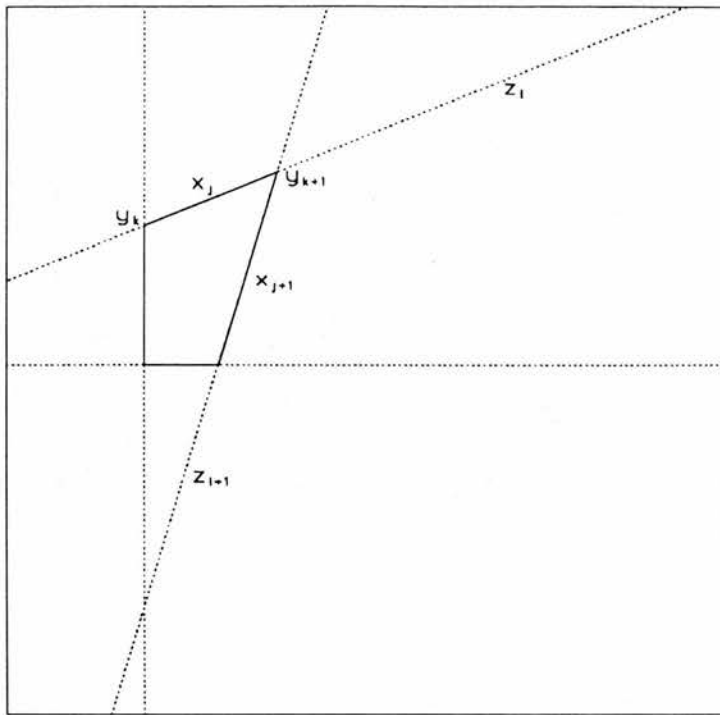


figure 10-4: The shape after moving the point y_{k+1}

10.4. Generalised Transformations

The previous example illustrates the kind of results we wish to achieve with transformations applied locally to parts of shapes. Changes in properties of primitives transform shapes whilst preserving their topology. We can generalise these three basic transformations to any shape by *recursive compilation* of local structural conditions, together with *recursive application* of transformational mappings. The recursion is with reference to any graphical parts that are *connected* to the part that is being transformed.

Starting at a conpoint (the translate conpoint transformation), the segment having this conpoint as an endpoint can be determined, and the properties (structural conditions) of this segment can be established. Similarly for other segments sharing this common conpoint. The application of a transformation proceeds by first applying a transformation to a chosen segment, and then proceeding to the next segment. Similarly for the other transformations.

In the case of changing a conline angle, the connectivity is that between segments that meet other segments having this conline as their bearer (other than those segments that meet the conline at its conpoint of rotation). This allows for the transformation of a figure

such as 10-5a into 10-5b.

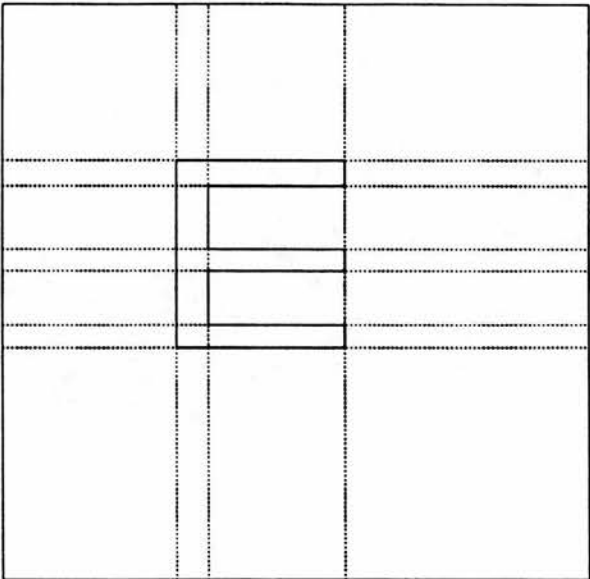


figure 10-5a

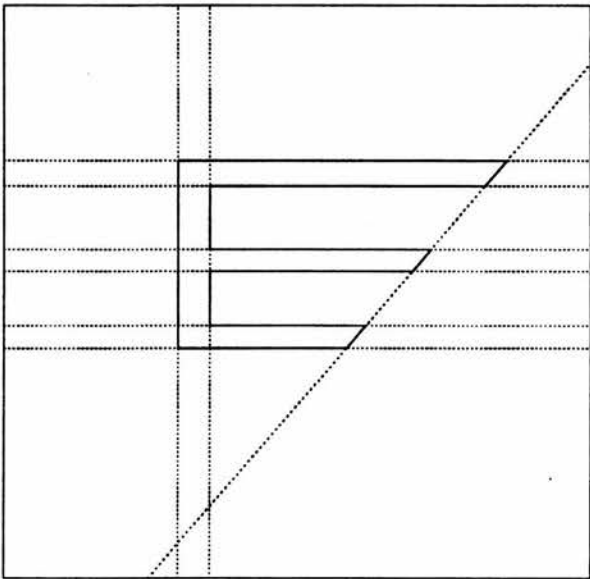


figure 10-5b

In the case of changing a segment length, the connectivity is between the segment

whose length is being changed, and any segment that meets this segment (excluding those that meet it at its stationary endpoint).

In the case of translating a conpoint, the connectivity is between all segments that meet at this conpoint, together with any other segments that meet these segments (other than at the unmoved endpoint). 'Meeting' depends upon forms of attachment which will be described in the following chapter.

A further important point to be made at this juncture, is that any of the basic transformations can be applied equally well to 'loose' segments i.e. segments unconnected to any other segments. This will be of importance when we come to treat the preservation of 'fixed' values within transformations (see § 10.6.).

10.5. Transformations, Binding, and Fixing

Transformations, in conjunction with the mechanisms for fixing values, and binding values to other values as described in the previous chapter, constitute a powerful tool for manipulating shapes. These three mechanisms working in combination with each other allow changes to individual shapes by means of *controlled distortions*. An illustration of particular distortions are shown in *figure 10-6*.

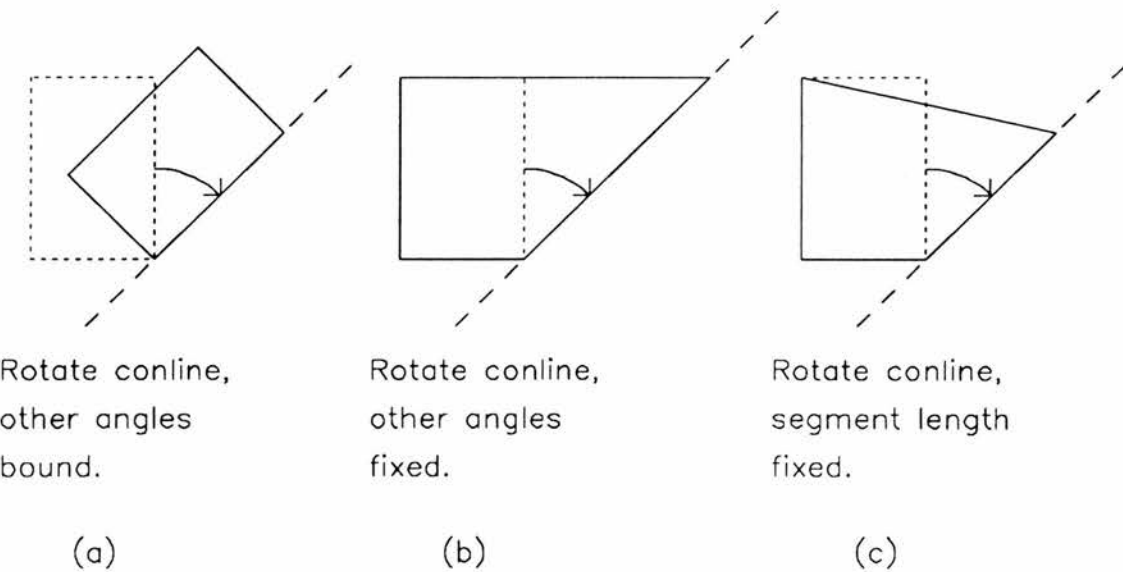


figure 10-6

Figure 10-6a can arise when a rotate conline transformation is applied to a rectangle

in which all the other conlines are bound to the one that is being rotated. This is in effect what is understood in conventional systems as a 'rotate' transformation. The conventional transformation of 'translation' can also be effected. Translation can be achieved by assigning a new co-ordinate value to the origin of a shape. Similarly, 'scaling' of a shape can be effected by binding all the segment lengths to the length of a particular segment. The system presented in this thesis, however, also allows for a wider variety of distortions. In the case of the rectangle, outcomes such as those shown in *figure 10-6b* and *10-6c* are easy to achieve. Controlled distortions, therefore, allow for parts of shapes to be included or exempted from transformations. How this works when two or more objects come together will be described in the following chapter.

We will first illustrate the use of fixing and of binding to effect controlled distortions in more detail.

10.6. Use of Fixing to Effect Controlled Distortions

Consider shape A shown in *figure 10-7a*, and suppose its conline angles and segment lengths have fixed statuses as indicated. Suppose also that the intention is to translate P_2 to P_2' (or P_3 to P_3' , or P_4 to P_4').

KEY:

Af Angle value fixed
 Au Angle value unfixed
 Lf Length value fixed
 Lu Length value unfixed

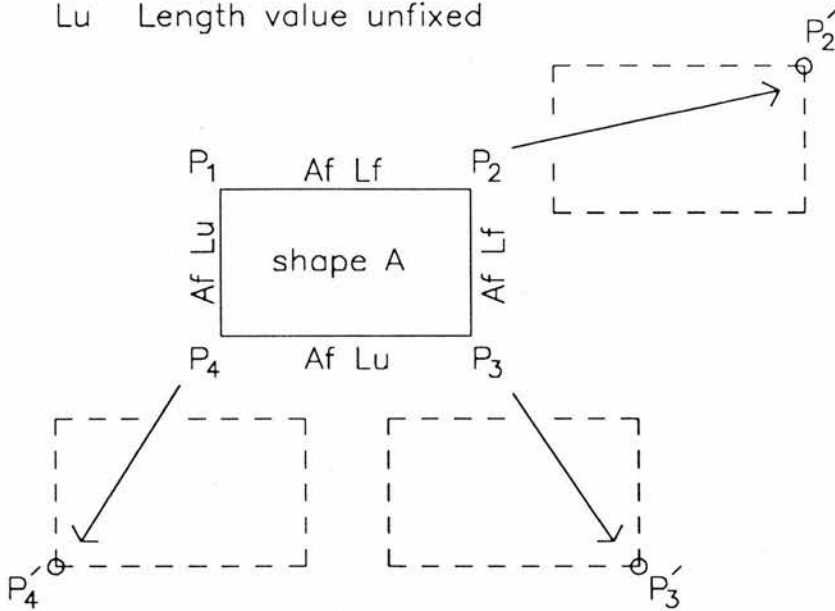


figure 10-7a : Shape A about to be translated

The drawing machine carries out this transformation as follows. It first finds all segments that have P_2 as one of their conpoints. For each of these segments, the statuses of their bearer angles and of their length values are noted. Any fixed values in this set are made unfixed so that one of the basic transformations can be applied. P_2 is then moved to P_2' by means of the (basic) move conpoint transformation.

The next step is critical to the preservation of fixed values. Each segment involved in the move conpoint transformation that possessed either a fixed length or a fixed bearer angle *prior* to the move conpoint transformation is *detached* at its other end (i.e. the other end to P_2). This is achieved by creating a new conpoint (P_n') made up from conlines whose angles are the same as those of the old conpoint.

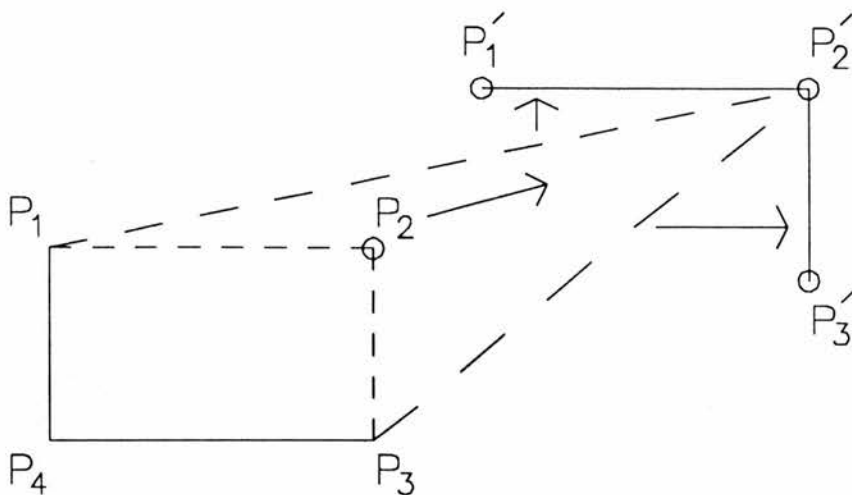


figure 10-7b : P_2 moves to P_2' - fixed values restored

This now allows for the restoration of previous fixed (angle or length) values (figure 10-7b). If new conpoints and old conpoints are coincident (i.e. their description in terms of conlines is the same), then segments can be reconnected by restoring the old conpoints. If new conpoints are not coincident with their corresponding old conpoints, then for each such pair (in any order), the drawing machine translates the old conpoint onto its corresponding new conpoint by repeating the procedure described thus far (figure 10-7c).

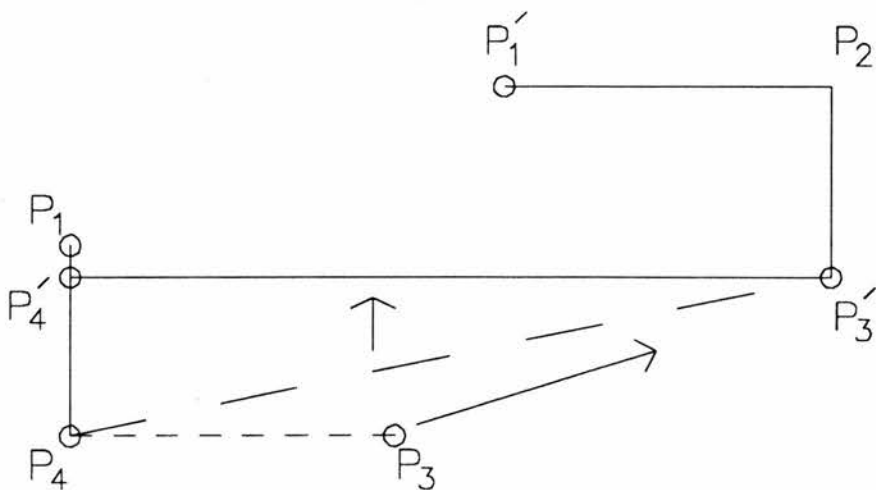


figure 10-7c : P_3 moves to P_3'

This procedure is repeated until there are either no new conpoints (i.e. of the form

P_n'), the result constituting a successful transformation. If the resolution of a new point with an old point entails retranslating the initial point (P_2 in this example), then this outcome results in *failure* of the transformation. In the example, the transformation is successfully completed by moving P_1 onto P_1' (figure 10-7d), and finally P_4' onto P_4'' (figure 10-7e).

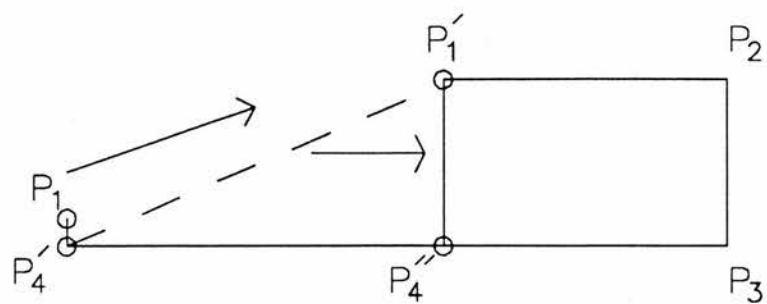


figure 10-7d : P_1 moves to P_1'

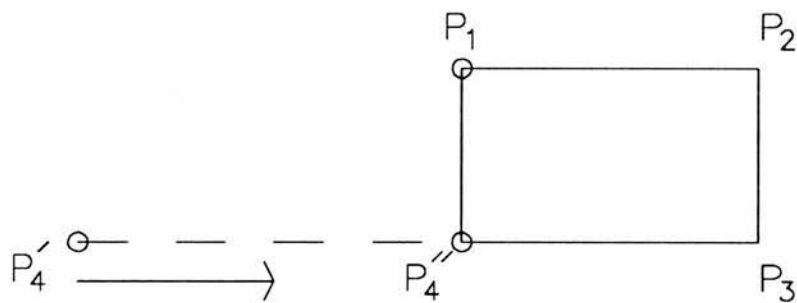


figure 10-7e : P_4' moves to P_4''

This last heuristic safeguards against regressive application of the recursive point translation transformation. In this respect, it is analogous to a heuristic used by Brüderlin [Brüderlin, 1985] for the construction of points. The procedure to which the heuristic used here is applied (recursive point translation) however, is mechanistically more complex than Brüderlin's recursive point construction. However, unlike Brüderlin, the procedure described here obviates the need for special geometrical construction techniques such as Brüderlin's circle intersections.

The same procedure applied to the goal of moving P_3 onto P_3' is illustrated in figures 10-8a to 10-8e. Moving P_4 onto P_4' in this example will give rise to yet another way of

arriving at a similar result.

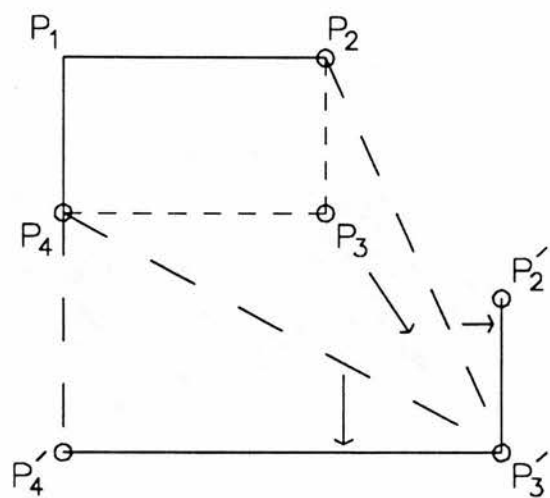


figure 10-8a : P_3 moves to P_3' - fixed values restored

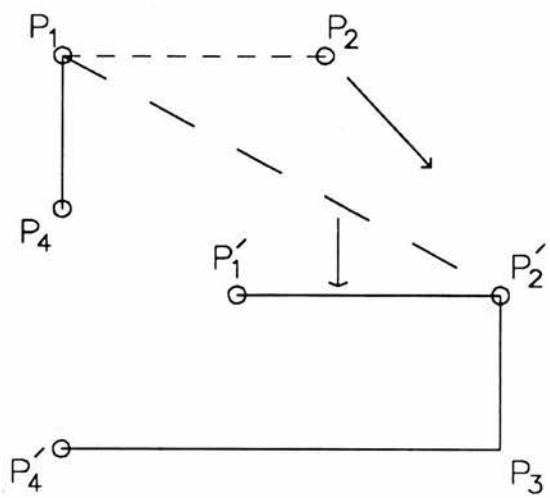


figure 10-8b : P_2 moves to P_2' - fixed values restored

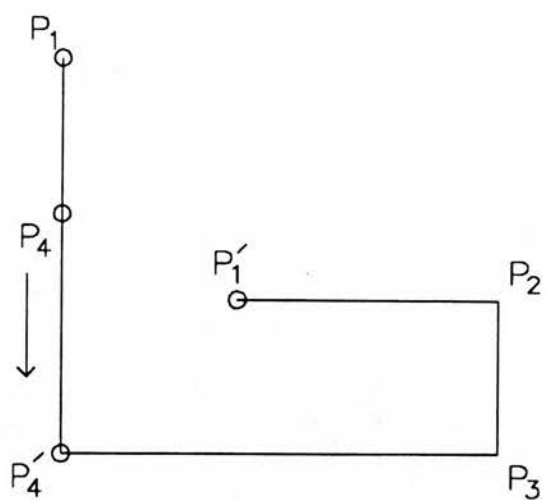


figure 10-8c : P_4 moves to P_4'

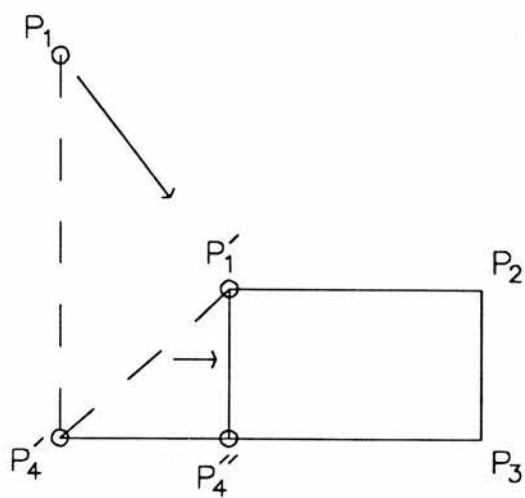


figure 10-8d : P_1 moves to P_1'

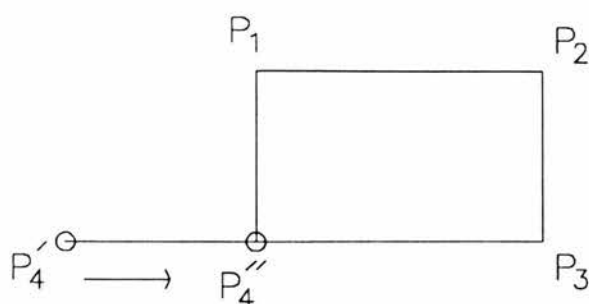


figure 10-8e : P_4' moves to P_4''

The general procedure outlined here allows translations to be defined on any point in a shape, such that the result preserves any fixed values of any parts anywhere within the shape.

It is evident from the above example that the procedure for implementing topological transformations permits the introduction of breaks within connected chains of segments. At first sight, this appears to contradict the aims of the previous chapter to support topological, and therefore *continuous* transformations. Since we are talking about transformations however, we are not interested in what happens anywhere except at the beginning and at the end. It is therefore permissible to introduce a break at some point provided it is eventually joined up again in the same way [Stewart, 1975].

Examples of outcomes of the transformation procedure when applied to the same shape (shape A), but with different parts fixed, are shown in figure 10-9a-c.

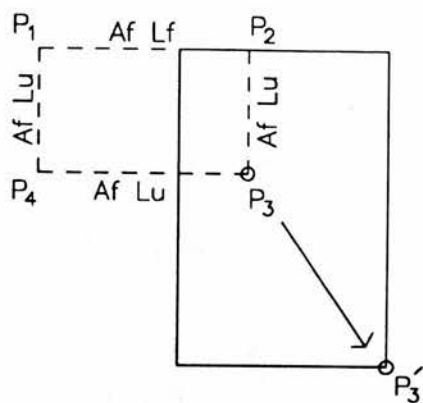


figure 10-9a

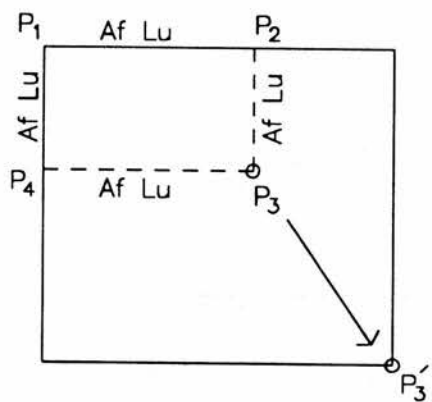
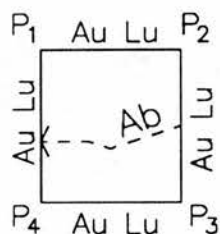


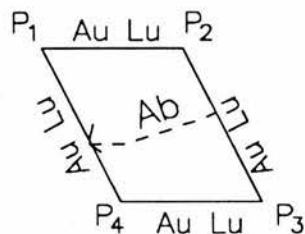
figure 10-9b

KEY:

Af Angle value fixed
 Au Angle value unfixed
 Lf Length value fixed
 Lu Length value unfixed
 Ab Angle value bound
 Lb Length value bound



(a)



(b)

figure 10-10

Suppose instead, that the binding had been expressed between the conline passing through P_1 and P_4 , and the conline through P_1 and P_2 (γ say), in a relationship of the form:

$$\gamma = \alpha - 90^\circ \quad (\text{figure 10-11a}).$$

If a similar change (to α) now takes place as before, we get *figure 10-11b* (or *10-11c*).

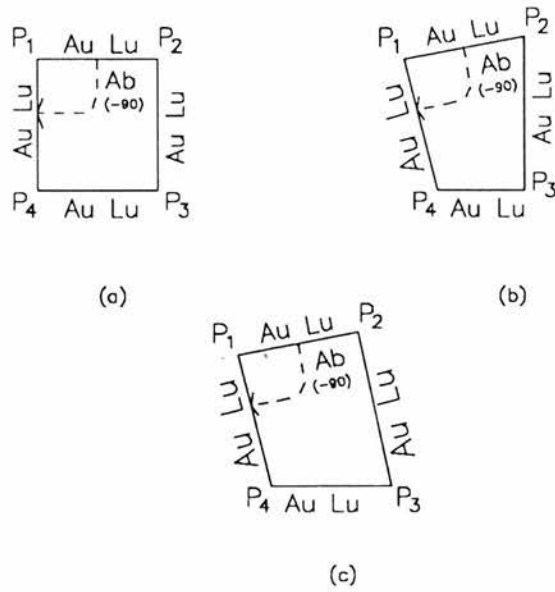


figure 10-11

Both of these examples show how binding relationships allow us to immediately deduce the bound value. The two alternative outcomes of *figure 10-11b* and *figure 10-11c* depend on whether the segment between P_2 and P_3 (case (b)) or the segment between P_3 and P_4 (case (c)) was regenerated first. The avoidance of such ambiguities is the responsibility of the user, and depends upon greater constraints being put upon the initial shape (e.g. fixing the angle value of the conline $P_2 P_3$ would guarantee the outcome in (b)).

Binding relationships can also be expressed between segment lengths (e.g. *figure 10-12a*). In this example, the length of the segment between P_5 and P_6 (L_2) has been set to be twice the length of the segment between P_1 and P_2 (L_1). If L_1 now increases, then so will L_2 , as shown in *figure 10-12b*.

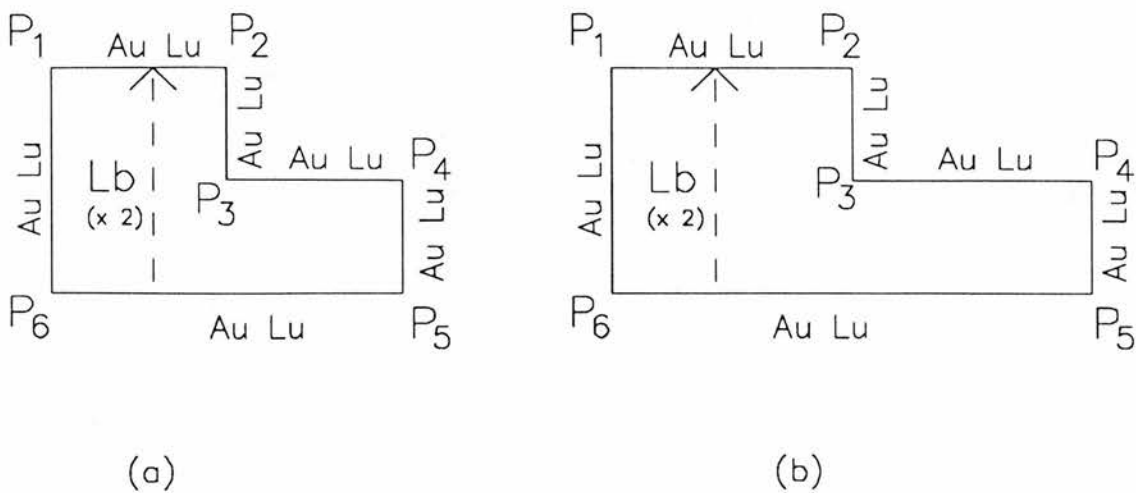


figure 10-12

10.8. Use of Fixing with Binding to Effect Controlled Distortions

Greater control over distortions can be achieved by the use of fixing and binding in conjunction with each other. A simple example is shown in *figure 10-13*. In this example, the segment between P_1 and P_4 has a fixed length, producing a slightly different outcome (*figure 10-13b*) to the one arrived at in *figure 10-10b*. In this case, the length of $P_1 P_4$ is preserved, and, as a consequence of the angle binding, so too is the length of $P_2 P_3$.

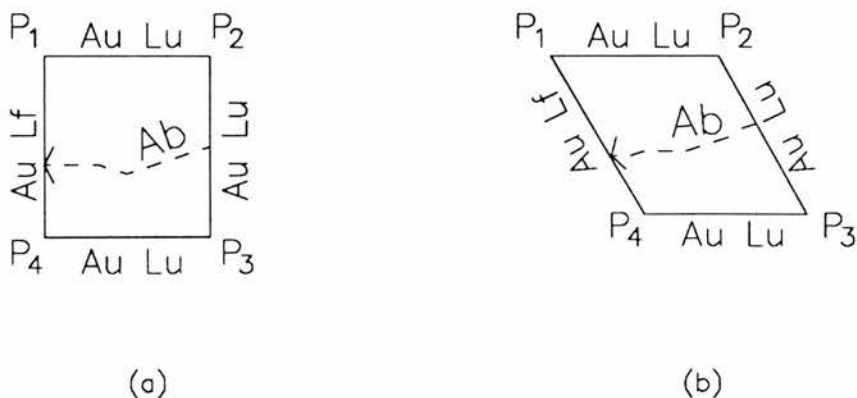


figure 10-13

A more complex example is shown in figure 10-14.

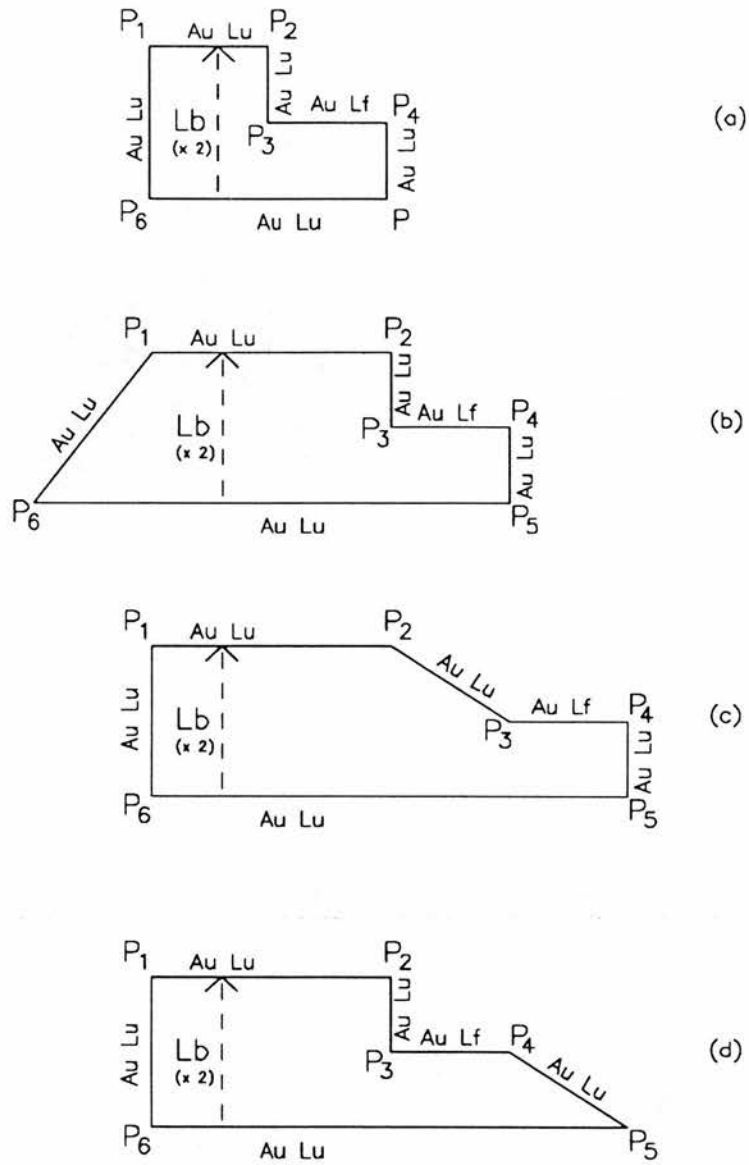


figure 10-14

Starting from an initial figure as in 10-14a, there are a variety of possible outcomes depending on which segment is regenerated first. Again, these could be reduced by further constraints upon the initial shape prior to the change length transformation.

10.9. Treatment of Transformations in Conventional CAD Systems

The transformations used in conventional CAD systems are forms of translation, rotation, and scaling. Transformations are typically applied to instances of system-defined elements, such as lines, boxes, circles, polylines and polygons. They can also be applied to groups of elements that have been indicated as such by means of a 'grouping' operation. In conventional systems such as MacDraft, MacDraw, and MacArchitron, such grouping implies the geometrical grouping of a set of elements within a region such that the region can move as a whole. Such a grouping can be achieved by means of defining regions within an existing picture. Regions themselves are usually box-shaped, but can also be free-hand drawn. Grouping can also be achieved by selectively picking out particular elements from amongst a set of elements. However the grouping is formed, the group itself (sometimes referred to as a 'symbol') can only be selected through its identifier or reference point (e.g. origins for symbols). A group formed using the Macintosh systems behaves as a rectangular region with its origin at the lower left-hand corner of the rectangle. This group can be transformed either by translating its origin, or by rotating about its origin, or by scaling along axes that meet at this origin.

Other forms of transformations, namely, boolean operations, are to be found in some 2-D CAD systems, and in 3-D solid geometry modelling systems. In these systems, relations between objects are usually represented by relations between their local co-ordinate frames, and the representation of compound objects consists of some relation between primitive objects, together with what are known as 'shape operations' [Earl,1987]. Shape operations can be represented in graph form. The problem with such systems, however, is that on the one hand one has objects which constitute logical wholes, and on the other one has some sort of hierarchical description of how such an object was constructed. At any moment in time, one cannot recover the individual parts from which a composite object was constructed other than by undoing the boolean join operations which formed it. Once a composite object has been formed, the distinction between its parts becomes blurred.

In architectural practice, objects are typically thought of as occurring in assemblies in which relations do not include interpenetrations formed as a consequence of applying boolean operations. In an assembly, objects touch one another, and this can occur in a number of ways. Any representation of assemblies invariably requires reference to vertices, edges, and, in 3-D systems, to faces. Each of these necessarily have to be viewed as symbolic objects that form *parts* of some graphical object. A single object, therefore, needs to be represented by relations between these primitive elements. The representation of assemblies can then be given by similar relations between vertices, edges and faces of

the assembled objects. However, in an assembly, components may move and so these relations may be variable. The representation of objects in terms of primitive elements such as vertices, edges, and faces, is at variance with the representation of objects purely in terms of the co-ordinates of endpoints of lines, and therefore with the way in which graphical objects are represented in most conventional CAD systems and solid geometry modellers. Relations between objects in assemblies are often variable and need to be represented as such. According to Earl, [Earl,op.cit.]:

"The interaction of a number of variable relations between objects and how they restrict actual variation is generally a difficult problem."

The representation of design objects in conventional CAD systems leads to an unsatisfactory treatment of variable relations. We have seen in this chapter, however, how the mechanisms of fixing and binding in the transformational component of the drawing machine can support the expression of variable relations.

An important observation to make about groupings in conventional CAD systems is that they are not logical groupings in the sense that the representation of a group constitutes a structure to which further properties can be ascribed and existing properties modified independently of the graphical effect of such changes. Grouping as it is used in conventional systems, prevents further access to individual elements within the group. The only way of recovering access to individual elements is to 'ungroup' the set of elements that were grouped. One is then again in the situation that the effect of moving any individual element is not to propagate changes to adjacent elements since there is no logical relationship between this element and any other element.

There is no way in conventional systems of saying things about other parts that meet any member of the group to be moved, and how these parts relate to the parts that are being moved. The question of how parts that abut onto/ touch/ have some particular spatial relation to a group that is being moved is unresolved in conventional CAD systems. Do these parts move with the group, or stretch onto the group, or do they get left behind? This criticism is true even of conventional CAD systems that have limited abilities to handle parameterisations within individual shapes. According to the Design Office Consortium report [Hamilton and Scoins, 1980], a system such as CADRAW [Ove Arup Partnership, 1973] is capable of parameterising the shape shown in *figure 10-15(a)* such that any of *figures (b), (c), or (d)* can be produced. The report considered CADRAW to be the best of its kind for handling parameterisation. This example, however, reflects the limited ambitions at the time of the report, since it is only length values that are parameterisable. The interplay between the parameterisations of angles *and* lengths is something that is not treated by any conventional CAD system.

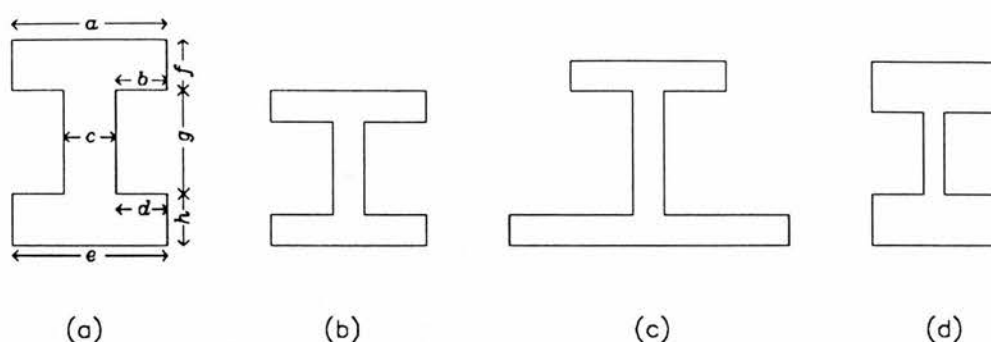


figure 10-15

Once transformations are applied in conventional systems, they are applied *uniformly* to all members of a group. A translation of a group of segments translates each segment by the same amount. Scaling a group of segments in the x -direction scales each segment by the same amount in the x direction. In conventional systems, segments are defined in terms of the co-ordinates of their end points. Conventional transformations therefore, apply to point sets. There is a sense therefore, in which conventional systems do preserve connectivity, since shared points prior to a transformation remain shared points after a transformation. In applying a conventional transformation to a group of segments, however, one cannot describe any segment within this group separately to any other segment, such that the graphical effect of the transformation upon this segment will be different from the graphical effect of the transformation upon any other segment. One could apply two or more consecutive conventional transformations to separate parts of a picture, but in this case, the connectivity relations between segments will invariably be broken as a consequence of defining the groups to which any transformation will apply.

The user of a conventional system frequently has to re-draw parts that previously existed, but that got 'left-behind' as a consequence of some transformation. Additionally, parts that one thought were not part of a transformation but were included in the group also have to be redrawn back in their original positions. There is often a mismatch between any conventional grouping of lines and the user's intentions for these lines. This points to a lack of topological knowledge embodied in conventional CAD systems. Topology is concerned with those properties of graphical objects that remain invariant under transformation, such as the connectivities between parts. Conventional systems not only

lack expressive mechanisms for describing the connectivities between parts, but also lack the underlying representations that would be required to support such mechanisms.

The essence of the transformational component of the drawing machine described in this thesis is that topology combined with logical operations can allow a user to describe graphical parts and to describe relationships between these parts, such that these relationships can be preserved during the course of transformations. The deficiencies of conventional CAD systems stem primarily from the opacity of their logical representations of graphical objects to the users of these systems. Through the transformational component of the drawing machine described in this thesis, we can begin to see how giving users some form of direct access to logical representations, even if only read-only access, can enable them to see the correspondence between depictions and the system's representation of these depictions. The interpretation of transformations becomes more transparent since the logical structures prior to and after a transformation are visible. Allowing users to constrain the individual elements involved in a transformation by changing their properties, or by relating properties of elements to properties of other elements, can only be achieved if users have write access to the logical representations of drawings. Constraints on transformations of shapes might affect all or only some conline angle or segment length values within a shape. The binding mechanism means that changing a conline angle value can have the effect of rotating a shape, or that moving a conline intersection point can have the effect of translating a shape, or that changing the length of a segment can have the effect of scaling a shape. In each case, the corresponding conventional transformation is only one of many transformations that are possible with the transformation component of the drawing machine.

The form of knowledge contained within conventional CAD systems is essentially geometrical. In architectural and other forms of design, there are numerous cases in which the designer has a particular view of the logical and topological structures of objects. It is essential that CAD systems should reflect not only the geometrical appearances of a design, but also the logical structure of any particular design. One can envisage a situation in which the logical structure of a machine depiction is completely at variance with the logical structure in the mind of the person looking at the depiction. This may be alright at the time of observation, but as soon as the user wants to do something with the machine depiction, this requires understanding of the logical structure of the machine representation to invoke the required responses in the system. If the user had himself produced the drawing by describing the logical structure as he went along, then there would not be so much conflict between the user and the machine. But then this assumes that the system possesses a representation environment which is capable of accepting expressions which convey a user's intentions for any particular logical structure.

10.10. Summary

We have outlined in this chapter the mechanisms by means of which transformations and controlled distortions can be produced. The outcome of any transformation that a user of the drawing machine might intend is inextricably governed by the combination of *fixed* and *unfixed* status values (associated with conline angles and segment lengths) *prior* to the transformation taking place, together with any *bound* values. It also irrevocably depends upon the state of connectivity between graphical parts *prior* to the transformation taking place.

There is a limit to the kinds of shapes that can be handled or produced by the transformation procedure outlined in this chapter. Possibilities are limited by the availability of only straight conlines, conpoints, and segments. It is possible to conceive of conlines with curvature such as circles, but these in turn would raise further problems. In the system proposed here, shapes and transformations need to be specified in such a way that new shapes can be produced by the drawing machine without resorting to the use of other familiar Euclidean geometrical constructs. This may in turn mean that some shapes and transformations will need to be overspecified in relation to a users's understanding of geometry.

This chapter has shown how shapes can be distorted in controlled ways by allowing users to redescribe the logical properties of these shapes prior to the execution of transformations. This represents a significant advance over conventional geometrical transformations.

11. Attachment

11.1. Introduction

The concern of this chapter is with the ways in which two or more objects can be attached to each other, both graphically *and* logically. The need for attachment arises from the view of CAD which places emphasis on the importance of description of object *configurations*. The link between the drawing machine as it has been described so far, and the practical concerns of architects and designers, will become evident in the operations of attachment between two or more objects. Attachment operations allow the possibility of creating a variety of formal constructions which are not constrained by any direct practical application. It is through the use of such operations that designers can explore design possibilities.

Attachment between shapes can give rise to a multiplicity of possible outcomes. The interest here is not in the quantitative enumeration of these possibilities; rather, it is in the *qualities* of their expression. Again, this can be contrasted with other approaches such as shape grammars (chapter 5).

11.2. Changing Shapes

An important part of CAD systems is the extent to which they support the expression of configurational relationships between shapes. Shape configurations arise as a consequence of interactions between objects. *Attachment* is the label given to a conglomeration of processes that take place when two or more objects are brought together into some kind of configuration. These processes involve edits (changes to object descriptions), transformations, and the exploitation of features such as fixed statuses and bound values. One of the nice features of the representation of shapes used in this thesis is that configurational relationships between objects do not need to be expressed in any explicit fashion, such as "*A is next to B*"; or "*A touches B*"; or "*A overlaps B*"; etc. Rather, such relationships are *implicit* in the representation of attached objects, and follow as a direct consequence of how much or how little the attached objects have in common. For example, the implicit representation of an explicit relationship such as "*If B is rotated then wherever it ends up, A will be drawn from the point it has in common with B*" is achieved by making the origins of both A and B refer to the origin of the composite object by means of indirections.

A consequence of the fact that the representation consists of named lines that bear symbolic dimensions and angles is that it becomes possible to express configurational relations between objects. Typically, these implicit relations are maintained even when the drawings change. For instance, consider a drawing such as *figure 11-1a*. A house is 40' wide, 20' of which is the sitting room, and 20' the kitchen. Suppose now that the width of the sitting room needs to be increased to 25'. What should the new drawing now look like? Will it end up as *figure 11-1b* or *figure 11-1c*? The outcome should be decidable from the way in which the original drawing was constructed.

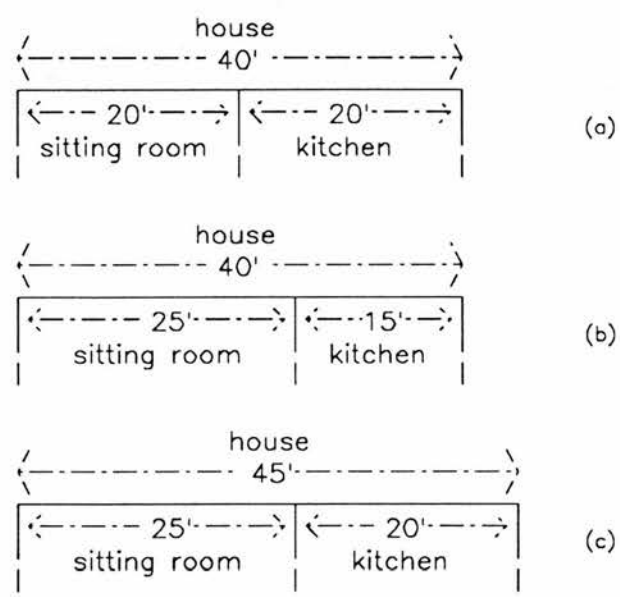


figure 11-1

The strategy proposed here is that configurational relationships should be expressed in terms of indirections, in which one part of an object indirections to the description of some other part. An example of indirection during attachment occurs, for example, when the lengths of segments between pairs of pairs of points that are being attached (each pair belonging to a distinct object) are made symbolically equal to each other. Another example occurs when the conlines on which pairs of pairs of points lie are declared to be the same in a composite drawing.

Such facts will interact with concrete details about a picture, to fix the eventual drawing. For instance, to solve the problem about house width, a suggested order of events could be as follows.

Pictures of the sitting room and the kitchen are first drawn. The widths of the sitting

room and the kitchen are initially set to be 20'. Both widths are initially of unfixed status (*figure 11-2a*). First the drawing of the sitting room is planted as part of the drawing of the whole house (*figure 11-2b*). Then conlines are added to the planted drawing until there are enough conpoints onto which enough points of the kitchen can be mapped in order to hold it fixed relative to the sitting room (*figure 11-2c*). Then the conpoints in the drawing to be added can be associated with conpoints in the drawing already located (*figure 11-2d*). This will establish the topological relation between the kitchen and the sitting-room. The drawing of the house should now look like *figure 11-2e*, except for the dimensions between the conlines A,B,C.

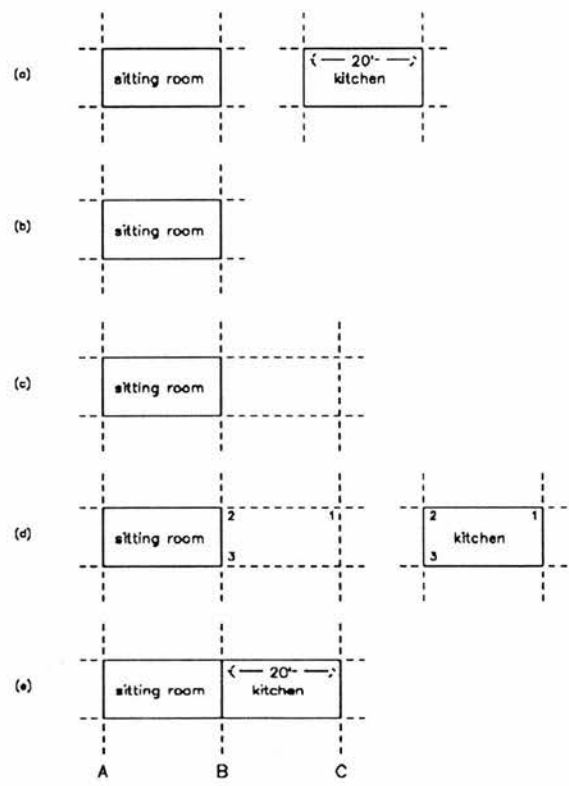


figure 11-2

How can alternative dimensional relationships be expressed?

First, the lengths AB and BC are initially 20' each. Then one can either:

- 1 Declare the length of the segment AC, which is the total width of the house, to be *fixed* at 40'. Expressing the relationship in this way means that the width of the kitchen will follow by subtraction once the length of the sitting room is increased.

- 2 Declare the length of the segment BC, which is the width of the kitchen, to be *fixed* at 20'. Using this method of expression implies that the width of the total house will follow by addition.

What happens if the sitting-room is made wider depends on which of these possibilities is chosen. If the first, then the width of the house is constant, and so the kitchen will shrink. If the second, the width of the kitchen is constant, so the width of the house will increase.

This approach handles another type of problem as well. Imagine that both a bath and the walls of a bathroom have been drawn (*figure 11-3a*). Now one puts the bath in the bathroom. Suppose (as is common in drawing systems) one locates the bath relative to some fixed point in the bathroom - say at the lower left-hand corner. (*figure 11-3b*). Now suppose the bathroom changes width, from 7.5' to 9'. What happens to the bath? If no configurational relationship has been expressed, then it will be stranded in the middle of the enlarged bathroom. (*figure 11-3c*).

What should have happened is that the left-hand side of the bath and the left-hand wall of the bathroom should have been attached to each other. Thus when one moved the other would have been dragged with it. How is this done? *Figure 11-3d* is like *figure 11-3a*. If the conpoints are attached as shown, the conlines on which lie the left-hand sides of the bath and bathroom will be declared to be the same, and if one of them is moved in the composite drawing, they will move together, to produce, not *figure 11-3c*, but *figure 11-3e*.

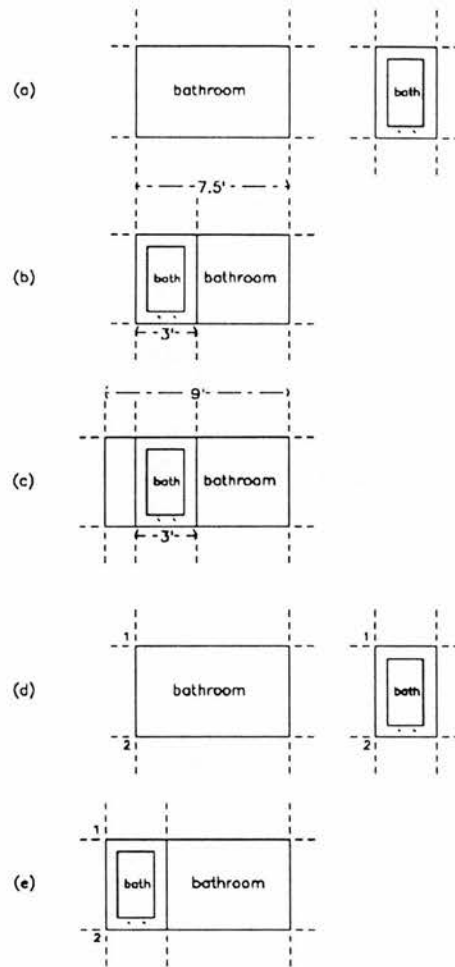


figure 11-3

This last example illustrates how the way in which objects are attached will have implications for subsequent changes that might be made to the new configuration. The behaviour of a composite object in which sub-objects are tightly bound to each other (i.e. sub-objects have several graphical parts in common with each other) is such that if any sub-object is moved, then the rest of the sub-objects will also move (to a greater or lesser degree). On the other hand, a composite object whose sub-objects are loosely bound may appear to break up once any of its sub-objects is moved, leaving others behind in the process (although logically it may still remain a composite object).

11.3. Attaching Two Shapes

Consider the apparently simple task of attaching together two objects such as those shown in *figure 11-4*. Each of the shapes A and B has its own origin defined in a separate co-ordinate space.

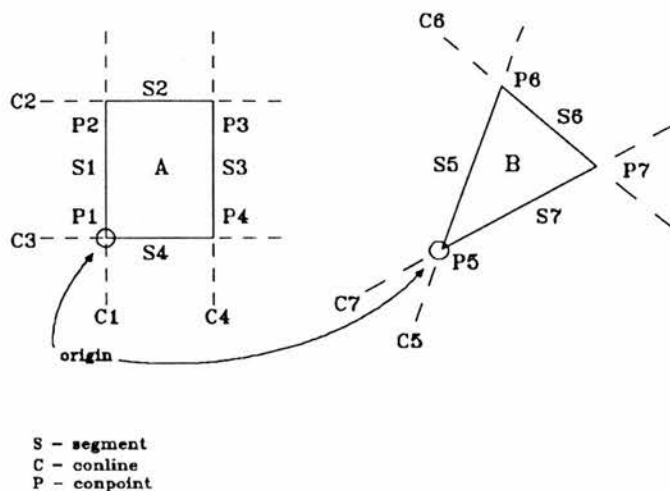


figure 11-4: Two objects to be attached

Depending upon the intentions of the user, one of several possible outcomes could arise, some of which are shown in *figure 11-5*.

The logical representation of a composite object requires a common point of reference between the attached objects. This implies at least one conline in common, as otherwise the two objects remain independent. Figure 11-5a illustrates a case in which the user may want to constrain the conlines of two separate shapes to be parallel. Figure 11-5b expresses a colinear relationship between conlines. Figures 11-5c to 11-5j assume that there is at least a conline, and one or more conpoints in common. In order to be able to distinguish between these different outcomes, the drawing interpreter needs to possess knowledge of drawing operations to arrive at these user-described outcomes. This knowledge embraces drawing operations and how they may interact with each other, rather than an exhaustive classification of possible outcomes.

Typically one works with a *location* drawing into which one can fit objects that may have been previously defined. When such insertions take place, either segments of the location can fix dimensions of the *component* drawing, or vice versa. There is no reason to suppose that one drawing should be dominant over another with respect to length

dimensions. However, for angles, it is reasonable to assume that one of the drawings should be fixed in this dimension, since it is effectively providing a grid upon which the attachment can take place.

Already specified graphical objects are presented in separate screen windows if they are needed for attachment. Attachment chains (sequences of conpoints) are indicated on location and component drawings in order that the drawing machine can establish which graphical parts are to be attached. Shapes need to be dimensioned before they can be drawn for the purpose of indicating attachment chains.

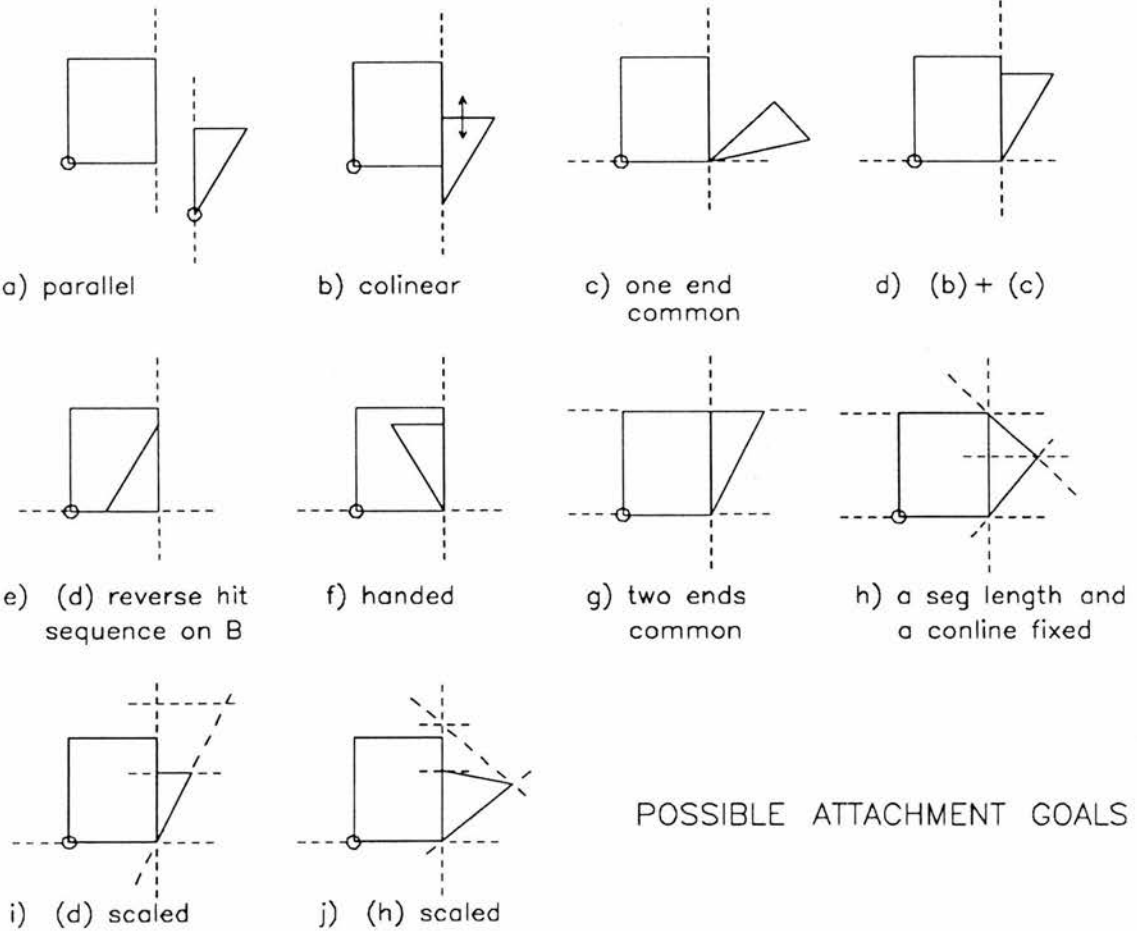


figure 11-5: Possible attachment goals

11.4. The Logic of Attachment

Attachment brings objects together not only graphically, but also logically, to create a composite object. The graphical 'moving together' involves the invocation of the transformations described in the previous chapter. Logical attachment, on the other hand, can best be described in terms of the edit operations provided by the representation environment. Logical attachment can take several forms, and we need to explore the various kinds of attachments by which graphical objects (shapes) combine to form new graphical objects (compositions). Each kind of attachment is essentially simple, but can lead to the creation of complex constructions, both in terms of the refinements in description that led to the creation of a new composition, and in terms of the richness of new forms that may arise as a consequence of operating upon this composition. Some logical possibilities are shown in *figure 11-6*:

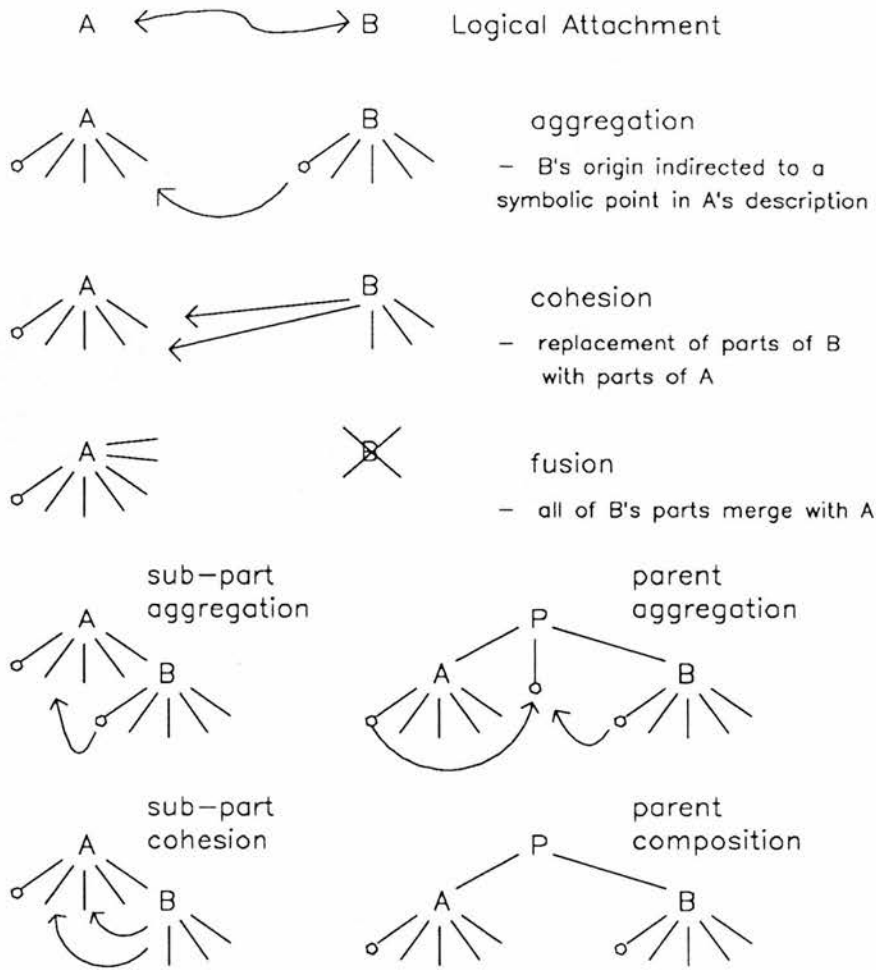


figure 11-6 : Logical Representations of Graphical Attachments

11.4.1. Aggregation

The component is loosely placed over the location by specifying its origin as taking a co-ordinate value of one of the points of the location. This is done by means of an indirection. The effect on the dynamics of the resultant object is that whenever what used to be the location part is subsequently moved, the attached object (what used to be the component) is dragged with it by means of its origin. This is the most tenuous form of attachment in that the attachment can be broken relatively easily by replacing the origin indirection that binds the two objects together by some symbolic point which can be given an independent coordinate value. The effect of this breaking would mean that although the two shapes are logically attached, they would be graphically separate.

For two objects A and B, as in *figure 11-6*, aggregation can be achieved by indirecting B's symbolic origin to some symbolic point in A's description. In terms of the representation environment's logical operators, this can be expressed by an expression of the following form:

$$B:origin <+ A:seg_x_j:conpoint_y_k$$

11.4.2. Cohesion

Parts of the component take the descriptions of parts of the location (implemented in terms of the << operator of the representation environment). The sub-objects behave as if they have been 'stuck-together'. The effect of this is that whenever transformations are subsequently applied to the location part of the composite object, they will invariably have 'knock-on' effects upon the attached object (what was the component). This form of attachment binds objects together in a tighter fashion in that it becomes more difficult to unpick the sub-objects from the composite object. Such an unpicking would have to be done by re-establishing those graphical parts that were replaced during the attachment.

In terms of *figure 11-6*, logical parts of B that have graphical depictions (i.e. segments, conpoints, or conlines) replace the descriptions of corresponding parts of A, e.g.:

$$A:seg_x_j << B:seg_x_j, \text{ or:}$$

$$A:seg_x_j:conpoint_y_k << B:seg_x_j:conpoint_y_k, \text{ or:}$$

$$A:seg_x_j:conpoint_y_k:conline_z_l << B:seg_x_j:conpoint_y_k:conline_z_l$$

11.4.3. Fusion

The component loses its identity and its description fuses into the description of the location. It does this by having certain of its parts taking the descriptions of parts of the location, and all the remaining parts declared to be parts of the location. Fusion effectively adds graphical parts from one object (the component) to the location. Undoing this form of attachment would involve a considerable amount of re-description. Fusion is implemented in terms of the <> operator described in chapter 7. Those parts of the component that take descriptions of parts of the location do so by means of renaming (implemented in terms of the << operator). Referring to *figure 11-6*, the fusion of B onto A can be described by:

$$A \diamond B$$

11.4.4. Sub-Part Aggregation

Same as aggregation except that the component becomes a logical sub-part of the location. In *figure 11-6*, sub-part aggregation between A and B can be achieved by means of the following two logical expressions:

$$B:\text{origin} \leftarrow A:\text{seg_}x_j::\text{conpoint_}y_k \quad , \text{ and:}$$

$$A \leftarrow [\text{subpart}=B].$$

11.4.5. Sub-Part Cohesion

Same as cohesion except that the component becomes a logical sub-part of the location. In terms of *figure 11-6*, logical parts of B that have graphical depictions replace the descriptions of corresponding parts of A, e.g.:

$$A:\text{seg_}x_j \ll B:\text{seg_}x_j \quad , \text{ or:}$$

$$A:\text{seg_}x_j::\text{conpoint_}y_k \ll B:\text{seg_}x_j::\text{conpoint_}y_k \quad , \text{ or:}$$

$$A:\text{seg_}x_j::\text{conpoint_}y_k::\text{conline_}z_l \ll B:\text{seg_}x_j::\text{conpoint_}y_k::\text{conline_}z_l.$$

And, additionally,

$$A \leftarrow [\text{subpart}=B].$$

11.4.6. Parent Aggregation

A parent object is created. Both component and location are related to the parent through their origins. In *figure 11-6*, once P has been defined as being a graphical object, parent aggregation can be achieved by means of two additional expressions of the following form:

$$A:\text{origin} \leftarrow P:\text{seg_}x_j::\text{conpoint_}y_k \quad , \text{ and:}$$

B:origin <+ P:seg_x_j:conpoint_y_k

11.4.7. Parent Composition

A logical parent is created, with component and location becoming parts of it. It is not clear how such an object can be realised graphically, since there is not necessarily any graphical relationship between the parent and the component, or between the parent and the location. The two expressions that establish parent composition in *figure 11-6* are:

P <+ [subpart_1=A], and:

P <+ [subpart_2=B]

11.5. Worked Example

How the drawing machine operates on the representation environment when executing changes is illustrated in *figure 11-7*. Beginning with a drawing of two shapes, a rectangle A and a triangle B (*figure 11-4*), we want to make a new shape C which is to be formed by joining B to A, an example of an attachment operation. The user has to declare which of the basic transformations is to apply, rotate conline, scale segment, or translate conpoint. The segments that will be used to effect the change are indicated by the user hitting the end points of a line (S5) of shape B and then hitting the end points of a line (S3) of shape A on the drawing. This has two effects. Firstly, that B has to be rotated about its point P5 with respect to the conline C4, and secondly that P5 is to be translated onto P4. The user may want to add that S5 is also to be scaled such that it matches S3's description.

The two logic diagrams in *figure 11-7a* and *figure 11-7b* show the description of the separate shapes A and B prior to being changed, with reference to the segments, conpoints and conlines that make up the drawing of these shapes. We may assume, if nothing else is said, that length values of segments have been taken from the drawing, and that these values fill slots attached to segment kinds (not shown).

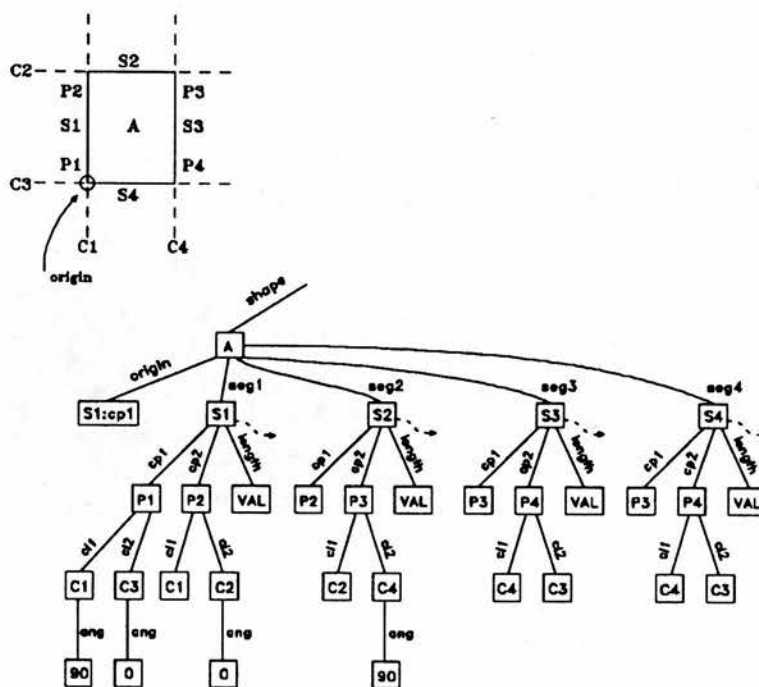


figure 11-7a: Representation of Shape A

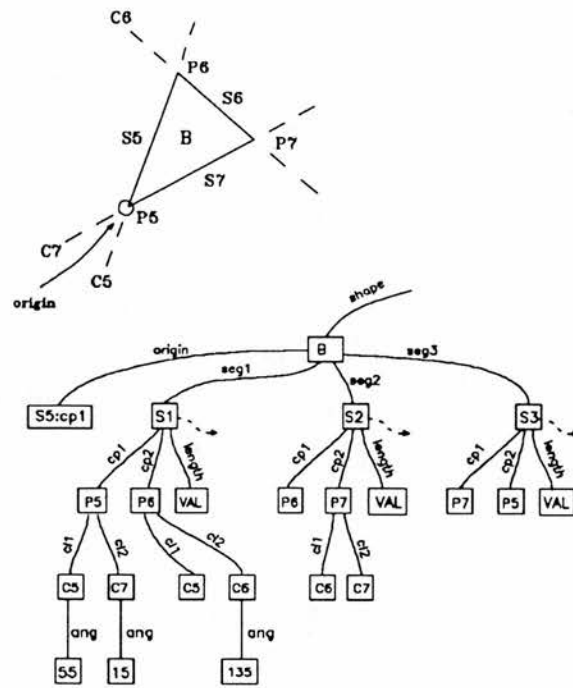


figure 11-7b: Representation of Shape B

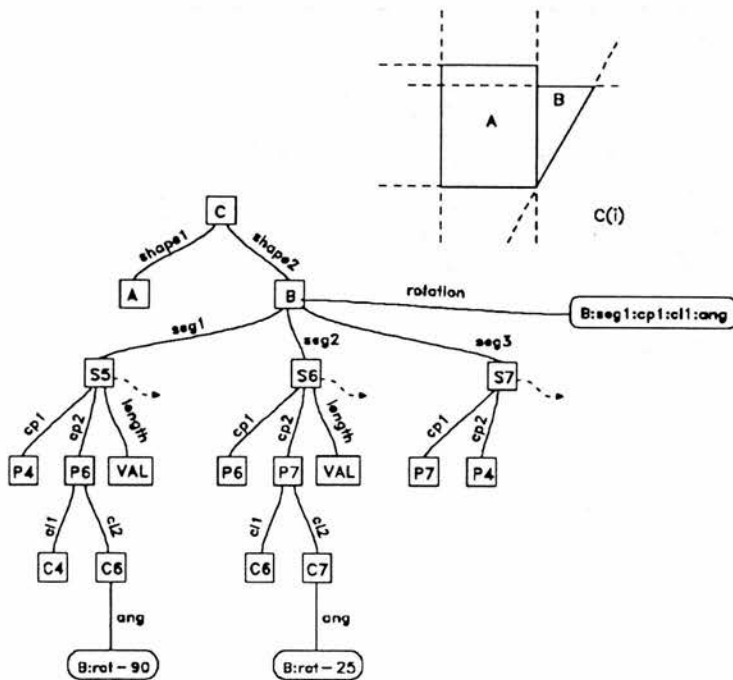


figure 11-7c: Rotation and Translation of B

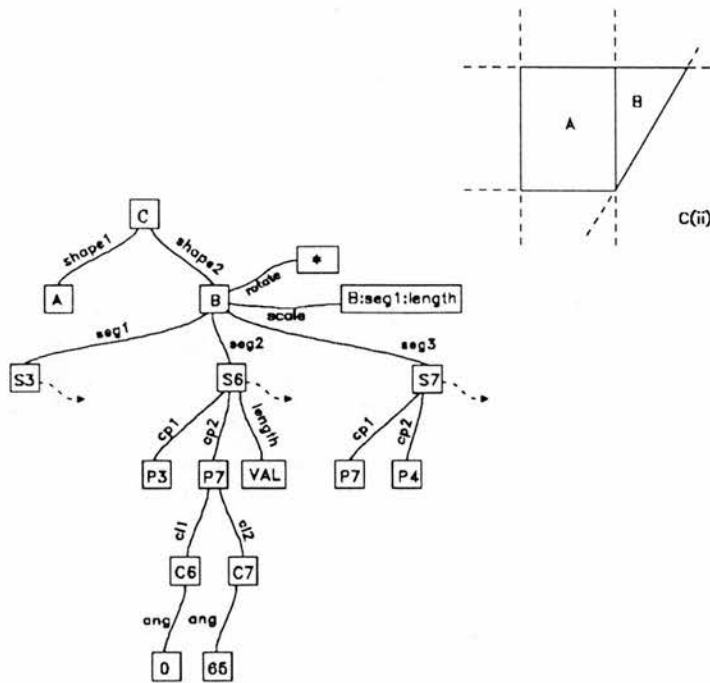


figure 11-7d: Scaling of B

To attach B to A, a new shape C is declared that has as its parts A and B. The drawing machine then fills the "rotate" slot of B with an indirection to the angle value of the conline bearing the segment of B that has been indicated by the user. All angle values of B are then bound together so that if one changes all change.

Next, shape B has to be moved up against A. The logic diagram in figure 11-7c shows the drawing machine having replaced all instances of conpoint P5 with P4, and all instances of conline C5 by C4. At this stage, the drawing machine has enough information to regenerate the drawing as shape C(i) which may represent what the user said he wanted.

Additionally, the user may have stated that S5 is to be scaled to match S3. To change the scale of shape B so that its edge joining A is equal to the length of the edge it joins in A, the drawing machine fills the "scale" slot of B with an indirection to the segment that forms the drawing line that has been indicated by the user, as shown in the last logic diagram figure 11-7d. All length values of B are then bound together so that if one changes all change. Finally, all instances of segment S5 in B are replaced by S3 (only one in this example) and all remaining length values of B are adjusted with reference to the length value of S3. The drawing machine is then able to regenerate the drawing as shape C(ii).

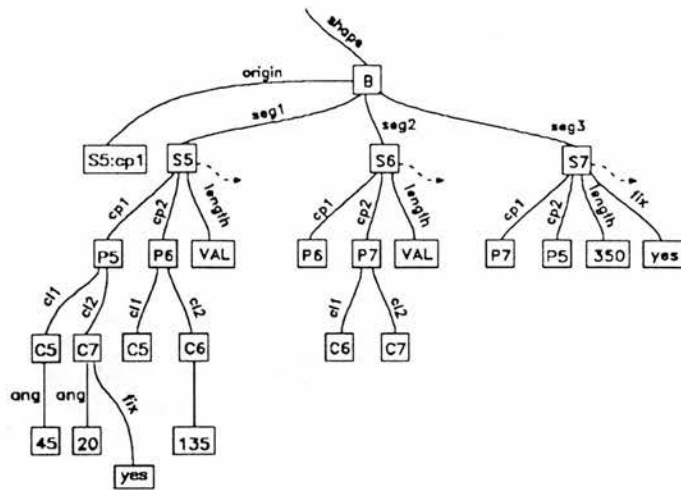


figure 11-8a: Fixing one segment length and one conline angle

11.5.1. Fixing

Recall that the representation of conlines and segments includes a reference to whether or not their dimension values can subsequently be overwritten, indicating whether they are "fixed" or not. This is a truth value which is the filler of the fixed slot of the segment kind.

In the previous example, all segment lengths and conline angles in B were left as inferable from values in A. It is possible to fix any of these values so that they are excluded from a change operation, with consequent changes in the resulting new shape. For example, one might want to preserve the angle of conline C7 and the length of segment S7 while making the same attachment as before. The user needs to indicate that these are the dimensions he needs to fix, and the drawing machine would generate an initial description of shape B as shown in figure 11-8a.

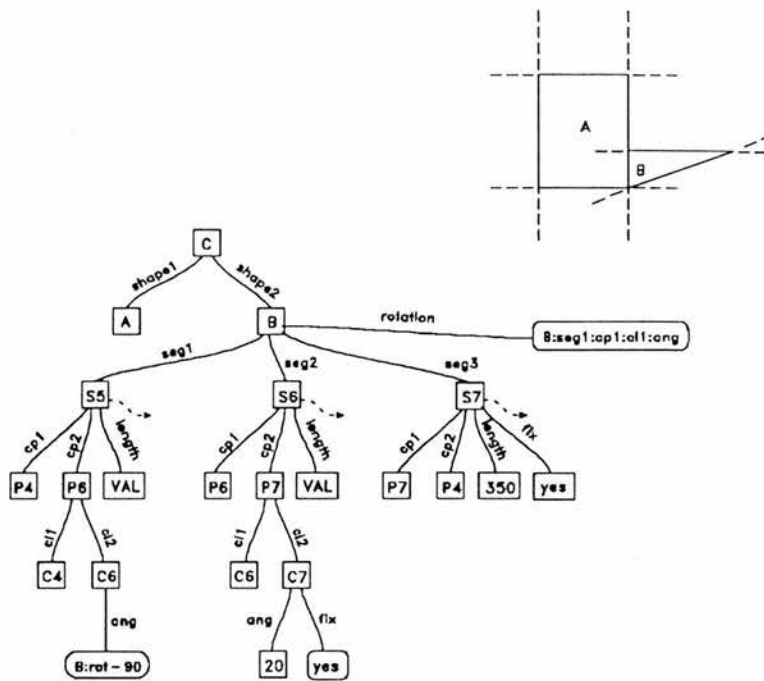


figure 11-8b: Rotation and Translation of B

If the user proceeds with the attachment in the same manner as before, then rotation and translation produces the representation shown in *figure 11-8b*. Notice that this time the value of C7's angle has not been affected by the binding of angles relative to C4's angle. Similarly in the translation operation, the fixed values remain fixed. Notice that with a fixed length dimension, it will scale the object relative to the part to which the fixed dimension belongs.

When this attachment was done without fixing, S5 was taken to be the default reference segment for scaling purposes. This time, moving P6 onto P3 cannot be done using the scale operation, but instead requires an edit of the composite object C such that P6 stays where it is, but the common endpoint of S5 and S6 moves to P3. At first sight this doesn't look quite right since S6 has lost its bearer. But it is not essential that segments should lie on bearers provided their end points are at existing conpoints. The final shape is shown in *figure 11-8c*.

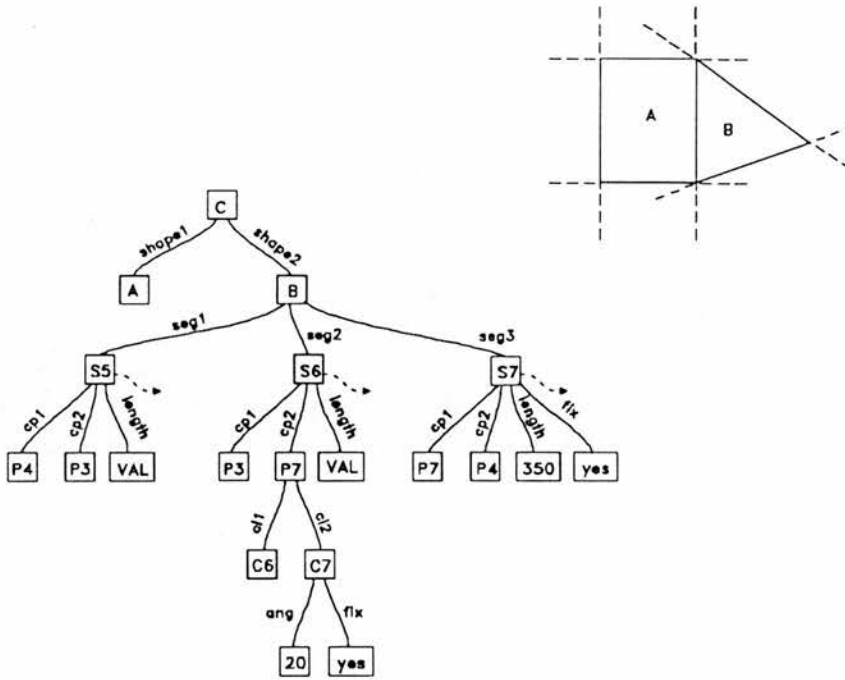


figure 11-8c: Moving a segment end point

Making use of fixed statuses in this fashion, as was done by fixing values associated with shape B in this example, again allows one to produce controlled distortions when shapes are attached.

11.5.2. Types of Attachment

In the previous example, the description of B was merged with that of A so that the merged segments of B disappeared. Alternatively, B could have been attached to A in a manner that would have kept B's segments as distinct parts of B lying on top of the corresponding segments of A.

By now it should be apparent that there are different kinds of attachment, different goals that a user may have in mind when joining one shape with another. The purpose of the representation environment's change operations is not to anticipate correct results for particular kinds of change, such as an attachment between two drawing objects, but is to allow a user to describe any required result. Again, it is evident that controlled distortions can take place just as easily when objects are attached, as they could with individual shapes. The equivalent attachments to the controlled distortions illustrated in figure 10-6 in the previous chapter are shown in figure 11-9.

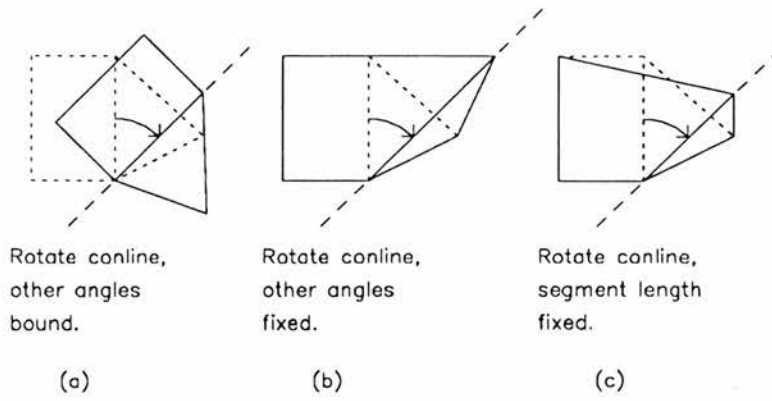


figure 11-9

11.6. Knock-On Effects of Qualitative Changes

Returning to the discussion of details in chapter 4, suppose that at some stage in the design process one would like to insert a detail into a more general layout drawing. Suppose the detail is something like the detail of the floor covering for a floor. Typically, such a drawing is bound to include information which, when combined with the representation for the layout of the whole floor, should be propagated to the whole of the floor, and not just stop where the detail meets the rest of the drawing. How does one know where such extrusion information should stop, or how it should be joined to other parts when it does stop?

This problem can be solved by indicating on the location drawing, the precise points which will delimit the extent of the component detail when it gets inserted, and the corresponding points on the detail itself. An invariable consequence of most attachments that take place in this fashion, is that any subsequent transformation which is applied to either of the sub-parts of the newly formed composite object, will have *knock-on effects* upon the other sub-part to which it is attached. These are a direct consequence of the transformation of one object onto another during the attachment, in that the transformation procedure attempts to preserve fixed and bound values.

11.7. Treatment of Attachment in Conventional CAD Systems

We saw in the previous chapter how conventional CAD systems treat the movement of objects in comparison with the greater control over movement that is possible with the transformation component of the drawing machine. We now have to consider how attachment is dealt with in conventional systems. We observed during the course of the worked example in chapter 6, that a 2-D drafting system such as MacDraw or MacDraft has very little concept of attachment whatsoever. In these systems, the user cannot join graphical parts with other graphical parts such that the connectivity of these parts is preserved despite subsequent changes to properties (such as angle and length values) of any of the connected parts.

A more sophisticated system such as MacArchitron [MacArchitron, 1987] provides functions (the 'L-Join' and the 'T-Join' functions) which enables two segments to be joined together. The 3-D version of MacArchitron has equivalent 3-D functions which operate on 'blocks'. The most significant observation to be made here is that the joining of segments using these functions is not a logical one, but purely a geometrical one. There is no sense in which the joined segments are considered as being parts of the same composite object. In contrast, in the system described in this thesis, the drawing machine, in conjunction with the representation environment, works on the principle that making attachments between shapes is a general means of building up shape information. At the most primitive level, drawing a single line, adding one line to another or adding one line to a collection of other lines (a shape) involves attachments to lines or line intersections (conlines and conpoints). Line by line additions to a logical representation of a design are generated by the drawing machine. Attachments between shapes that each consist of separately assembled collections of lines, and may each exist in separate drawing spaces, require direct use of the logical operators of the representation environment.

From the geometrical point of view, the MacArchitron 'L-Join' and the 'T-Join' functions work on the principle that the join will be located at the intersection point formed between the implicit 'construction lines' upon which the segments to be joined lie. One can envisage many situations in which this default assumption would be an incorrect one.

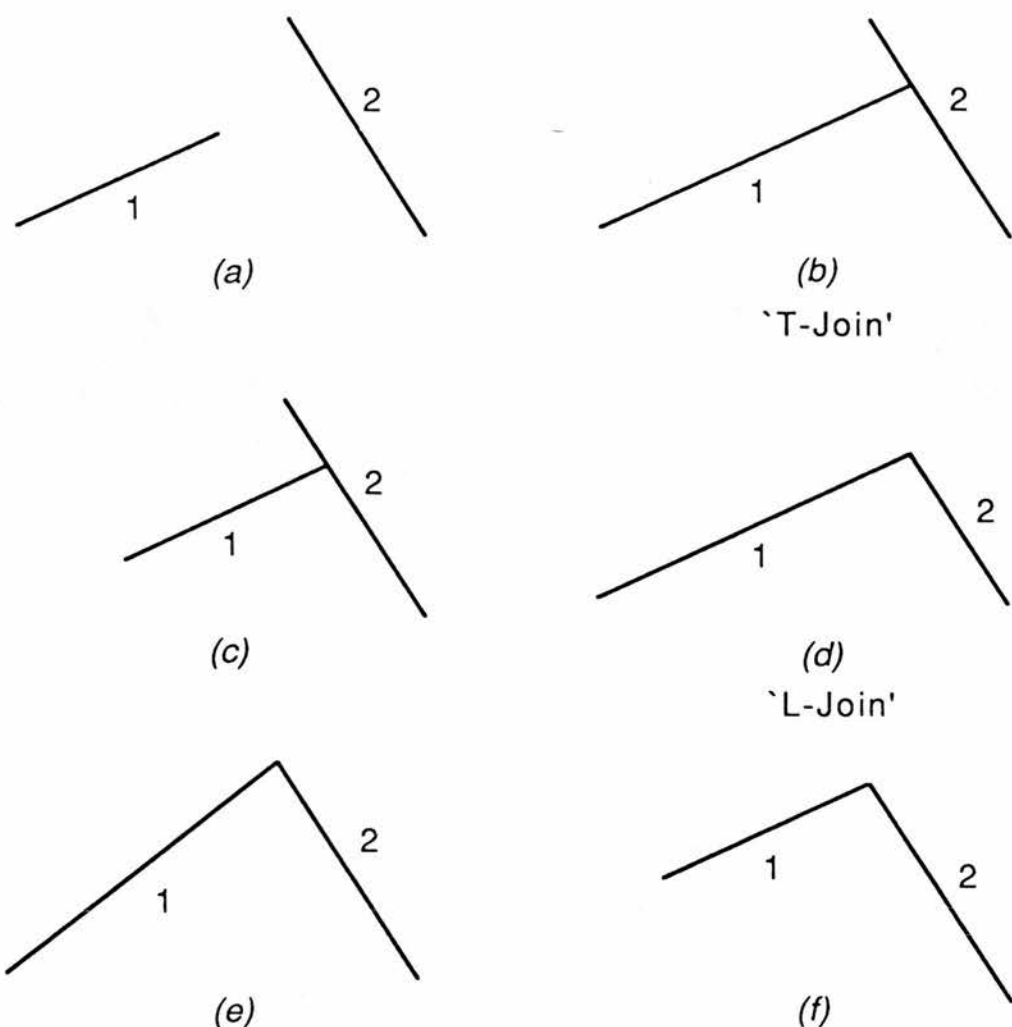


figure 11-10

Consider a drawing such as figure 11-10(a), consisting of two segments. The MacArchitron 'T-Join' function would produce figure 11-10(b), by extending segment 1 to meet segment 2 at the intersection of the implicit 'construction lines' of both segments. As an alternative goal, one might want to join the two segments by preserving the length of segment 1 as well as its angle (figure 11-10(c)). Figure 11-10(d) is an outcome that can be obtained using the MacArchitron 'L-Join' function. Notice how in this case the length of segment 2 is not preserved since this segment is truncated at the intersection point of the two segments. One might also want to fix the opposite endpoint of segment 1, changing both the length and angle of this segment (figure 11-10(e)). There is no reason to suppose that one would not want an 'L-Join' between these two segments such that the lengths of both segments stay fixed as in figure 11-10(f). There are many other possible attachments between these two segments. Although the phenomenon of attachment is one that is

beginning to be acknowledged in conventional systems, it is not yet one that has been given a principled treatment. The only form of logical join that is used in solid geometry modelling systems corresponds to a 'fusion' type of attachment, which, as we have already seen, is only one of several possible logical attachments.

To conclude this section, we give a simple illustration of how the logical operators of the representation environment can support a variety of alternative attachments between shapes which may lead to alternative outcomes during the course of subsequent transformations. The variety of outcomes is not only as a consequence of having a variety of logical operators, but, more importantly, as a consequence of applying them within different contexts in the logical representation of any particular object. It is this variety of possibilities that is absent from the operations that are available in conventional CAD systems. Consider *figure 11-11*.

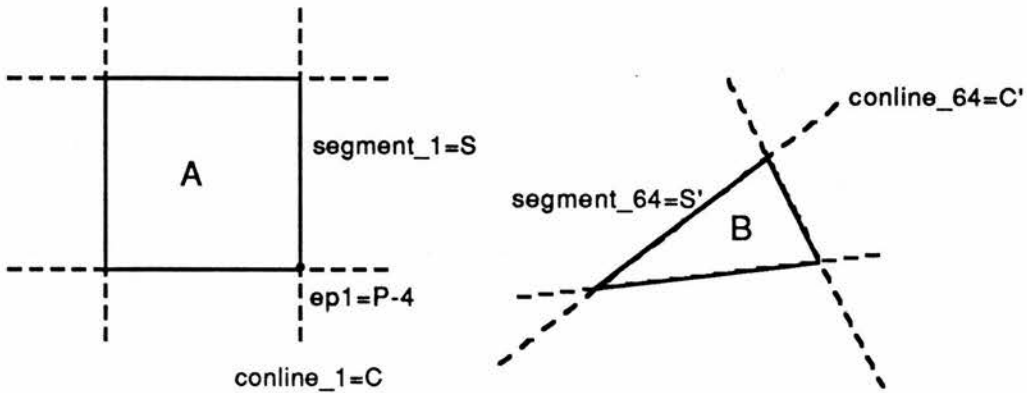


figure 11-11

Examples of attachment operations include the following:

- 1 Conline C' << C.

C' is renamed as C and therefore takes its angle value. By this means shape B can be rotated onto A. Subsequently, if C as a nested part of A is transformed, B's C part will be left behind, since A's C and B's C are two different instances. This kind of transformation can be invoked by by means of a logical expression having the following form:

A:segment_1:ep1:conline_1 <+ [angle=40].

If C's kind name is referenced directly in the transformation expression, then the C part of B will also change, and such a transformation will also affect B, perhaps to the extent that B moves with A, depending upon the logical structure of the remainder of B.

Such a transformation can be invoked by a logical expression of the form:

C <+ [angle=40].

```
2 Conline C' <= [angle=[A:segment_1:ep1:conline_1]].
```

In this case, C' takes the value of C , and B is rotated onto A . Subsequently, this attachment will be preserved under any transformation of C in A , until the indirection filling the angle slot of C' is changed, e.g. by means of:

 $C' \ll C$, or

$C' \prec + [\text{angle}=40]$.

3 $S' \leq S$.

The description of S' is replaced with the conpoints, conlines, and angle values of S. B is rotated, translated, and scaled onto a segment of A. In this case, having used the <= operator, the attachment can be preserved under transformations of A only by directly referencing a kind name below S in the transformation expression, e.g.

P-4:conline_1 <+ [angle=40].

4 $S' \ll S$

As in 1 above, S' no longer exists as a part of B. Note that S now exists as separately identifiable parts of A and B as though the line represented by S is formed of two coincident segments. This property is important to the decomposition of shapes that are made up of separate sub-shapes.

11.8. Summary

This chapter has been predominantly concerned with the nature of logical attachment between graphical objects. The expression of configurational relationships between objects determines the subsequent behaviour of composite objects. In other words, it determines when a particular part moves, to what extent it will pull other parts with it. Different outcomes will arise depending on the exact nature of the attachment relationships between the sub-parts.

The most significant part of the syntax as far as attachment is concerned is the part that deals with drawing *compositions*. A *composition* in itself is nothing other than a

syntactic primitive of the grammar as described in chapter 8, which materialises when a user of the representation environment says he is going to create one. Subsequent interactions may include: naming of objects that will constitute the composition; description of common parts; description of hierarchical relations; description of inheritance relationships. It is these subsequent interactions, each of which can take place within various logical contexts, that determine the final nature of the resultant composition.

Attachment of objects in conventional CAD systems is typically geometrical and not logical. In conventional systems, there is no sense in which joined elements are considered to be parts of the same composite object. In contrast, the system described in this thesis works on the principle that making attachments between shapes is a general means of building up shape information.

12. The Flow of Control in the System

12.1. Introduction

The essential flow of control in the drawing machine as implemented is outlined in *figures 12-1, 12-2, 12-3 and 12-4* respectively, in which the procedures for editing, regeneration, transformations, and attachment are described. Apart from standard graphical operations such as saving the current drawing, plotting the current drawing, etc, these four components are the most complex parts of the drawing machine. The figures themselves make use of Nassi-Schneiderman (NSD) diagrams [Pong and Ng, 1983], which clearly illustrate the logic flow in the system. If then constructs are represented by conditional boxes which redirect control to further boxes depending on whether a condition is satisfied (T) or not (F). This redirection may be preceded by some action associated with the truth condition.

12.2. Editing

The description of graphical objects is effected by means of edits which add to the description of any such object further graphical primitives standing in some relation to existing parts. New instances of primitives which are to become part of a shape or composition are instantiated by particular key-hits as shown in *figure 12-1*.

Consecutive non-coincident conpoint hits instantiate by default a connected chain of segments. Collections of conlines, conpoints and segments form shapes. Collections of shapes form compositions. *Figure 12-1* also indicates the presence of save and plot facilities.

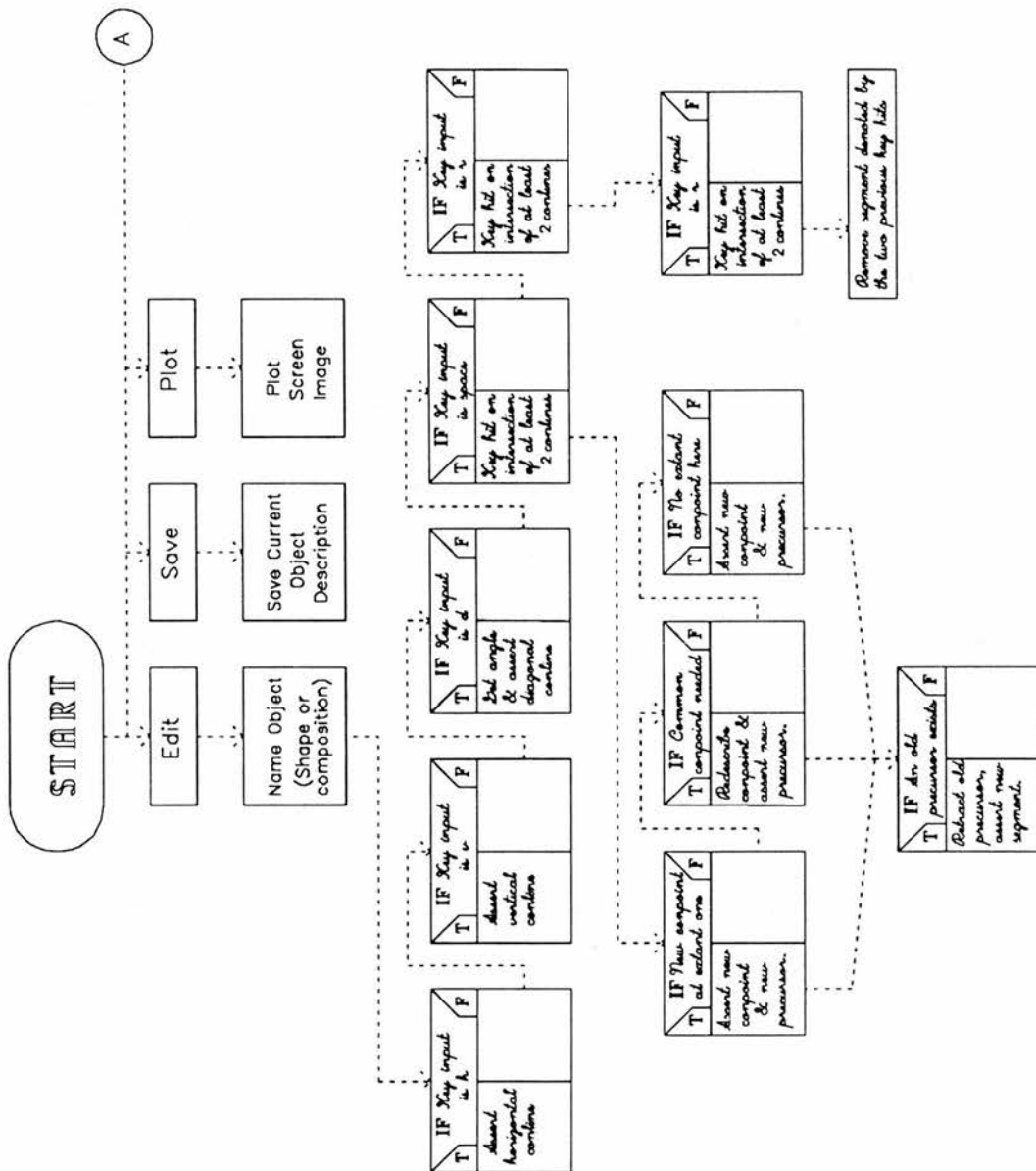


figure 12-1

12.3. Regeneration

The regeneration of shapes and compositions begin by allocating a screen co-ordinate value to some symbolic point (*figure 12-2*). Through this point the drawing machine can then regenerate a conline that passes through it, together with the segment for which the conline is a bearer. The segment's other endpoint provides a new point for the continuation of this process for connected chains of segments.

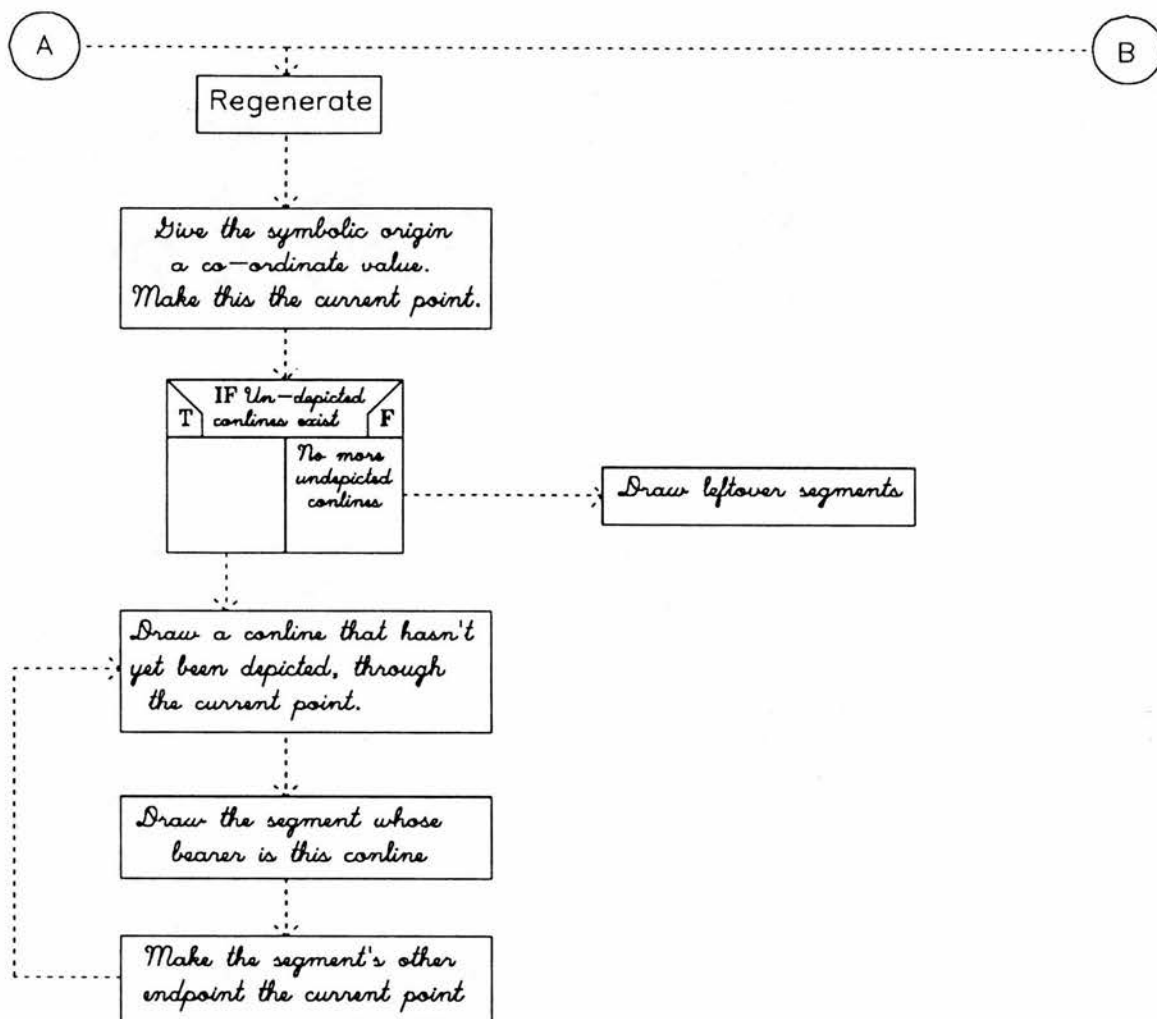


figure 12-2

12.4. Transformations

Transformations are sequences of rotate, scale, and translate operations, each operation being applied locally to a part of a drawing and executed in sequence (*figure 12-3*). The type of connectivity relationships that exist between the part that is being transformed and adjacent parts, will determine the knock-on effects of each transformation.

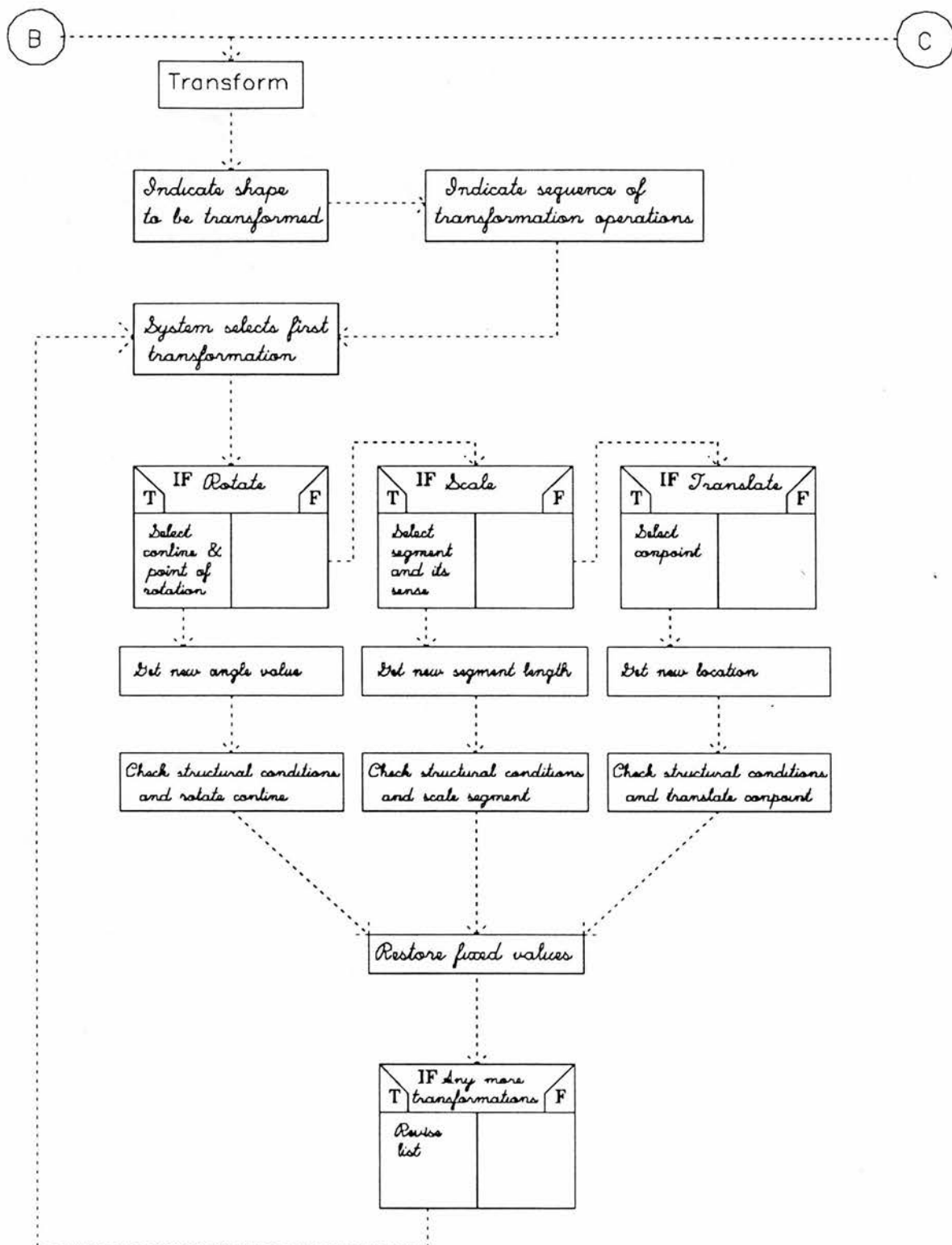


figure 12-3

12.5. Attachment

Expression of conditions upon both location and component objects prior to attachment is essentially menu driven. Conditions include logical sub-part relationships, and form of attachment (aggregation, cohesion, or fusion) - *figure 12-4*. Additionally, the corresponding chains of segments from both objects need to be indicated. The drawing machine establishes the attachment relationships between each segment of the component attachment chain in turn, and its corresponding segment in the location attachment chain. The remainder of the component object is transformed onto the location using the transformations (illustrated in *figure 12-3*).

In addition, *figure 12-5* indicates facilities for the expression of binding relationships, and for the expression of precise angle and length values associated with the actual dimensions of depicted objects.

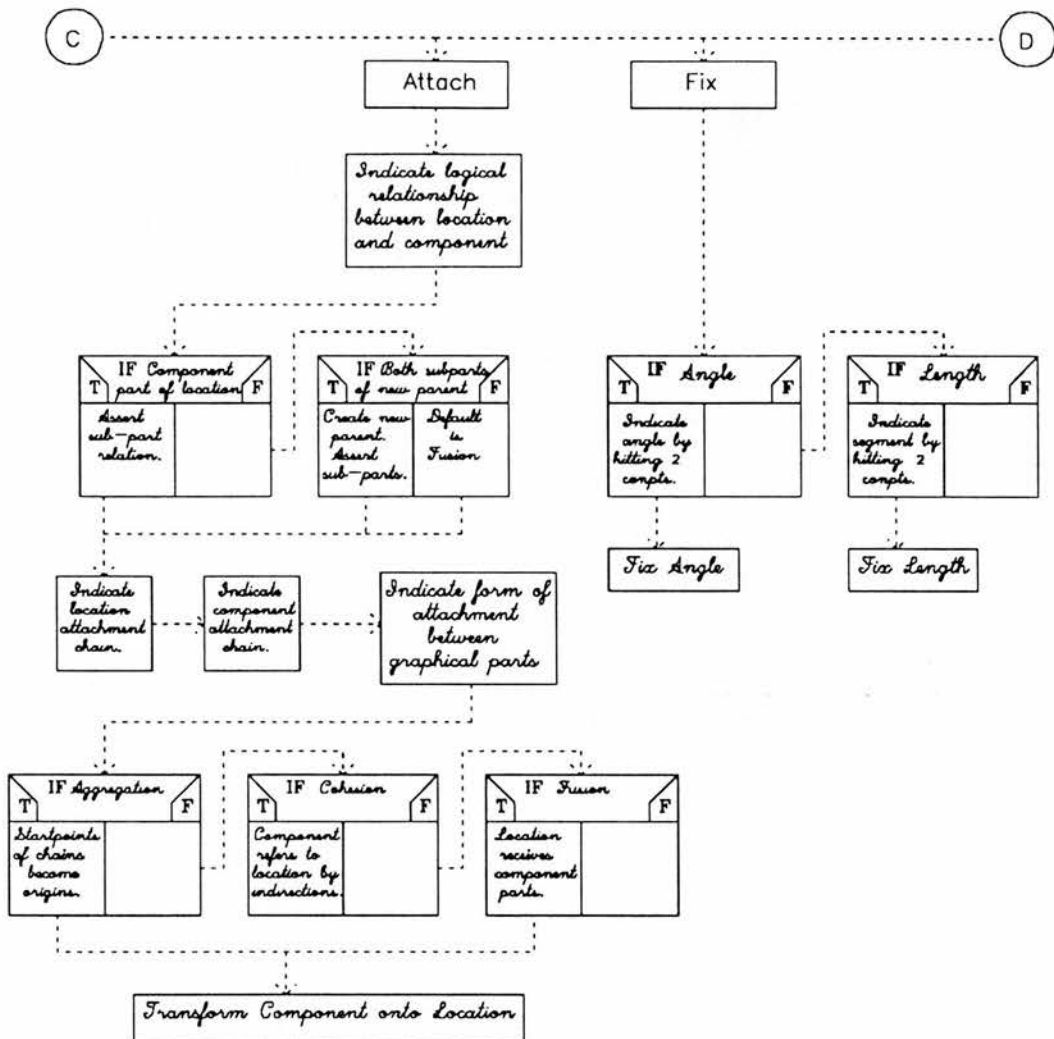


figure 12-4

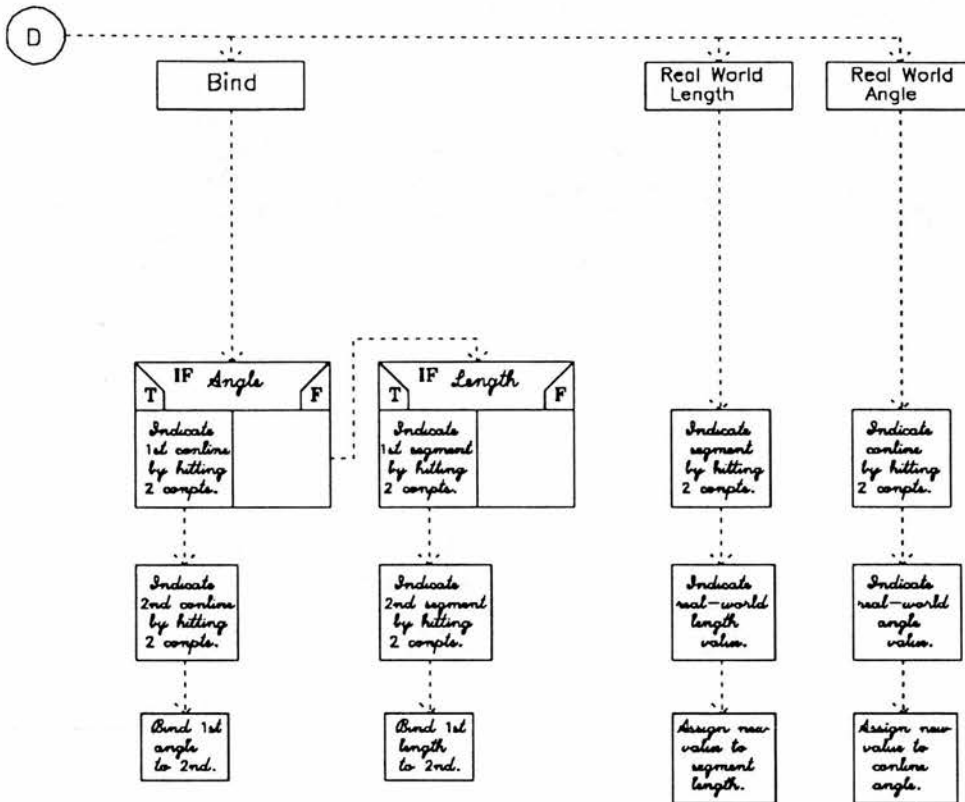


figure 12-5

Part Three

Conclusions, Observations, and Open Problems

13.1. An Instrumental Approach to Design Systems

By way of introduction to the concluding section of this thesis, it is useful to restate the context in which this system fits, namely its use in design. Design is different from other activities that are already amenable to computerised techniques in that it is more a process of problem generation and problem exploration which refers to individual human intuitions and judgements. Design is not subject to separate criteria for correctness of results. Good designs are decided, never proved.

This lack of criteria for correctness compels anyone working on design theory and CAD systems to adopt an instrumentalist approach. One interpretation of instrumentalism [Flew, 1979] is as a term applied to a view about the status of theories held by anti-realist philosophers. According to the instrumentalist view, theories are merely instruments, tools, or calculating devices for deriving some predictions from some data. Consequently, there is no question of such theories being true or false, since theories cannot be either true or false (at least not w.r.t. anything outside the theory). Instrumentalism is thus opposed to most realist theories of science.

The interest in viewing drawing as language is aimed at formulating and interpreting expressions, without looking for a correct language with the right vocabulary for designers. Instead, we have to consider more fundamentally how any language works. Any study of this kind inevitably involves some speculation.

Knowledge is passed between people in the form of externalised expressions, and the things indicated by expressions are further expressions which, ultimately, are interpreted intuitively. This poses a question about the way in which people arrive at and share truths about the objective existence of things and the correctness of objective knowledge. Such truths can exist only as expressions that refer to intuition - and this position applies to both scientific and non-scientific knowledge. Domain knowledge then refers to specialisations of knowledge that people attribute to other people.

It is not our intention to find an explanation of how people read drawings, in order to arrive at procedures that might produce uniquely correct results. The intention has been to discover just enough about how people *make* drawings, in order to define a drawing machine which provides a communication *medium* (chapter 1).

The end to which the means of the representation scheme described is being applied, is to support construction and manipulation of expressions in the form of drawings, where the purpose of the drawings is to communicate object assembly and construction information.

Drawing systems should be as natural as possible in that they should allow users to express changes to drawings, and have the system respond appropriately. It was this goal that has guided the development of the system described in this thesis, even though the success achieved in attempting to satisfy this goal has been limited.

The drawing machine described is intended to produce and manipulate drawing objects constructed from lines in a 2-D drawing space, and to describe states of attachment between lines. It cannot do this without having a particular understanding of geometry/topology. We can regard this domain as being one at which the representation is to be targeted. In other words, such a knowledge domain is an application of the representation scheme. Domain knowledge of this nature is contained in the drawing machine, and should be distinguished from knowledge associated with the application domain which will be referred to as *application-domain* knowledge.

An implication of viewing the drawing machine as a linguistic system, is that one can define the internal structure or syntax of drawing objects, such that these definitions are *independent* of shape descriptions and application-domain knowledge. Shape descriptions can then be defined as parts of drawing objects, so that the drawing system can depict shapes. Shape descriptions themselves will ultimately be context sensitive in the sense that they are conditioned by intentions with respect to application-domain knowledge. Thus representations of shape descriptions and application-domain knowledge will be user-defined, and will vary from person to person.

We have decided upon lines as basic constituents of drawings, defined between end-points. Attachments between lines can vary, affecting what happens when a line is moved. Lines can be translated, rotated and scaled, affecting properties and arrangements of other lines, conditioned by states of attachment (see chapter 11). These are properties of drawing objects, they are not obvious in instances of drawings, they become apparent when drawings are changed, and they have to be controlled by the user of the drawing machine.

Shapes correspond to single connected (open or closed) chains of lines. Angle and length values of lines describe properties of a shape and its sub-shapes. Lines arranged in 2-D drawing space depict angle, length and attachment, as shape properties that can be mapped directly to corresponding shape properties of other things. Interpretations of drawings are *externally determined*, outside drawings and inside the minds of people who produce and read them.

Part One of this thesis was an informal attempt to discover some fundamental issues presented by drawing practice, to inform the more disciplined work of **Part Two**. Here in **Part Three** the salient points of the thesis will be reviewed.

13.2. Relation of Drawing Machine to Representation Scheme

A clear separation was made between a *depiction* and its logical *representation*. Each is an analogy of the other. The distinction is explicit in that one can work in either medium. Conventional CAD systems give the impression of working with the depiction. Carrying out transformations with conventional systems can produce unexpected and unintended results as a consequence of the fact that the user cannot adequately *localise* the effect of transformations by constraining the logical representations of depictions. This is firstly because the representations are not accessible, and secondly because those transformations that do exist have been *prescribed* by a system programmer.

Having logical representations of each part of a drawing allows reference to these parts for the purpose of intended changes. This allows one to design better, if by design you agree with Pye's description [Pye, 1978], viz.

"..... the business of ensuring that at least you get the change you want along with all the others which you don't"

Ensuring the change you want becomes possible if the parts to which you want a change to apply are explicitly represented. The representation environment supports partial description, and is founded on the principle that architectural design description consists of a variety of *aggregative* processes in which discrete parts are brought together to form a representation of some new thing, with parts preserving, to some degree, their individual identities. This can be contrasted with CAD systems which work on the strict principle (often defined in terms of boolean operations) that whenever two or more objects come together, they fuse into a single discrete object such that ancestral objects vanish. This is adequate for domains such as mechanical engineering in which the emphasis is typically on 'sculpting' particular desired components.

Transformations for executing changes provided for users of conventional systems are typically applicable to discrete sub-parts of drawings (commonly referred to in CAD systems as symbols). Consequently, although they can initially be used in such a way as to effect a particular desired change, any further required changes pose severe problems in the unravelling of the description of the discrete sub-part to which a change is being applied.

The drawing machine, therefore, is essentially a *user* of the representation scheme. It can be used both to generate logical representations of graphical objects within the representation scheme, and to generate depictions of such logical representations (*figure 13-1*).

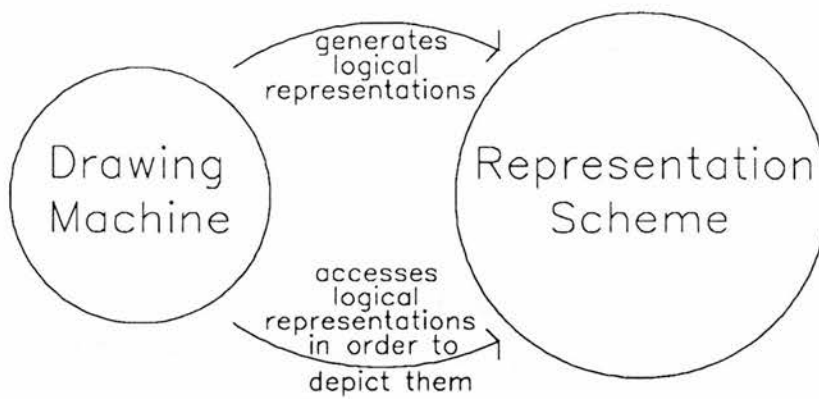


figure 13-1

The drawing machine is defined in terms of primitives and relationships such that instances of graphical objects automatically receive logical representations.

13.3. Depiction-Representation Equivalence

When any system (human or otherwise) looks at a drawing together with its representation, there is a problem of equivalence between these two forms. We need to characterise the nature of the relationship between the user and the drawing machine, and between the user and the representation scheme.

A user interacts with the system in the manner shown in *figure13-2*. The user can either draw things using the drawing machine, or input descriptions directly into the representation environment. The former will indirectly generate logical descriptions of graphical objects in the representation environment. The latter is a means for integrating non-graphical (i.e.textual) descriptions with graphical descriptions, although there are limitations on the form that such integration may take, at least within the context presented by this thesis.

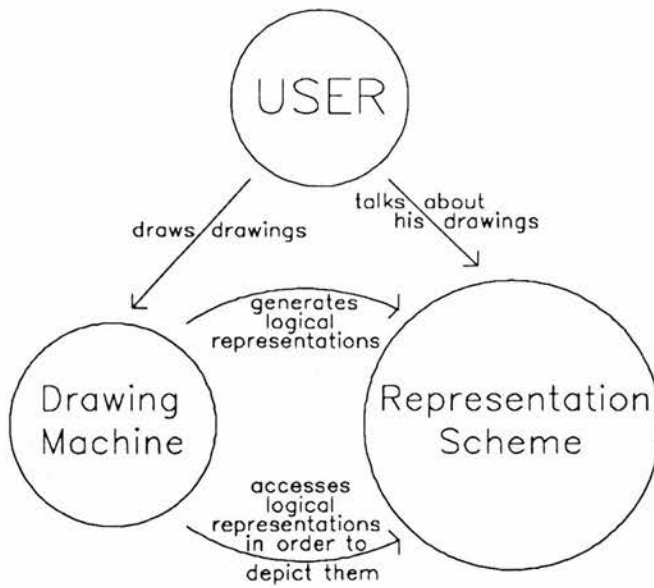


figure 13-2

An illustration of the problem for a user trying to make some sense of the underlying logical representation of a graphical object, occurs when he may want to undo an operation that has been carried out, such as a transformation that may have produced an undesirable outcome. One could argue that there is no need for an undo operation, given that transformations preserve topology. Such a view would imply that a user should be able to recover his original object by applying inverse transformations to the newly depicted graphical object. The user could not, however, recover exact positions easily, since topological equivalence does not imply geometrical equivalence. Attempting to recover the original object via its logical representation implies that the user would have to be aware of logical relationships that existed between graphical parts prior to the initial transformation in order to re-establish them. This he may not always be able to do.

To marry closer the two forms of representation, the depiction and its equivalent logical representation, one can work at *presenting* the representation in relatively transparent ways. An example is the use of the *virtual* operator, by means of which the representation system constructs (from a mess of bottom-level facts) and displays the structure of a kind in terms of slots (paths) and fillers, the presentation being in a hierarchical form.

The strategy adopted in the system described in this thesis has been to allow users to access the representation environment directly in order to be able to say further things about the graphical objects that have been produced. The logical representation scheme

which is the target representation for the representation of drawings, is the same as that used to represent depicted (design) objects i.e. non-spatial aspects. Thus potentially, one can integrate representations of drawing objects with representations of depicted objects.

Users are not allowed, however, to make *destructive* changes to *graphical* parts (i.e. the removal *or* deletion of graphical objects) by accessing the representation environment directly. One can envisage how the ability to do this kind of thing may easily lead to inconsistencies in the logical representation, which the drawing machine has been explicitly designed to avoid. The extent to which one can allow the user of such a system to make destructive graphical changes directly through the representation environment is an outstanding issue which has not been resolved in this thesis. The system described does not have the capacity for such changes, but one can envisage situations in which such a facility would be desirable (cf. the junction example in chapter 9 of [Bijl, 1989]).

One could argue, as Bijl [Bijl, op.cit.] does, that inconsistency should be permissible in a representation environment for the support of evolving designs (i.e. in a plastic representation environment), but from the point of view of a drawing machine, inconsistent logical descriptions of graphical objects may mean that such objects cannot be depicted. More importantly from an implementational point of view, different categories of inconsistency would have to be specified in order for the drawing machine to recognise them without itself falling over when it encounters an inconsistent description. A separation has been maintained, therefore, between those operations executable by the drawing machine which are capable of making destructive changes to logical representations of graphical objects, and those operations executable by the representation environment which have 'read-only' access to such representations.

An important observation is that the modification of drawings does not require knowledge of a storage structure that is separate from that used for design object representations. It becomes possible to refer to any part of a drawing through associated user-described names which form part of the representation of the drawing.

Furthermore, there is a separation of the drawing representation from the physical (and logical) drawing space. The representation of drawings is not dependent upon coordinate point values in some computerised drawing space. The instantiation of depictions, however, is. The drawing machine is responsible for the construction of representations from depictions and vice versa.

13.4. Drawing Compositions

According to Pye:

"Everything everywhere may be regarded as a component of a system. It is fruitless to consider the action of a thing without considering the system of which it is a component. This fact has special importance for designers in various fields because they tend to think of things separately and to design them separately. We ought not to do this if we can help it. We ought at least to remind ourselves that we are concerned with a whole system even if we are only able to effect the design of one component."

[Pye, op.cit.]

Pye proposes an anti-reductionist stance which he observes runs counter to the way designers typically work. The fact that the representation environment only deals with descriptions, allows the user to view some graphical objects as being more 'complete' than others, or alternatively as being part of other objects. In either case, no distinction is made within the representation environment. The top-level object within the drawing machine is the composition. Compositions can be representations of particular details, or 'whole' drawings, or anything in between; in all cases, any composition can be part of some other composition.

The way in which drawing compositions are defined is such that arrangements of graphical primitives (which constitute parts of some logical object) are significant. Compositions are non-discrete in that they typically consist of interrelated shapes, whose parts may also refer to other compositions. A central feature of the drawing machine is its support for the propagation of parametric changes across shapes and drawing compositions. The design of the drawing machine arose out of the observation of the inadequacy of conventional CAD systems in their support for the expression of such relationships. The expression of such relationships allows for the control of transformations such that parts are included in or exempted from them. Transformations, therefore, become controlled distortions. A current trend in computer graphics is to move more towards icon and menu driven systems, in which users can select from a range of prescribed operations, and interact less with the graphical domain to which the icons and menus refer. The ability to be able to access specific graphical parts, to describe relationships between them, to re-describe them (by means of edits, transformations, and post-hoc decomposition) in order to describe new relationships, are activities that are crucial to design. It is advances in this direction that are offered by the work of this thesis.

13.5. The Structure of Design Objects

The structure of design objects rests upon a view of design as being essentially hierarchical. Evidence in support of this view was obtained primarily from looking at drawings such as detail drawings which are obviously hierarchically structured (chapter 4). It is also evident just by talking to designers that design itself can be viewed as being hierarchical. However, within such a view, it is never clear when the description of one object ends and the description of another begins. Consequently, one cannot impose part/whole typing upon design objects. Each object is of equal status.

Although we have adopted the view that architectural design is essentially an aggregative process in which known objects are brought together, the nature of this process may resemble one of several markedly different types. Those types identified in this thesis arise naturally from the representation scheme itself. It would be an interesting exercise to study in more detail the types of combinations that can occur between objects, and how these could in turn be represented in a logical representation environment. Informal studies of forms of relationships between objects was conducted by Soviet constructivists such as Chernikhov [Cooke, 1984]. According to Chernikhov:

"The fundamentals of constructivism consist of all the various possible kinds of unions by which elements can be combined into a structure. Each kind of union is in essence simple, but especially when supplemented by dynamics, they can create complex combinations which amaze us with the refinement and richness of their forms."

Chernikhov identified several kinds of union between 3-D objects:

Penetration - a way of combining bodies where one is inserted into the other.

Embracing - in which one body sits in a loose fashion next to, or around another.

Clamping - is when one body seems to be seized by another which grips it.

Integration - occurs when a single, integral body is given a shape which in itself demonstrates graphically constructive principles.

Mounting - where a series of volumes come together with a single crowning body which unites all the rest.

Interlacing - a synthesis of dynamic and constructive properties.

Of these, 1, 2, 3, and 5, can all be seen as being particular forms of attachment; 4 and 6 on the other hand, are more concerned with particular properties of individual objects, and would have to be defined more precisely before they could be recognised by

any formal system.

Pye [Pye, op.cit.] also mentions categories of change:

".....energy is capable of producing changes, changes in things; more exactly, redistributions of matter. The various ways in which matter can be redistributed are: shifting - that is to say, moving bodily; deforming - bending, stretching, compressing, etc.; dividing - splitting, powdering, cutting, abrading, etc.; joining - welding, fusing, adhesion, etc.; change of state - melting, solidifying, vapourising. But all of them in the last analysis amount to shifting."

Again, some of the above could be interpreted in terms of attachment operations. Others, such as dividing may prove useful for functions such as post-hoc decomposition of object descriptions, and therefore require further investigation. For example, a desirable feature would be to allow lines to be subdivided by other lines without intersections being specified as new end-points.

13.6. Post-Hoc Decomposition

Starting with elements that are combined to form more complex objects is ante-hoc composition. The reverse procedure, of splitting up what you already have into new parts, is post-hoc decomposition. Although it has been claimed that the representation environment is potentially capable of supporting post-hoc decomposition, not much effort has been invested in exploring the significance of this claim, particularly in the context of multiple users of the same system. There is much scope for further work in this direction.

Any object may be decomposed by different people in different ways. For example, an architect and an engineer may divide a wall differently. One will see it as parts of different rooms, the other as part of a support. The extent of these may not be the same. Alternative analyses can be imposed on extant conlines and segments, by pointing out the intended extents. In a representation environment that supports multiple views (as indeed the representation environment of this thesis does), the interactions that take place when the description in one set of terms is altered requires detailed characterisation.

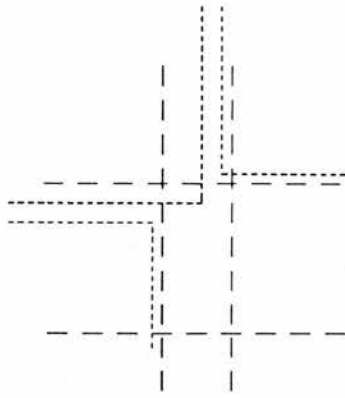


figure 13-3

Consider *figure 13-3*, drawn by an architect on dotted construction lines. An engineer might define the dashed construction lines, and base an object on it. He might then produce *figure 13-4*.

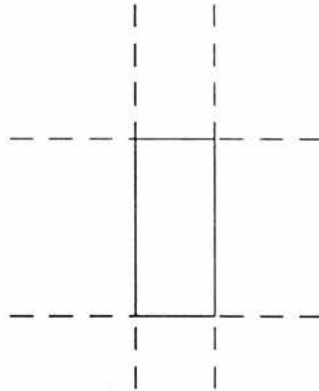


figure 13-4

Where then is the architect's room? It should still be extractable from the new context since the architect's description will still be part of the representation. The architect could have defined some object(s) bounded by construction lines which interacted with the structural engineer's drawing. For example, as in *figure 13-5*.

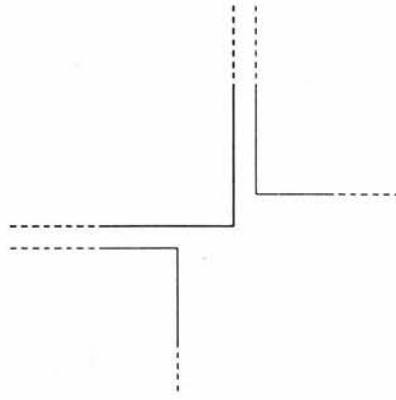


figure 13-5

The problem referred to at the end of chapter 6 involved being able to extract a figure such as 13-7 from 13-6.

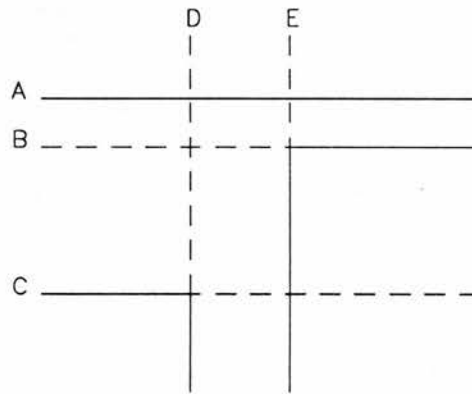


figure 13-6

The solution would be straightforward if one had made *figure 13-6* by assembling walls, so that the construction lines needed were all available, and just had to have their separation increased by a small amount in order that objects moved apart. This is indeed quite feasible with the system described, given that a 'loose' form of attachment existed between the walls in question. But if one had drawn the junction on conlines ABCDE in *figure 13-7*, and the decomposed wall needed conlines XYZ (*figure 13-8*) as well, where do these conlines come from?

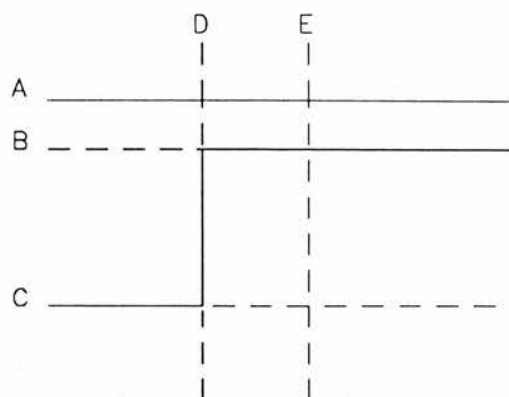


figure 13-7

It will have to be possible to point out construction lines as well as segments, in a way that will have the effect of *duplicating* the construction lines in order to detach parts of drawings. Parts then become available to be moved apart, i.e. X should come from the duplication of B.

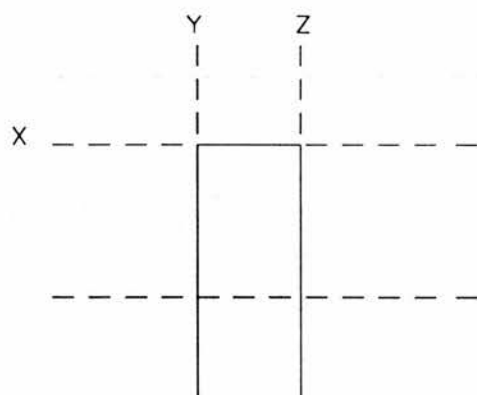


figure 13-8

Looking at drawing objects alone, it is evident that any drawing can receive different decompositions. For example, different people may read into a picture different numbers of lines. Any two objects can refer to the same parameterised shape which, in turn, refers to different possible states of its referent in domain knowledge. The differences are revealed by changes made to the depictions, which reveal the kinds of attachment between the lines in the drawings. The changes are executed by applying transformations to the constituents of drawings. The decision that two drawings refer to the same shape, despite exhibiting

different shape properties, is dependent on human recognition of correspondence to domain knowledge. Results can differ within and between individuals.

13.7. Consistency Maintenance

Consistency, in a general sense, refers to a property of a collection of things, such that all things relate to each other in a manner that is true to the collection, to some 'whole' thing. A body of facts which constitutes the current contents of a knowledge base (in a person or a computer) is said to be consistent if each fact can be related to another fact so that all facts contribute to a single and unambiguous state of knowledge. For formal systems, the necessary pattern of facts is determined by logical truth values. A state of knowledge that is to be supported by the consistency of its constituent facts, has to be specified from outside the system and must be communicated to the system, as a specification of goal. Consistency then refers to facts being true to some goal.

13.7.1. Recognising Consistency

Consistency maintenance is a widely recognised problem [Hofstadter,1979]. It is commonly categorised into separate issues of internal consistency (with respect to internal functions of anything exhibiting external behaviour) and external consistency (with respect to the acceptability of such behaviour to some other thing, or person). As Hofstadter [op.cit.] points out, this distinction is itself problematical, relying on clear demarcations between things, and people. A commonly observed phenomenon is that something, such as a computer, can be internally consistent and yet behave in a manner that a person regards as inconsistent.

Basically, for any body of facts to be consistent, it has to exist in a form that is acceptable to the representation scheme employed by the thing or person in which the facts occur, or to which the facts are presented. This suggests that any parties exchanging facts need to be employing the same representation scheme. Indeed, this seems to be so in many cases where people use computers - the person has to use the representation scheme of the computer (this condition being ensured by a good and perhaps 'user-friendly' user interface), and the person might be able to employ results in further representation schemes that are not available to the computer.

13.7.2. Consistency As Contradiction

A common characterisation of inconsistency (whether internal or external) is to say that facts logically *contradict* other facts, and to say that facts have ambiguous relationships with other facts - those facts are said to be not true, or without meaning [Earle, 1973; Frost,1986]. We need to establish what is meant by contradiction and by ambiguity, and to establish whether the same meaning can be applied equally to different representation schemes used by computers and people. The problem is reduced if it is claimed that the meaning needs to hold only for computable representation schemes, leaving it to people to resolve their own further difficulties if they employ any further representation schemes.

Inconsistency in the form of a contradiction can mean that a body of facts includes one or more facts that cannot be resolved with other facts as known in a representation scheme. The effect of a contradiction is to produce an unacceptable body of facts within a representation. The resolution of a contradiction may necessitate the rejection of a particular fact.

13.7.3. Application of Internal Consistency Checking to Design

As a design develops, and facts are accumulated in a representation environment, one could attempt to ensure that those facts were internally consistent in order to preserve the coherence of further interactions. Further facts added, either explicitly by a user, or as a consequence of invoking system functions, could be checked against all existing facts to establish whether any constraint violations and inconsistencies have been introduced into the design description.

Internal inconsistencies could arise both as a consequence of establishing contradictory relationships between parts, and also at a more basic level when facts, not necessarily related by dependency relations, are introduced into the representation environment. In the system described in this thesis, this problem is potentially most serious in the context of non-graphical updates to the representation environment, where they can potentially contradict those logical representations produced by the drawing machine (this problem can also occur in the reverse direction i.e. drawing machine changes to non-graphical representations). This form of update (i.e. across different forms of expression) was not allowed in the implementation described in this thesis, and consequently greatly reduced the amount of internal consistency checking required. However, there are situations in which one could envisage such changes as being essential [Bijl,1989]. In this case some form of internal consistency checking would also be essential.

From the illustration of the problems associated with sketching in chapter 3, it was observed that in the case of calculable dimensions (i.e. in the case of dimensions that can be propagated from other explicit dimensions), one makes changes to explicitly dimensioned parts, and expects to see calculable dimensions calculated. One could expect that only explicit dimensions should be alterable, and that it should be impossible to change dimensions that can be calculated. However, one can envisage situations in which the required calculations might be very difficult, in which case this requirement might be hard to enforce. Again, by means of non-graphical updates rather than via the drawing machine, suppose that we have constructed a logical representation which one might consider to be sufficient to generate a figure such as *figure 13-9*. Suppose that the angles of all the conlines are known explicitly, but only the length of *one* of the segments. (If the logical representation of such an object had been produced graphically, then there would be length values for all the segments.) There are geometrical techniques which would allow one to determine the lengths of all the remaining segments, and potentially, therefore, to generate the graphical depiction of the whole object. The calculations to do this, however, would be awkward. To find if a logical representation so generated produces geometric inconsistency, one would have to calculate dimensions in all possible ways. Abandoning such checks could produce logical representations that were geometrically inconsistent.

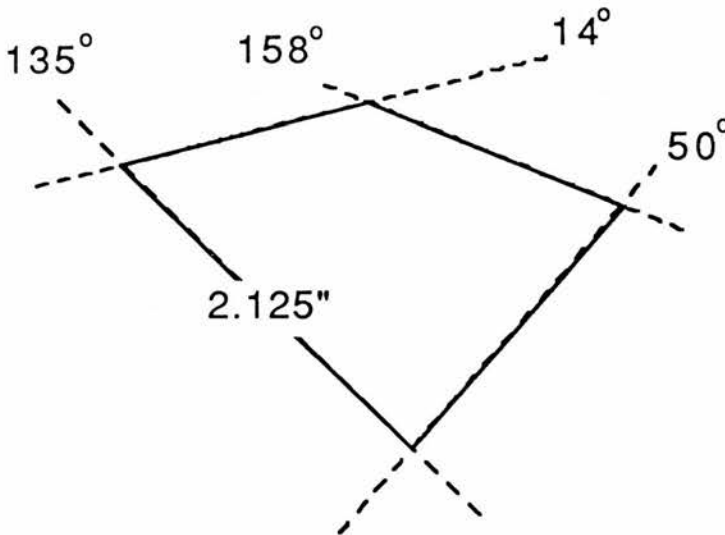


figure 13-9

An internal consistency checker might be able to find sets of facts that were not cotenable, perhaps by looking at all facts in all possible ways, and then informing the user about which facts were inconsistent. A standard way of checking for internal inconsistencies and redundancies would be, for each fact in the representation environment, to first delete that fact, then check whether it's negation is derivable from the remaining facts [Frost, op.cit]. If it is, then the representation environment is inconsistent. If the fact

itself is derivable from the remaining facts, then the representation environment is redundant. Otherwise you have an independent fact.

The system described in this thesis provides a mechanism that reduces the problems of internal consistency, namely, the creation of new instances of objects, which, as far as the system is concerned are unique and separate. Given that there will be multiple views, we do not take the view that they should all be views on a single central model. Rather, they all have equal status and may well be inconsistent with each other. Demanding consistency of a central model system would involve the arbitration of incompatible updates; and no system can do that. In a central model system, if one user says that the roof material is tile, and another user system says that the roof material is slate, such a system would have to enforce agreement before the second user could carry on with his own variant. The representation environment described in this thesis can be configured such that whenever a new user begins to use the system, his top-level kind corresponds to himself as a user. This would distinguish the objects that this user is working on from the objects that some other user is working on. Any user can begin work by starting with another's definition of a kind, and creating his own variant of it. Changes to another user's kind are prevented as a consequence of the context in which it sits. A whole design project can contain several views, one per person, or function. Each view has access to the whole machinery of the representation environment. An implementational simplification is that each user can only make read-only indirections to other views. Indirections being read-only is justified on the grounds that the system shouldn't solve the problem of different people wanting different designs - the users themselves should do that. In other words, there is no central model. Each view is itself a model, which may use bits of other people's views (circular reference might be a problem). Different people might also be responsible for the fillers of different slots.

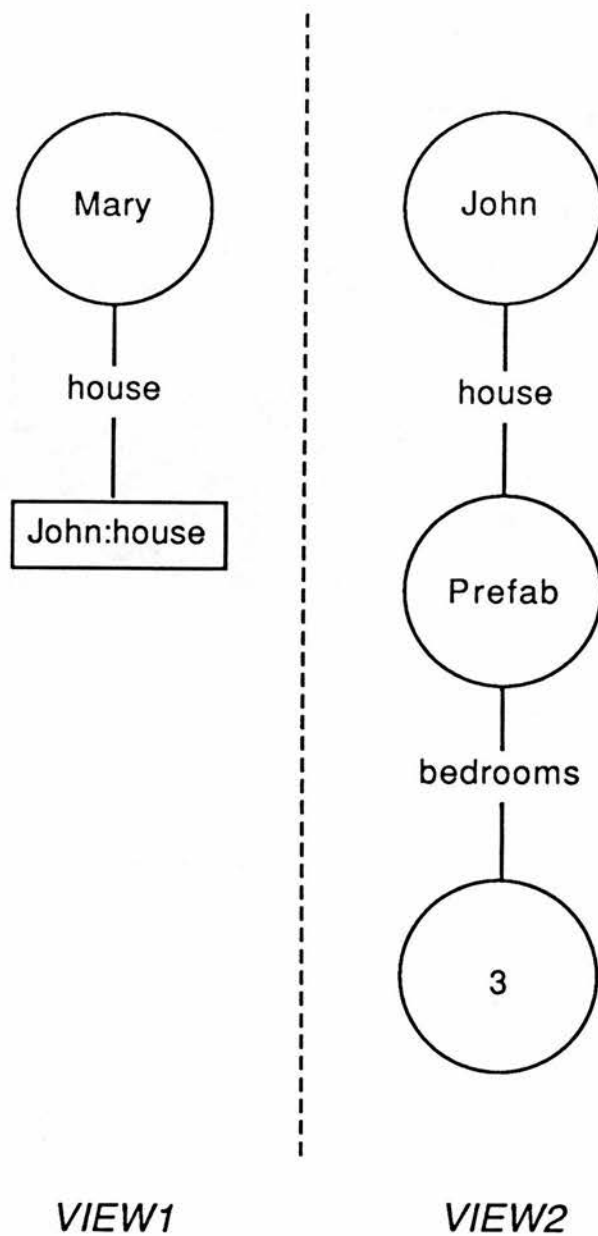
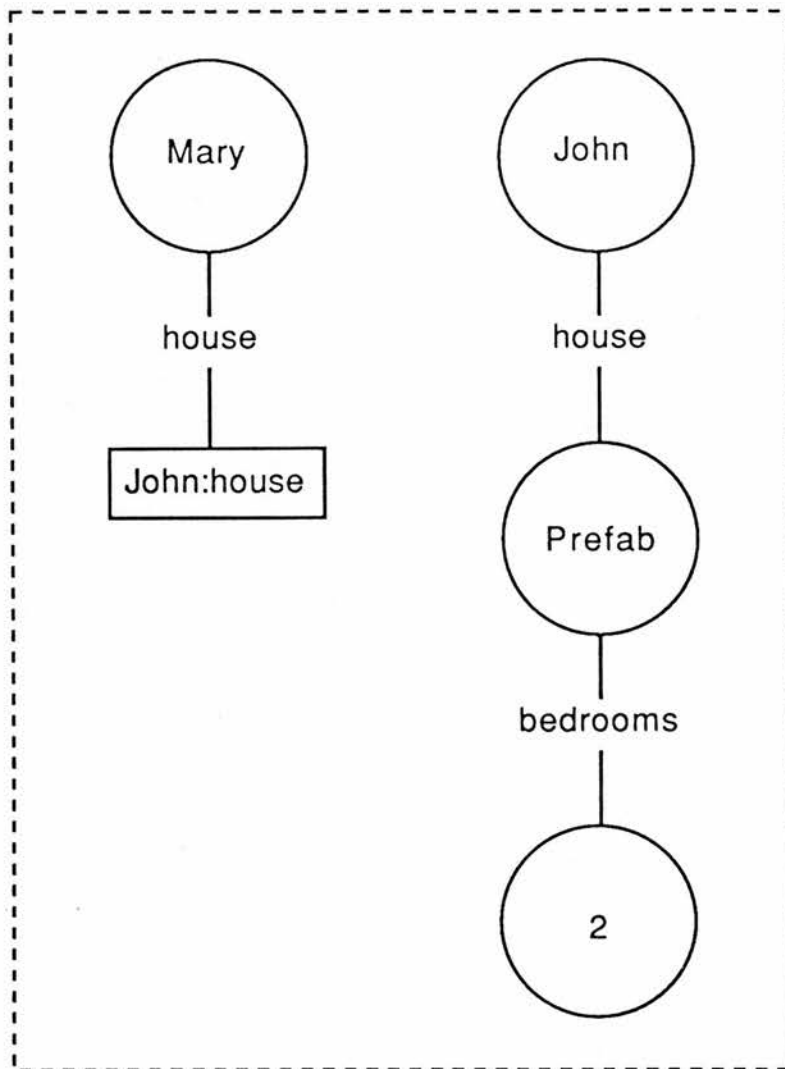


figure 13-10(a)

Imagine the set-up in *figure 13-10(a)*. Suppose Mary makes the update

Mary:house <+ [bedrooms = 2]



A SINGLE VIEW

figure 13-10(b)

If Mary's and John's descriptions were part of the same view, then one would expect the result shown in *figure 13-10(b)*. But updates of someone else's view are not allowed. Alternatively, one should have an equivalent change, inside the changer's own domain, as in *figure 13-10(c)*, such that a new instance of the kind "John" is created, as well as new instances of kinds along the path to the kind that is being changed.

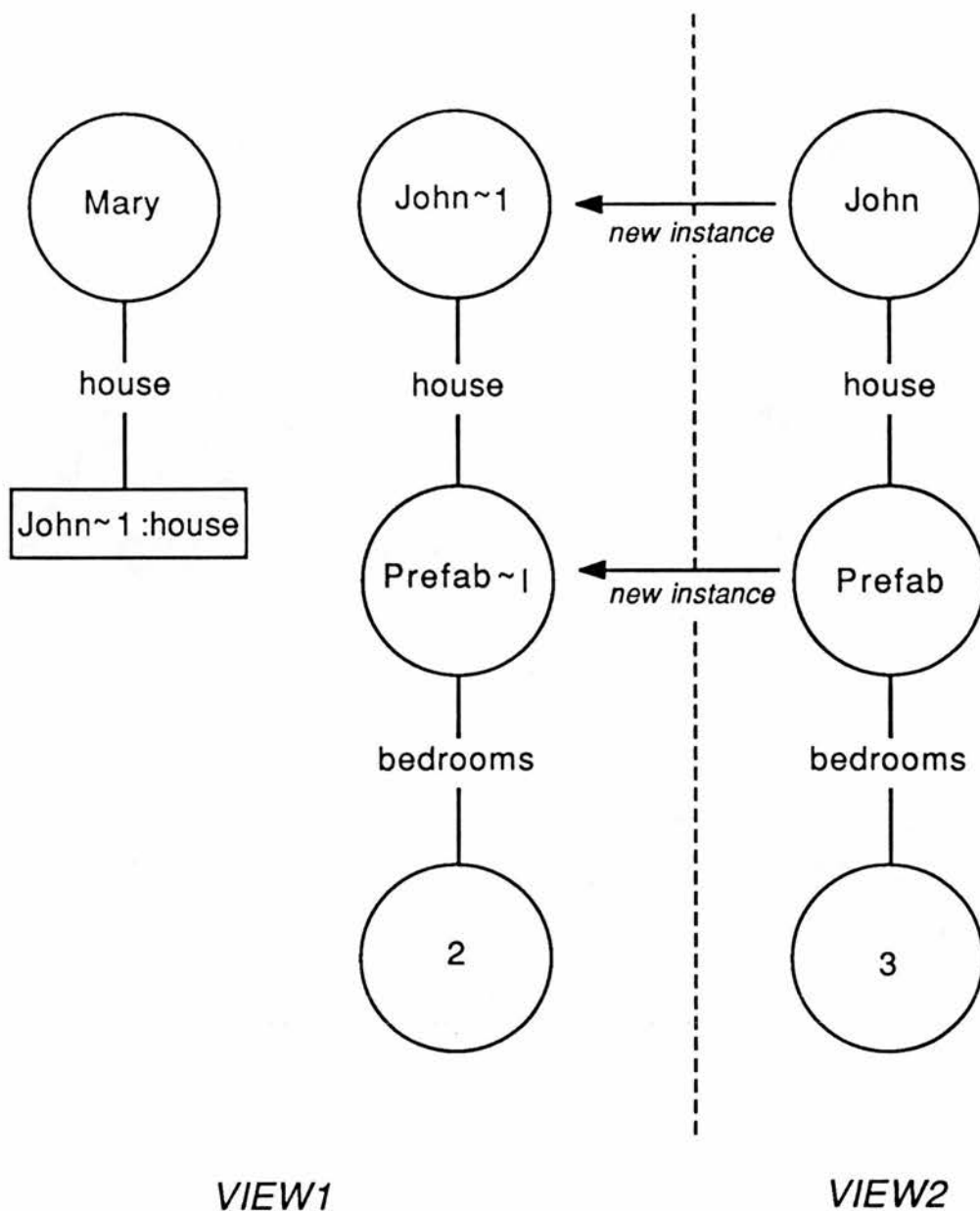


figure 13-10(c)

13.7.4. The Problems of Internal Consistency Checking in Design

The above account is inadequate in the context of design activity. The progression of a design in some ways runs counter to conventional logic programming concepts. As well as checking for the truth of something, or whether some action is possible (such as the system being able to depict an object), there is also a notion of *forcing* a fact to be true. This is a form of update. One can introduce a fact that one wants to be true into the representation environment at a point when the introduction of this fact would make the representation environment inconsistent. One wants to be able to *continue* adding facts into the representation environment until the fact that was introduced is no longer inconsistent with the rest of the representation environment. One could envisage this kind of

circumstance arising when non-graphical updates are being made to the logical representation of an object that the system itself knew to be graphically depictable, perhaps to the extent that further functions depended on this being the case. Such changes may result in the temporary removal of certain properties from the logical description of this object, which would then not allow this object to be depicted. However, by making additional non-graphical updates, one might subsequently be able to depict this object, and arrive at the desired outcome. This observation leads one to suspect that internal consistency maintenance should only be carried out when required, and not at every possible opportunity, such as after every update.

Clearly one can envisage situations in which one might want some form of internal consistency maintenance to be carried out in a CAD system. We have already observed, however, the close relationship that exists between internal and external consistency. In the context of design activity, where a designer is using a computer, the criteria for recognising a contradiction ultimately has to rest with the designer. Those criteria are not likely to be externalised in some prior and objective form - indeed some criteria cannot be externalised. This position has the positive consequences that any contradiction can be regarded as temporal and resolvable. Although one can build internal consistency checkers directed at particular tasks or goals, it should be evident that the design context provides a particular application domain in which external consistency is paramount.

13.7.5. External Consistency

Inconsistencies can exist between things represented in a representation environment and things external to the representation environment that they are intended to represent. Suppose we are thinking of some material which we know has a reflectance value of 0.6 units, but we wish to represent this material in the representation environment according to information about this material's properties obtained from available textbooks. This material might be represented by a kind "MATERIAL". We may first have added a slot to this kind called "brightness" filled by a value of "0.5" units. At some subsequent stage, we add a slot "albedo" to this kind with value "0.7". At this point, as far as the representation environment is concerned, there is no internal contradiction. But if we discover that what we knew of as reflectance was synonymous with brightness, then the *external interpretation* of the representation becomes inconsistent. We could replace the slot "brightness" by "reflectance" thus removing this inconsistency. If we later find out that reflectance is synonymous with albedo and replace "albedo" by "reflectance", then the representation environment itself would be internally inconsistent, since it would now have two slots with the same name, but with different values. In this case, no object that we

have in mind could be represented by the kind "MATERIAL". Which attributes an object has, therefore, are as vital to its identity as the values of these attributes. The same phenomenon can occur between different views. There could be two objects, in different views, both intended to represent the same material. They could have the same or different names. They could have identically or differently named slots, to represent reflectance, and those slots could have different fillers. Inconsistencies of this nature can only ever be expressed intersubjectively, and not objectively. The owners of the views may have notions of what parts of their view they wish to equate, and this can always lead to inconsistency.

The problem of locating a contradiction is usually resolved by some formulation of external reference, meaning that facts can be verified by the actual presence of objects externally to representations of objects within computers and people [Wittgenstein,1922]. This line of argument does not hold for design objects, or for certain properties of design objects. The basic position for design (as described in chapter 5) is that design objects exist as propositions *prior to* the existence of their corresponding designed artefacts. Furthermore, design objects are not wholly describable by reference to parts that will be found in already existing concrete or artefactual objects - designing is more than a process of searching among candidate solutions (c.f. chapter 5). A contradiction, then, has to be considered as occurring between parties that are employing different representations, and it can be resolved only by one or more parties accepting another's representation and amending their own. When the parties are a person and a computer, intrinsically it is only the person who can adapt her or his own representation.

Designers using a computer have to know and work with the representation scheme that defines the computer system, just as they have to when they use any other artefactual tool. The design of the tool must be visibly consistent with designers' views of its functionality, as in the case of a system for producing drawings. The idea that a computer system can serve as a tool that behaves in a manner consistent with different designers' views of its functionality, is problematical. This prompts a strategy for system design which focuses on minimum functionality, presented in a manner which designers can exploit in their own more complex and varied tasks.

This is the strategy that has been adopted for this thesis. The goal has been an abstract and general structure for drawings, based on line primitives and their properties and relationships, plus edits and transformations to chains of lines. The system's functionality is minimal in that it is targeted at the production of lines to form drawings, without anticipating their meaning in terms of higher-level structures.

Consistency in such use of the drawing system to describe other things, has to be the

responsibility of the users of the system. The system designer can be responsible only for the consistency of the abstract and general structure for drawings, taking due (but informal) regard of how other designers use drawings. It should be noted that the work presented in this thesis focuses on the abstract logical structure of drawings, as a necessary precursor to any future implementation to make that structure visible and accessible to users. The drawing structure that has been presented has to be judged in terms of its potential to support the varied (and idiosyncratic) depictions that designers draw, which depict varied and unforeseen knowledge of designers (refer to examples). The structure is intended as a basis for a general tool for designers, by which they can link their drawings to other logical representations of design objects. Consistency maintenance, within the drawing structure itself, is targeted primarily at the general drawing transformations and as such, is concerned with the satisfaction of structural conditions upon logical representations prior to the invocation of transformations (see chapter 10).

It is evident from the above discussion that design activity invariably introduces consistency problems. Consistency maintenance seems to be a natural consequence of the exploratory nature of the design process, where different possibilities, and different views will be compared. The relevance of AI techniques for dealing with this type of problem (Truth Maintenance techniques - and Assumption-based Truth Maintenance Systems in particular [deKleer,1984]), based as they are upon consistency as logical contradiction, is unclear. Providing consistency maintenance support to teams of designers working together on a large design project is as yet an unsolved problem requiring further research.

13.8. Drawing as Language

Viewing drawing as language should be differentiated from drawing as linguistics. The view of drawing as language presented in this thesis can be interpreted on two levels. Firstly, as an analogy, in which generalised linguistic concepts can be seen to relate to the activity of drawing; and secondly, at a theoretical level, in which linguistic techniques are applied.

The extent to which one can regard the grammar used in this thesis as being transformational is best described by a general definition given by Lyons:

"The term 'transformational' has unfortunately engendered a good deal of unnecessary controversy and confusion in the recent literature of linguistics. If we use the term in a general and rather informal sense, rather than in the particular sense in which it is defined in any one theory, we can say, quite reasonably, that the 'deeper connexions' between sentences which 'cut across the surface grammar' are transformational relationships: this is a perfectly legitimate use of the term 'transformational'.

[Lyons, 1968]

The 'deeper connexions' are the topological relationships between parts of drawings. These 'cut across the surface grammar' in the sense that such relationships are formed from the way in which drawings are constructed, rather than from how they appear as depictions.

Transformational components of grammars in linguistics are driven by the premiss of the preservation of meaning between sentences whose surface structures appear to be different [Radford, 1981]. The equivalent for drawings is topology. One can say that in the system presented in this thesis, the semantics of a graphical object is just its topology. Semantics as topology is valuable in architectural CAD systems since attachment (connectivity) between parts is vitally important, both from a designer's conceptual view of a design, and from the point of view of being able to express relationships between parts of drawings. These two aspects are more closely related than might first appear, since how a designer conceptually thinks of objects will affect his expectations of how 'plastic' representations of them ought to behave when properties are changed (cf. the early examples given in chapter 11 of bedroom/sitting room and bath/bathroom). Lyons goes on to say:

"Any grammar that claims to assign to each sentence that it generates both a deep-structure and a surface-structure analysis and systematically to relate the two analyses is a transformational grammar (whether it uses the label or not)."

[Lyons, op.cit.]

One can regard the logical representation of the topological relationships between parts of drawings as constituting the deep-structure analysis. A depiction is a surface-structure analysis. Representations and depictions are systematically related through the transformational grammar of the drawing machine.

No attempt has been made to conduct a detailed and formal comparative study between the particular transformational grammar presented, and transformational grammars as they exist in linguistics. As has already been stated at the beginning of this chapter, the philosophical strategy adopted was an instrumental one. A transformational approach was adopted because of the fact that it seemed to arise naturally as a direct consequence of the logical representation environment. In so far as one would like to have at least a brief comparison between the two forms of transformational grammar, a brief description of the similarities between the transformational grammar for graphics (GTG) of this thesis, and transformational grammar (TG) in linguistics will be given here. It has to be stressed that this is only a rough, and not exact comparison. The following observations can be made:

- (i) Use of the notion of constituent structure ('bracketing' in TG; indirection in GTG) to establish well-formed objects (sentences/drawings).
- (ii) *Diversification* exists in both TG and GTG; i.e. two objects may differ in surface-structure, but be identical in their deep-structure. For example, a graphical object A, and a translated copy of it A' have the same logical representation structure.
- (iii) *Neutralisation* exists in both TG and GTG; i.e. two objects having different deep structures may produce identical surface structures. For example, a composite object produced by the attachment of two objects such that the two objects are sub-parts of the composite object, can look identical to a single discrete shape that was drawn during the course of a single edit.
- (iv) A distinction between *generalised* (or 'double-base') transformations and *singular* (or 'single-base') transformations exists both in TG and GTG. Singular transformations in GTG are those which apply to an individual object or some particular part of an object. They are the transformations of rotation, translation, and scaling defined in chapter 10. Generalised transformations in GTG are those involving the combination of two or more objects through the process of attachment, described in chapter 11.
- (v) Just as in TG, generalised transformations are *optional*, and singular transformations can be either optional or *obligatory*.

13.9. Modularity of the System Components

The representation environment and the drawing system, together constitute a system in which each of these components work relatively independently of each other.

The representation environment could equally well be used to satisfy other ambitions within other research areas. It is quite probable, therefore, that in these other cases, it would fit into an alternative larger system - a natural language system for example. The point is that each of the system components is, to a certain extent, modular. However, they are also inter-dependent. In this respect, each of the components contributes towards system integration.

It would be interesting to explore to what extent the representation environment would be capable of supporting drawing systems, natural language systems, vision systems, etc. Some of the larger systems so formed would be more efficient than others if one views the representation environment as being part of some larger system. It may turn out that

the representation environment is only appropriate to a particular problem domain. Consequently, the larger structure containing the representation environment that has been outlined so far, together with some drawing system such as the one described so far, would inevitably be more efficient than other larger systems since the intent was to construct a representation environment for the specific purpose of representing drawings. In other words, to construct a *natural* representation for this domain.

13.10. Extension to 3-D

One can envisage how the 2-D line drawing system presented in this thesis might be extended to take into account graphical depictions of 3-D objects. There is a natural progression into 3-D which preserves a mode of working using lines in 2-D, if one assumes that the graphical domain is one consisting of 2-D *projections* (of whatever kind - isometric, for example) of *laminae* in 3-D space.

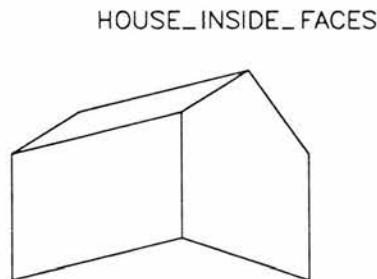


figure 13-11

Consider *figure 13-11* for example. This is essentially a 2-D drawing which can be associated in the representation environment with a named kind, such as HOUSE_INSIDE_FACES. This in turn can be part of the description of a whole house such as that depicted in *figure 13-12*. In such a system, a *closed* connected chain of lines identifies a lamina, which, when attached to other closed chains of lines, form depictions of laminar objects, which, if closed, can constitute depictions of polyhedral objects. One would have to assume that the base generation component in a 3-D system would support the construction of connected chains of laminae. A lamina would in such a system be an additional primitive object.

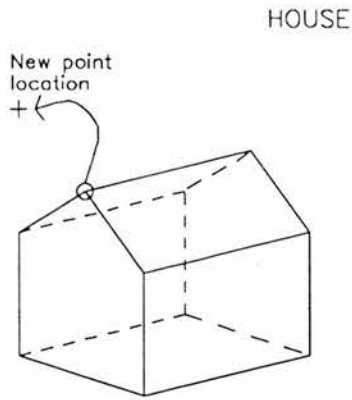


figure 13-12

A user can interact with the 2-D depiction in a way analogous to the strategy for interaction proposed in this thesis, namely, by means of applying transformations (rotate conline, scale segment, translate conpoint) to local parts. The *effect* of such transformations, whilst still preserving topology, would be different in this domain, in that they would additionally take into account the type of projection that is being modelled. A method for the extraction of planes from depictions of transparent objects and wire models, involving the interpretation of 'decussate' vertices, has been described by Szalapaj [Szalapaj, 1980].

In *figure 13-12*, the translation of a conpoint in the description of the HOUSE_INSIDE_FACES object has been indicated. This point is topologically connected to the other parts of the HOUSE description, causing the transformation indicated to have an effect such as that shown in *figure 13-13*. Notice that a prerequisite for the application of such a transformation would be the introduction of the new lamina PQR formed as a consequence of adding the line PQ. One could maintain a distinction between edits and transformations of laminae, just as was done in the domain of 2-D line drawings. Edits would be the addition/deletion of lamina, and transformations, topology-preserving property changes.

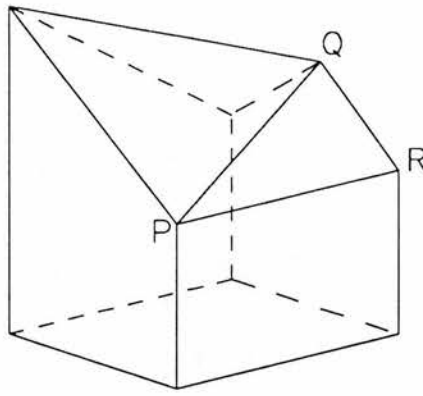


figure 13-13

One can see how attachment of 3-D laminar objects can be controlled through interactions with the graphical primitives of their 2-D depictions. For example, *figure 13-14* is a depiction of a composite object consisting of two logical parts which represent two attached polyhedra.

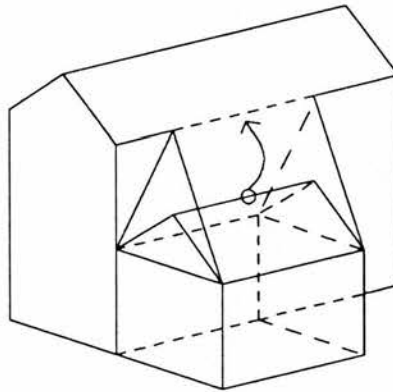


figure 13-14

Suppose a further attachment relationship needs to be expressed in which the logical object RIDGE of the lower part is intended to coincide with the EAVES of the upper part. This could be achieved by means of translating a segment (indicated by an attachment chain consisting of two conpoints) onto another segment, with a possible outcome such as that shown in *figure 13-15*.

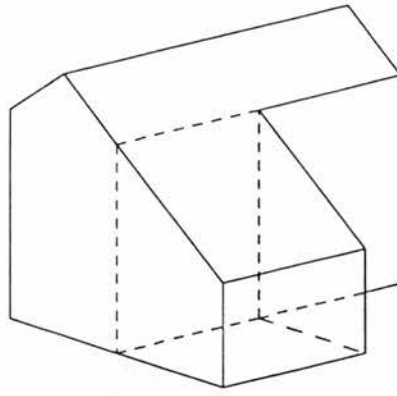


figure 13-15

13.11. Conclusion

During the course of the thesis, the objectives described in the preface were largely adhered to. A certain amount of prescription was inevitably introduced by restricting the system to work only with 2-D line drawings, although from the brief account given in this chapter, an extension of the system to work in 3-D should in principle not be too difficult.

A leitmotiv of the thesis was the consideration of graphical elements and drawing compositions as logical objects. The basic graphical objects have properties such as length and angle which can be used to establish relationships between parts of a drawing. Point coordinates do not play a role in the *logical* representation of an object since they do not contribute to the expression of such relationships.

Certain operations provided by the system allow the user to express more complex relationships (e.g. parameterisation) between parts of objects. Such relationships have significance for the user in that he would like them to be maintained during the course of further changes to the drawing. The effect of having expressed such relationships is that changes to any elements within the relationship will be propagated to the other elements. This form of logical consistency checking arises naturally from the choice of knowledge representation, and is far more powerful than locating objects by means of absolute geometrical position as in most conventional systems.

Transformations upon graphical objects differ fundamentally from conventional transformations in that the user can effect partial transformations, and is not bound by a prior commitment to the wholeness of parts. This is done by including/exempting parts of graphical objects in/from a transformation thus producing *controlled distortions*.

The knowledge representation itself allows for the description of objects in a hierarchy with the expression of relationships between parts within it. Such a representation has some psychological basis in that it seems to reflect the way in which architects think of objects. It also has affinities with some of the current developments of object-oriented environments in Artificial Intelligence.

REFERENCES

- Alexandroff, P. (1961) *Elementary Concepts of Topology*. New York: DOVER Publications, Inc..
- Amarel, Saul (1980) Initial Thoughts on Characterization of Expert Systems. Technical Report No. CBM-TM-88, Computer Science, Rutgers University, New Brunswick, N.J., September, 1980.
- Amarel, Saul (1986) Problem Solving. *Encyclopedia of Artificial Intelligence*.
- Anjewierden, Anjo (1986) An Overview of PCE-Prolog. . University of Amsterdam, Amsterdam.
- Apple Computer, Inc. (1986) MacDraft V1.2a. Technical Report, Cupertino, CA, 1986.
- Apple Computer, Inc. (1984) MacDraw V1.9.5. Technical Report, Cupertino, CA, 1984.
- Autodesk, Inc. (1987) Autocad Release 9.0 Reference Manual. Technical Report, Sausalito, CA, 1987.
- Bach, Emmon (1974) *Syntactic Theory*. London: HOLT, RINEHART and WINSTON, Inc..
- Bijl, Aart; Stone, David; and Rosenthal, David (1979) Integrated CAAD Systems. DoE No. DGR 470/12, EdCAAD, Edinburgh University, Edinburgh, 1979.
- Bijl, Aart (1982) Dumb Drawing Systems and Knowledge Engineering. In Alan Pipes (ed.) *CAD '82*, Brighton, April, 1982, pp346-364.
- Bijl, Aart (1983) Can Computers Understand Designers?. In *PARC '83*, London, 1983, pp169-181.
- Bijl, Aart and Szalapaj, Peter (1984) Computer Literacy: Designers' Despair or Hope?. In *The Role of the Designer*, Design Research Society Conference, Bath, 1984.
- Bijl, Aart (1985a) Computer Aided Design and Artificial Intelligence. In *ESCAD '85 Workshop*, Reading, July, 1985a.
- Bijl, Aart (1985b) Graphical Input: Can Computers Understand People?. *Comput. & Graphics*, Volume 9, No. 8, pp85-95.

Bijl, Aart (1989) *Design in Mind & Machine: Computer Discipline and Design Practice*. Book in Preparation.

Bijl, Aart and Szalapaj, Peter (1985) Saying What You Want With Words And Pictures. In B. Shackel (ed.) *IFIP INTERACT '84*, London, 1985, pp275-280.

Bobrow, Daniel G. (1975) Dimensions of Representation. In *Representation and Understanding: Studies in Cognitive Science*. New York: Academic Press.

Bobrow, Daniel G. and Collins, Allen M. (eds.)(1975) *Representation and Understanding: Studies in Cognitive Science*. New York: Academic Press.

Bobrow, Daniel G. and Winograd, Terry (1977) An Overview of KRL, a Knowledge Representation Language. *Cognitive Science*, Volume 1, pp3-46.

Booch, G. (1986) Object-Oriented Development. In *IEEE Transactions on Software Engineering*, Volume SE-12.

Brachman, Ronald J., and Levesque, Hector J. (eds.)(1985) *Readings in Knowledge Representation*. Los Altos, California: MORGAN KAUFMANN.

Brachman, R.J. and Schmolze, J. (1985) An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, Volume 9, No. 2.

Braid, I. C. (1973) *Designing With Volumes*, Second Edition. Cambridge: CANTAB Press.

Bresnan, Joan W. (1976) On the Form and Functioning of Transformations. *Linguistic Inquiry*, Volume 7, No. 1, pp3-40.

Broadbent, Geoffrey; Bunt, Richard; and Jencks, Charles (eds.)(1980) *Signs, Symbols, and Architecture*. Chichester: JOHN John Wiley & Sons.

Brüderlin, Beat (1985) Using Prolog for Constructing Geometric Objects Defined by Constraints. In Bob F. Caviness (ed.) *European Conference on Computer Algebra*, Linz, Austria, April, 1985, pp448-459.

Chomsky, Noam (1957) *Syntactic Structures*. The Hague: MOUTON.

Chomsky, Noam (1965) *Aspects of the Theory of Syntax*. MIT Press.

Chomsky, Noam (1966) *Topics in the Theory of Generative Grammar*. The Hague: MOUTON.

Chomsky, Noam (1975) *The Logical Structure of Linguistic Theory*. London: The UNIVERSITY of CHICAGO Press Ltd..

Clocksins, William F. and Mellish, Christopher S. (1981) *Programming In Prolog*. Berlin: Springer Verlag-VERLAG.

Cooke, Catherine (1984) *Chernikhov, Fantasy and Construction: Iakov Chernikhov's Approach to Architectural Design*. London: LONDON ARCHITECTURAL DESIGN.

Dreyfus, Hubert L. (1979) *What Computers Can't Do: The Limits Of Artificial Intelligence*, Harper Colophon Edition. London: HARPER & ROW.

Earl, C.F. (1987) Representing Relations. Chapter 2 in Rooney, J. and Steadman, P. (eds.) *Principles of Computer-Aided Design*, pp39-56. PITMAN/Open University.

Earle, Nick (1973) *Logic*. London: MACMILLAN Press Ltd..

Eco, Umberto (1980) Function and Sign: The Semiotics of Architecture. Chapter 1.1 in Geoffrey Broadbent, Richard Bunt, and Charles Jencks (eds.) *Signs, Symbols, and Architecture*, pp11-69. Chichester: JOHN John Wiley & Sons.

Flew, Anthony (1979) *A Dictionary of Philosophy*. London: PAN BOOKS Ltd..

Foley, James D., and Van Dam, Andries (1984) *Fundamentals of Interactive Computer Graphics*. London: Addison Wesley.

Fowler, R. (1971) *An Introduction to Transformational Syntax*. ROUTLEDGE and KEGAN PAUL Ltd..

Frost, R.A. (1986) *Introduction to Knowledge Base Systems*. London: COLLINS.

Gero, John S., Akiner, V. Tuncer, Radford, Anthony D. (1983) What's What and What's Where: Knowledge Engineering in the Representation of Buildings by Computer. In *PARC '83*, London, October, 1983, pp205-215.

Gimeor S.A. (1987) *MacArchitron*. Technical Report, Douai, France, 1987.

Gips, James (1975) *Shape Grammars and their Uses: Artificial Perception, Shape Generation and Computer Aesthetics*. Basle and Stuttgart: BIRKHAUSER VERLAG.

Gombrich, E. H. (1960) *Art and Illusion: A Study in the Psychology of Pictorial Representation*. Oxford: PHAIDON Press.

Goodman, Nelson (1968) *Languages of Art: An Approach to a Theory of Symbols*. Indianapolis: HACKETT Publishing Company, Inc..

Goodman, Nelson (1966) *The Structure of Appearance*. BOBBS-MERRILL Company, Inc..

Hamilton, Ian and Scoins, David (1980) *Computer Draughting In Construction*. Evaluation Report No. 7, Construction Industry Computing Association, Cambridge, 1980.

Harris, Zelig (1956) *Introduction to Transformations*. In Zelig Harris (ed.) *Papers in Structural and Transformational Linguistics*, reprinted 1970. Dordrecht, Holland: REIDEL.

Hayes, Patrick J. (1974) *Some Problems and Non-Problems in Representation Theory*. In *AISB Summer Conference*, University of Sussex, Brighton, 1974, pp63-79.

Hofstadter, Douglas R. (1979) *Gödel, Escher, Bach: An Eternal Golden Braid*. HARVESTER Press Ltd..

Holmes, Chris (1979) *Subsets of 2-D Space: Computational Geometry of Polygonally-Bounded Areas*. Technical Report, EdCAAD, Edinburgh University, Edinburgh, July, 1979.

Jackson, H.T. (1962) *The Design of Structural Members*, Volume Two. London: Architectural Press Ltd..

Kernighan, Brian W. and Ritchie, Dennis M. (1978) *The C Programming Language*. Englewood Cliffs, NJ: PRENTICE-HALL Inc..

Kirsch, Russell A. (1964) *Computer Interpretation of English Text and Picture Patterns*. *IEEE Transactions on Electronic Computers*, Volume EC-13, No. 4 (August), pp363-376.

Klee, Paul (1953) *Pedagogical Sketchbook*. London: FABER and FABER.

de Kleer, J. (1984) *Choices without Backtracking*. In *AAAI-84*, 1984, pp79-85.

Klein, Ewan (1987) *A Saussurean Approach to Graphics*. unpublished memorandum, Cognitive Science, Edinburgh University, Edinburgh, June, 1987.

Knight, T. W. (1980) *The generation of Hepplewhite-style chair-back designs*. *Planning and Design: Environment and Planning B*, Volume 7, pp227-238.

Koning, H. and Eizenburg, J. (1981) The language of the prairie: Frank Lloyd Wright's prairie houses. *Planning and Design: Environment and Planning B*, Volume 8, pp295-323.

Krishnamurti, Ramesh (1980) The Arithmetic of Shapes. *Planning and Design: Environment and Planning B*, Volume 7, pp463-484.

Krishnamurti, Ramesh (1981) The Construction of Shapes. *Planning and Design: Environment and Planning B*, Volume 8, pp5-40.

Krishnamurti, Ramesh (1986) Representing Design Knowledge. SERC Final Report (GR/C/07568), EdCAAD, Edinburgh University, Edinburgh, April, 1986.

Lansdown, John (1982) Expert Systems: Their Impact on the Construction Industry. Technical Report, RIBA, London, 1982.

Lyons, John (1968) *Introduction to Theoretical Linguistics*. Cambridge: CAMBRIDGE UNIVERSITY Press.

March, Lionel, and Steadman, Philip (1971) *The Geometry of Environment*. London: RIBA Publications Ltd..

McDermott, Drew (1986) A Critique of Pure Reason.

Mellone, Sydney Herbert (1916) *An Introductory Text-Book of Logic*. Edinburgh: WILLIAM BLACKWOOD & SONS Ltd..

Minsky, Marvin (1975) A Framework For Representing Knowledge. Chapter 6 in *The Psychology Of Computer Vision*. McGRAW.

Murray, J. L. (1982) CAM-X - A Linked CAE System. In *CAD '82*, Brighton, 1982, pp44-53.

Patterson, David A. (1985) Reduced Instruction Set Computers. *Communications of the ACM*, Volume 28, No. 1, pp8-21.

Pereira, Fernando (1982) Seelog - A Prolog Graphics Interface. Technical Report, EdCAAD, Edinburgh University, 1982.

Pong, M. C. and Ng, N. (1983) PIGS - A System for Programming with Interactive Graphical Support. *Software - Practice and Experience*, Volume 13, pp847-855.

Pye, A. D. (1978) *The Nature and Aesthetics of Design: A Design Handbook*. London: HERBERT Press Ltd..

Radford, A. (1981) *Transformational Syntax*. CAMBRIDGE UNIVERSITY Press.

Rentsch, T. (1982) Object-Oriented Programming. *SIGPLAN Notices*, Volume 9.

Rittel, Horst W. J. and Webber, Melvin M. (1984) Planning Problems are Wicked Problems. Chapter 2.3 in *Developments in Design Methodology*. Chichester: JOHN John Wiley & Sons. Nigel Cross (ed.).

Rosenfeld, A. (1971) Isotonic Grammars. In *Machine Intelligence*, Volume 6. Edinburgh: EDINBURGH UNIVERSITY Press.

Rushworth, Roger (1982) STAG User Manual. Technical Report, EdCAAD, Edinburgh University, 1982.

Scalvini, Maria Luisa (1980) Structural Linguistics versus the Semiotics of Literature: Alternative Models for Architectural Criticism. Chapter 3.3 in Geoffrey Broadbent, Richard Bunt, and Charles Jencks (eds.) *Signs, Symbols, and Architecture*, pp411-420. Chichester: JOHN John Wiley & Sons.

Shaw, Alan Cary (1968) The Formal Description And Parsing Of Pictures. PhD Thesis, Mathematics Department, Stanford University.

Slovan, Aaron (1975) Afterthoughts on Analogical Representations. In *Theoretical Issues in Natural Language Processing*, Cambridge, MA, 1975, pp164-168.

Slovan, Aaron (1978) *The Computer Revolution in Philosophy: Philosophy, Science and Models of Mind*. Hassocks: HARVESTER Press Ltd..

Steel, Sam W. D. and Szalapaj, Peter J. (1983) Pictures Without Numbers. In *PARC '83*, London, 1983, pp217-232.

Stewart, Ian (1975) *Concepts of Modern Mathematics*. Harmondsworth: PENGUIN.

Stiny, George (1975) Pictorial and Formal Aspects of Shape Grammars and Aesthetic Systems. PhD Thesis, System Science, UCLA.

Stiny, George (1976) Two exercises in formal composition. *Planning and Design: Environment and Planning B*, Volume 3, pp187-210.