STATE UNIVERSITY OF NEW YORK AT BUFFALO
DEPARTMENT OF COMPUTER SCIENCE

A KNOWLEDGE REPRESENTATION THEORY FOR NATURAL LANGUAGE GRAPHICS

by

James Geller

A dissertation submitted to
the Faculty of the Graduate School of
the State University of New York at Buffalo
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy.

May 1988

ABSTRACT

Natural Language Graphics (NLG) deals with diagram generation driven by natural language utterances. This investigation applies the methods of declarative knowledge representation to NLG systems. Declarative knowledge that can be used for display purposes as well as reasoning purposes is termed "Graphical Deep Knowledge" and described by supplying syntax and semantics of its constructs. A task domain analysis of Graphical Deep Knowledge is presented covering forms, positions, attributes, parts, classes, reference frames, inheritability, etc.

Part hierarchies are differentiated into three sub-types. The usefulness of inheritance along part hierarchies is demonstrated, and criticism of traditional inheritance-based knowledge representation formalisms is derived from this finding. The "Linearity Principle of Knowledge Representation" is introduced and used to constrain some of the presented knowledge structures. The analysis leading to Graphical Deep Knowledge also results in the description of two fundamental conjectures about knowledge representation.

The Gricean maxims of cooperative communication are used as another source of constraints for NLG systems. A new maxim for technical languages is introduced, the "Maxim of Unnecessary Variation". It is argued that common symbolic representations like circuit board diagrams have not yet been described in the literature by explicit feature analysis, and that this is necessary to give a system knowledge about the meaning of the diagrams it is displaying.

Part of the presented theory has been implemented as a generator program that creates pictures from knowledge structures and as an ATN grammar that creates knowledge structures from limited natural language input. The function of the picture generation program ("TINA") as a user interface for a circuit board maintenance system (VMES) is demonstrated. Finally an older version of TINA is described that incorporates a module for "Intelligent Machine Drafting" (IMD), a completely new subfield of AI that has been introduced in this research and that relates to Computer Aided Design (CAD). The IMD program does layout and routing for the members of a simple class of functional circuit diagrams based on a policy of symmetry and equal distribution over the available space. This layout/routing is based on cognitive requirements as opposed to physical requirements used by CAD systems.

TABLE OF CONTENTS

1. INTRODUCTION 1
   1.1. Overview 1
   1.2. Outline of the Dissertation 2
   1.3. Literature Review 2

2. A SCENARIO FOR THE USE OF NLG SYSTEMS 8

3. WHAT PHILOSOPHY HAS TO OFFER TO NLG 29
   3.1. Grice's Maxims of Co-operative Communication 29
   3.2. A special Maxim for Technical Languages 36
   3.3. Feature Analysis of Symbolic Graphical Representations 40
      3.3.1. Features Common in Symbolic Representations 42
      3.3.2. Logical Wire Plans 44
      3.3.3. Physical Wire Plans 45
      3.3.4. SNePS Networks 47
      3.3.5. Maps 48
      3.3.6. Pie Charts 49
      3.3.7. Bar Graphs 50
      3.3.8. Euler Circles 50
      3.3.9. Graphs of Functions of one Variable 50
      3.3.10. Flow Charts 51
      3.3.11. Syntax Diagrams 51

4. GRAPHICAL DEEP KNOWLEDGE AND INTELLIGENT INTERFACES 52

5. GRAPHICAL DEEP KNOWLEDGE AND REASONING FOR NLG 55
   5.1. Knowledge Representation 55
      5.1.1. A Goal Statement for NLG 56
         5.1.1.1. The Reductionist Character of Representations 56
         5.1.1.2. The Need for an Internal Relevance 58
         5.1.1.3. Differentially Adequate Knowledge Representation 60
         5.1.1.4. A Second Fundamental Conjecture about Knowledge Representation 62
         5.1.1.5. Definition and Goal of Graphical Deep Knowledge Research 64
         5.1.1.6. Semantic Primitives and Differential Adequacy 65
         5.1.1.7. The Linearity Principle of Knowledge Representation 67
      5.1.2. The SNePS Knowledge Representation System 70
         5.1.2.1. Introduction 70
         5.1.2.2. Notational Conventions for SNePS Networks 73

5.1.3. Representational Constructs of Graphical Deep Knowledge 75
5.1.3.1. Form Knowledge 75
5.1.3.1.1. Individual Form 81
5.1.3.1.. The Problem of Display Modalities 84
5.1.3.1.3. Modalities versus Partitions 86
5.1.3.1.4. Form with n Step Inheritance 88
5.1.3.2. Positions 91
5.1.3.2.1. Requirements for Position Representations 91
5.1.3.2.2. Reference Frames 93
5.1.3.2.2.1. Concrete Units in Reference Frames 101
5.1.3.2.2.2. Fuzzy Units in Reference Frames 101
5.1.3.2.2. Modalities Revisited 102
5.1.3.2.3. The Representation of Positions 103
5.1.3.2.4. Concrete versus Fuzzy 2-d Descriptions 107
5.1.3.2.5. Polar Coordinates 109
5.1.3.2.6. Absolute versus Relative Positions 110
5.1.3.2.6.1. Fuzzy Absolute Positions 111
5.1.3.2.7. Explicit versus Deduced Reference Objects 114
5.1.3.2.8. Own versus Inherited Relative Positions 116
5.1.3.2.9. Coordinate Unit Types 117
5.1.3.2.9.1. Fuzzy Coordinate Unit Types 120
5.1.3.2.10. Screen, Plane and World Coordinates 121
5.1.3.2.11. 2 1/2 Dimensional Representation 122
5.1.3.3. The Representation in Attributes of Graphical Deep Knowledge 123
5.1.3.3.1. Types of Attributes 123
5.1.3.3.2. Simple Attribute Representations 126
5.1.3.3.3. Attribute Mappings 131
5.1.3.3.4. Unmapped Attributes 135
5.1.3.3.5. Inheritability of Attributes 137
5.1.3.3.6. The Representation of Superlatives and Comparatives 138
5.1.3.4. Part Hierarchies 139
5.1.3.4.1. The Definition of Parts 140
5.1.3.4.2. A General Purpose Part Representation 142
5.1.3.4.3. Real Parts 142
5.1.3.4.4. Assemblies 144
5.1.3.4.5. Clusters 146
5.1.3.4.6. A Comparison of Taxonomies 150
5.1.3.4.7. Are there more Graphically Interesting Part Relations? 151
5.1.3.4.8. Modalities and Parts 152
5.1.3.4.9. The Inheritability of Attributes in a Part Hierarchy 153
5.1.3.5. The Class Hierarchy 155
5.1.3.5.1. The Class Inheritance Mechanism 155
5.1.3.5.2. Class Constructs and Categorization Theory 157
5.1.3.6. Periodicity 160

5.1.3.7. Angle 163
5.1.3.8. Pragmatic Hierarchies 164
5.1.4. Knowledge Compilation 165
5.2. Reasoning 168
5.2.1. Logical Reasoning about GDK Structures 168

6. IMPLEMENTATION 173
6.1. The TINA Display Program 174
6.2. Example Runs 180
6.3. Readform, the Graphics Editor 254
6.4. TINA used as Maintenance Interface 255
6.5. Limitations of the Implementation 257

7. INTELLIGENT MACHINE DRAFTING 260
7.1. The Representations of Ports and Connections 260
7.2. Layout and Routing as Intelligent Activities 263
7.3. Intelligent Machine Drafting 264
7.4. The IMD Grammar 273

8. FUTURE WORK 276

9. CONCLUSIONS 278

ACKNOWLEDGEMENT

I would like to thank G'd that he has made me see this day, and that he has given me the energy to do all the work that was necessary for it.

I thank Stuart C. Shapiro, my advisor. Even if I could do something for him every day from now till 2088, I could never pay back what he has done for me. He has listened to me patiently for hours and hours; his door was always open, literally and metaphorically. At a time when many people see teaching as a job of eliminating the students that do not already know, he has taken me as a challenge, to show that he can explain anything and everything. He has translated my "writings" from what I thought is English to real English. His grant ‡ has paid my rent and some of my conference trips, and he even took me with him on his Sabbatical. Since my father's (of blessed memory) death he has been the nearest to a replacement for him. Stuart Shapiro's high personal and academic integrity is known to anybody that has ever dealt with him, his academic rank is well known in the field. I have been very lucky to find such an advisor.

I want to thank Sargur N. Srihari my " co-advisor" for lots of support and help that he has given me through the years, and for saying the right words at the right time. He is certainly a shining example of what one can achieve in a short time if one just knows how to do it and is willing to work hard for it. On top of all he has always been a good sport, more of a fellow graduate student than a faculty member.

I want to thank my third committee member, Deborah K. W. Walters, whose mind is sharp as a razor blade, and who has taught me what the word "scholarship" really means. She has shown me that science means a fight for the truth in which every word counts. Maybe I repeat

---

‡ This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

myself now, but I have been lucky to have met her, and I am happy to have asked her to be on my committee.

I would like to thank Norman Sondheimer, my outside reader, for supporting me for a year at ISI, which is certainly the most beautiful place in the world to do research at. His comments on my work are deeply appreciated, and so are his jokes.

Next to be thanked is Helene Kershner. I could give a million reasons why, but it is not necessary, because Helene is one of the preciously few people whom you thank just for being as they are. Helene can say things about life that are fit for printing without thinking a second. If she ever decides to run for governor then my sympathy to her opponents. I love you, Helene.

I would also like to thank Harry Delano who is one of those "miracles take 24 hours people". I have never been anywhere where software services were as fast and reliable as in the SUNY CS department, and it is the only place in the world I know, where the word "user" is not a four letter word.

I want to thank Ellie Benzel, secretary, who has, in her own words, taken it upon herself to mother me poor foreign student away from home, and Gloria Koontz and Lynda Spahr who have tried to compete with her. I have to say that in all my life I have never received that much (undeserved) love from so many people as in the CS department in Buffalo. Buffalo might be a cold city, but there is a lot of warmth in the hearts of many people. I will never forget anyone of you.

There are lots and lots of other people who really deserve more than a brief mentioning, like Jeff Zucker, John Case, Bill Rapaport and Pat Eberlein among the faculty; Mingruey Taie my project mate; An Tzu Chin my long time office mate; Joao Martins, Ernesto Morgado, Jeannette Neal, and Terry Nutter, my seniors in the SNePS research group; I am sorry, but I will never reach their level of achievement; Jon Hull and Radmilo Bozinovics, who helped me make the first steps in Buffalo; Scott Campbell, Keith Bettinger, Amy Melcher and many other class mates with whom I share great memories of organizing two conferences; Michael Almeida, Jan Wiebe,

Hanyong Yuhan, Sandy Peters and other members of the SNePS research group who have contributed interesting ideas to my work; Bill Eggers, my student, whose program I have been using for several years, and many many other students some of whom, like Lynda Kingsbury, have given me the greatest compliment possible, they have taken every class I taught.

I want to thank the members of the II project team at ISI, Larry Miller, Yigal Arens, and Ray Bates, for having taught me many valuable things, and Paul Raveling for the screen dump routine that I have been using constantly. Also Tom "W" should be mentioned, who keeps the software running at ISI.

Finally I want to thank my mother who has given me as much love as one person can possibly give, Abraham Chaim Wertzberger for having taught me Torah, and S. C. who has shared my crazy life in joy and in frustration, and in long nights in the computer room.

CHAPTER 1

INTRODUCTION

And the Lord spoke unto Moses saying. "Speak unto Aaron, and say unto him: When you lightest the lamps, the seven lamps shall give light in front of the candlestick." And Aaron did so: he lighted the lamps thereof so as to give light in front of the candlestick, as the Lord commanded Moses. And this was the work of the candlestick, beaten work of gold; unto the base thereof, and unto the flowers thereof, it was beaten work; according unto the pattern which the Lord had shown Moses, so he made the candlestick.

Numbers, 8:1-4

1.1. Overview

This dissertation deals with the problem of generating diagrams in reaction to natural language utterances, a task that has been classified scientifically as Natural Language Graphics.

Our approach to Natural Language Graphics is based on a declarative knowledge base, implemented using a propositional semantic network. The theory developed is domain independent, but the underlying task domain has been a graphical user interface to a circuit board maintenance system.

In the course of this investigation the following questions will be addressed.

(1) What are the knowledge structures that are necessary (and ideally also sufficient) to cover the generation of a wide range of diagrams?

(2) What can knowledge representation (KR) research learn by rigorously applying its methods to a domain which is different from the one for which KR was initially designed, namely language understanding and generation?

(3) Can known pragmatic theories of language communication be extended to graphical representations?

(4) Are there any novel applications that capitalize from an improved theoretical basis for NLG?

The main focus of this dissertation is (1). To show the consistency and usefulness of results from (1), an implementation will be presented.

1.2. Outline of the Dissertation

In Section **1.3** we will review relevant literature.

In Chapter **2** we will present a test run of "TINA" the implementation of the theory developed in this dissertation. By discussing aspects of this test run we will point out desirable features of NLG systems.

Chapter **3** introduces Grice's theory of pragmatic maxims and presents an application of this theory to diagrammatic representations. Common features in diagrammatic representations will be listed, and a number of common representation systems will be analyzed according to their use of these features as semantic, accidental or conventional.

Chapter **4** discusses the relations between Natural Language Graphics and Intelligent Interfaces.

Chapter **5** is the major part of this dissertation. After defining the notion of Graphical Deep Knowledge (GDK) and supplying a detailed goal statement for NLG, a task domain analysis of GDK is presented.

Chapter **6** discusses the implementation of the theory presented in Chapter **5**. Numerous test runs are demonstrated.

Chapter **7** reviews an implementation that was done earlier in this research effort. The field of "Intelligent Machine Drafting" is defined and its relations to Graphical Deep Knowledge as well as to Computer Aided Design are discussed.

Finally, chapter **8** lists future work, and chapter **9** presents our conclusions.

1.3. Literature Review

Natural Language Graphics (NLG) [BrK77] is a subfield of computer science at the intersection of computer graphics and artificial intelligence. An NLG system is a program which permits a user to create, manipulate and query graphical representations with natural language. Typically

the given language input consists of descriptions of objects and spatial relations between them, followed by questions about relations not explicitly contained in the input. Alternatively a user can give commands that describe actions that he wishes performed on the objects of his description.

The manifesto of Natural Language Graphics is a paper by Brown and Chandrasekaran [BrC81]. Programs that can be classified as NLG systems are described in [AMG84a, AMG84b, MGP84, YTK84].

Brown and Chandrasekaran [BrC81] are supplying an impressively complete analysis of the graphics phenomena involved in NLG. They also correctly differentiate between "knowledge based graphics" and "conventional computer graphics". They say that "In all of computer graphics....a great deal of emphasis on efficient, compact, hand-tailored data structures. Our emphasis is almost in the other direction. We need a scheme with which to represent many different kinds of information in an explicit way,...(p. 182)". Brown and Chandrasekaran use a frame system as their knowledge representation. A small implementation is described, and the rest of the paper deals with "a design for the picture production part of the system" (p. 178).

Brown and Chandrasekaran also review attempts to combine natural language and graphics that predate NLG as a field. Kirsch [Kir64] discusses the combination of natural language and diagrams and, besides dealing with problems of natural language understanding, presents the first attempt in the literature to formalize the notion of a two-dimensional grammar. The best known early NLG-like program is Winograd's SHRDLU [Win72]. SHRDLU is of course known as a paradigm of using procedural knowledge, while we are interested in declarative knowledge.

The earliest combination of graphics with semantic networks, the knowledge representation technique that we will make use of, is reported in [GLM78]. Giustini refers to the reported semantic network as "mathematical" as opposed to "linguistic", meaning that it does not span the complete set of properties that one would expect from a semantic network. Specifically it is neither "teachable" nor is it possible "to retrieve information given data whose format differs from that by which the memory previously learned the information" (p. 13).

The work by Adorni, Di Manzo et al. [AMG84a,AMG84b,MGP84] concentrates on problems of equilibrium, support, instantiation of unmentioned objects and space occupied by an object. It does not draw a clear distinction between the domain of diagrams and the domain of real world objects. This makes it difficult to think about different diagrammatical views of the same object. Another problem with Adorni et al.'s work is that their publications do not contain screen dumps. The range of their graphics efforts is therefore not readily accessible to the reader. The research concentrates on difficult natural language problems and uses diagrams only as a proof of the NL understanding abilities of the system, not as main subject of investigation. We will refer to this approach as *criterial use* of NLG.

The weather map analysis system by Yokota et al. [YTK84] offers language input and output as well as pictorial input and output with retrieval of maps based on pictorial input. This makes their system the most complete of all approaches to NLG, however it is limited to a small domain of expertise, namely weather situations in the Far East.

It is a largely accepted practice in natural language generation that utterances are generated from a knowledge base by an independent mechanism comprising the lexical and grammatical knowledge of the system. The result of the generation process consists of a chain of verbal primitives (morphemes). The model of NLG presented here follows the same pattern. Diagrams are generated from a propositional knowledge base and consist of pictorial primitives (icons). This approach has been discussed previously in work of Kosslyn and Shwartz on imagery [KoS77]. In a lengthy argument (p. 270-271) the authors argue that "some sort of constructive activity" is needed to assemble an image and that the units of assembly are presumably "entire interpreted units of an image". We want to identify these units with icons in a graphics system. Concerning the propositional linkage Kosslyn and Shwartz continue their argument and state that "... images could be created by relating together perceptual and more abstract 'propositional' representations... To buttress the claim.... we performed a simple experiment:... Thus conceptual information can be used in image generation..." (p. 270-271).

There is another interesting connection between NLG and natural language processing. It has been noted by Waltz [Wal80] that in many cases the formation of an image is a necessary ingredient of natural language understanding as a whole. In other words, the language understanding problem will not be solved completely unless the process of picture generation from language is well enough understood.[1] In the previously mentioned imagery literature Kosslyn [KoS77,Kos80,Kos81a,Kos81b,Kos85] has investigated the generation of diagrams from a partially propositional representation for purposes of "visual" reasoning. He states that "Propositions are abstract languagelike discursive representations, corresponding roughly to simple active declarative statements" [Kos81b]. Kosslyn does not investigate interactions with natural language and does not present a formal catalog of his representations, but he lists important items that need to be represented, and his approach has been an inspiring guide for our work.

The view of NLG systems as auxiliary devices in the process of language understanding does not imply a limitation of their usability to spatial language, i. e. language dealing with physical objects and location relations between them. It is well known [MiJ76] that much of language makes use of spatial metaphors.

There are two other research efforts that relate closely to this investigation. Friedell [Fri84] describes the generation of images from high level object specifications which he calls "quasi-descriptions". He presents two example systems one for the generation of ship images from a data base and one for the automatic synthesis of backgrounds for three-dimensional scenes. Friedell's approach is similar to ours in that he uses a graphics knowledge base. It differs in that we are using a semantic network, while his work is based on frames. He also does not interface his knowledge base to a natural language parser.

Under the name "Information Presentation System" (IPS) another approach to image generation from a knowledge base has been published [ZGY81]. The AIPS system is based on the KL-ONE semantic network and is in spirit similar to our approach. However, there is

[1] Not all scholars agree with the merit of internal images, but this discussion is of no concern here.

disagreement concerning the success of the details of the AIPS theory. While Zdybel et al. state that "By an IPS we mean a system that... functions reasonably well without demanding custom-tooling for a particular application..." (p. 978) they do not make it clear that their system lives up to this expectation. Friedell notes that "Systems such as BARAT and AIPS succeed in narrow, well-defined domains for which it is practical to provide an adequate repertoire of predefined parametric object descriptions." (p. 54). Our experience with KL-ONE [2] has been that object attributes of individuals are represented in an unintuitive way, because KL-ONE is by its nature not designed for dealing with individuals. Our own representation of attributes is very general and is not characterized by what Friedell calls "parametric object descriptions".

Since the original inception of NLG as a field, the attitude in some areas of computer graphics has changed, and "world model based" graphics systems [FoD83] now include "redundant" information. However, equating this with knowledge based graphics would be tantamount to ignoring twenty years of research in knowledge representation.

The research described in this dissertation has grown out of work on a circuit board maintenance system, and this naturally raises the question about NLG-like work in CAD. It turns out that references to natural language in CAD are sparse. An exception is Samad [Sam86], who cites four reasons why natural language *is* useful in CAD. Natural Language is not tool specific, does not require extensive manual use, permits natural expression, and overcomes limitations of menu based systems in complex applications. Samad's work deals with post processing and querying of the output of a simulation system and is not graphics oriented.

Recently there has been increased interest in NLG-like behavior of programs in the area of multi media interfaces and knowledge based user interface management systems (KBUIMS). This is in line with our own implementation that was designed as a user interface. The HITS (human interface tools) system [HMR88] is an example for these efforts. HITS is heavily graphics oriented,

---

[2] I am thankful to Norm Sondheimer at ISI who has given me ample opportunity to get hands on experience with KL-ONE.

but has also a natural language component. While the sub-systems of HITS surpass the sub-systems of our own implementation, the designers of HITS apparently had to find out that the integration of sub-systems in NLG, unless planned from the beginning, as well as the use of an appropriate knowledge representation system are crucial for such efforts. "HITS currently exists primarily as a set of independent tools, most of which are implemented on top of the Proteus knowledge base system [..]. We are in the process of reimplementing many of these tools on top of a richer knowledge representation system..." (Section 4).

Another approach to multi media interfaces that is based on our earlier endeavors is [NeS88]. Two workshops give a good overview of the current state of the art in multi-media/knowledge-based user interfaces [NeK86b,SuT88].

CHAPTER 2

A SCENARIO FOR THE USE OF NLG SYSTEMS

And the Lord said, Behold, the people is one, and they have all one language; and this they begin to do: and now nothing will be restrained from them, which they have imagined to do. Go to, let us go down, and there confound their language, that they may not understand one another's speech.

Genesis, 11:6-7

In this chapter we will present an example dialogue to clarify the problems attacked in this dissertation. It has been compiled from the demonstrations in the implementation chapter (Chapter 6) and irrelevant parts have been removed or edited for clarity. Especially "screen erase" commands have been deleted in a few places. User input follows the " : >>" prompt. Annotations are given in square brackets.

[*The system has some basic information about the valid coordinate system. It also knows a few form primitives, like ship-form, xand, xport, xboard, etc. Finally it knows the correct coordinates of a few special locations like "top".*]

: >>ship-1 is a ship (1)

PARSED

: >>the form of a ship is ship-form (2)

PARSED

: >>ship-1 is at 200 300 on the screen (3)

PARSED

: >>please show ship-1 (4)

[*Refer to Fig. 2.1*]

: >>erase the screen (5)

DONE

: >>the state of ship-1 is non-operational (6)

PARSED

: >>state is expressed by rotation and 180 represents non-operational (7)

[*We have just informed the system how to graphically realize the non-graphical attribute "non-*

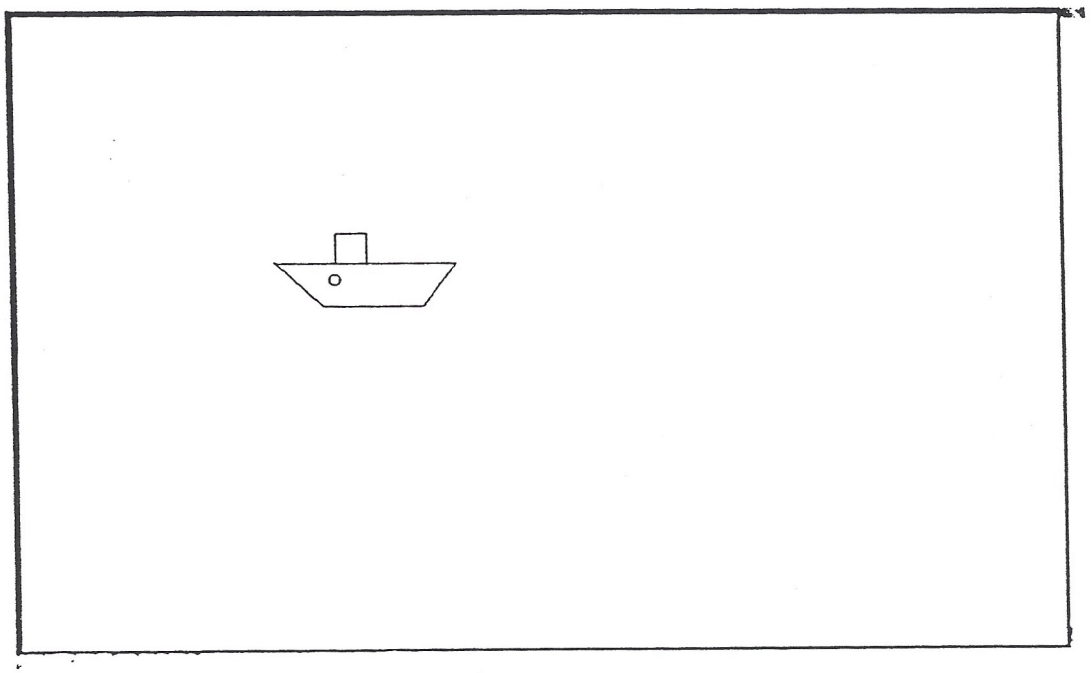


Fig. 2.1

operational" namely by rotating the icon of a non-operational object by 180 degree. "rotation" is a graphics function that performs icon rotation.]

PARSED

: >>show ship-1 (8)

[*Refer to Fig. 2.2*]

: >>what is the form of ship-1? (9)

SHIP-FORM

: >>what is the state of ship-1? (10)

non-operational

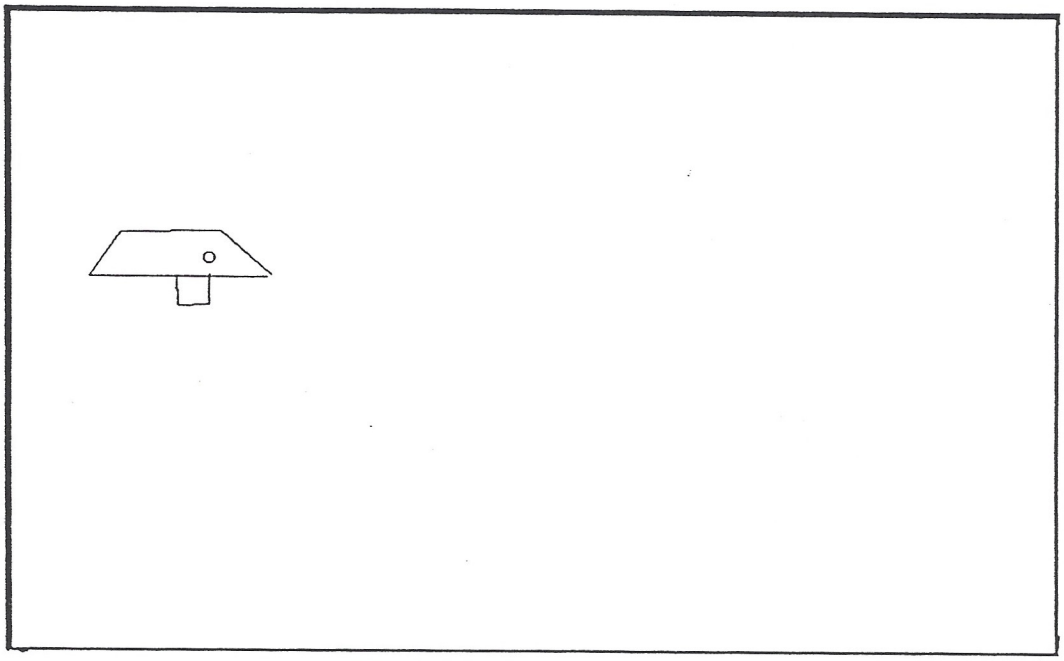


Fig. 2.2

: >>where is ship-1? (11)

(200 300 " relative to" screen-center)

: >>what are the members of ship? (12)

SHIP-1

: >>clear the screen (13)

DONE

: >>elmer-montgomery is a ship (14)

PARSED

: >>elmer-montgomery is here (15)

[*"here" activates the mouse, and the position is entered by pressing the left mouse button.*]

PARSED

: >>what are the members of ship? (16)

(ELMER-MONTGOMERY SHIP-1)

: >>how would elmer-montgomery look with state non-operational? (17)

[*See Fig. 2.3. Above example of hypothetical reasoning is implemented by building and then erasing a structure asserting the non-operational attribute.*]

SO

: >>clear the screen (18)

DONE

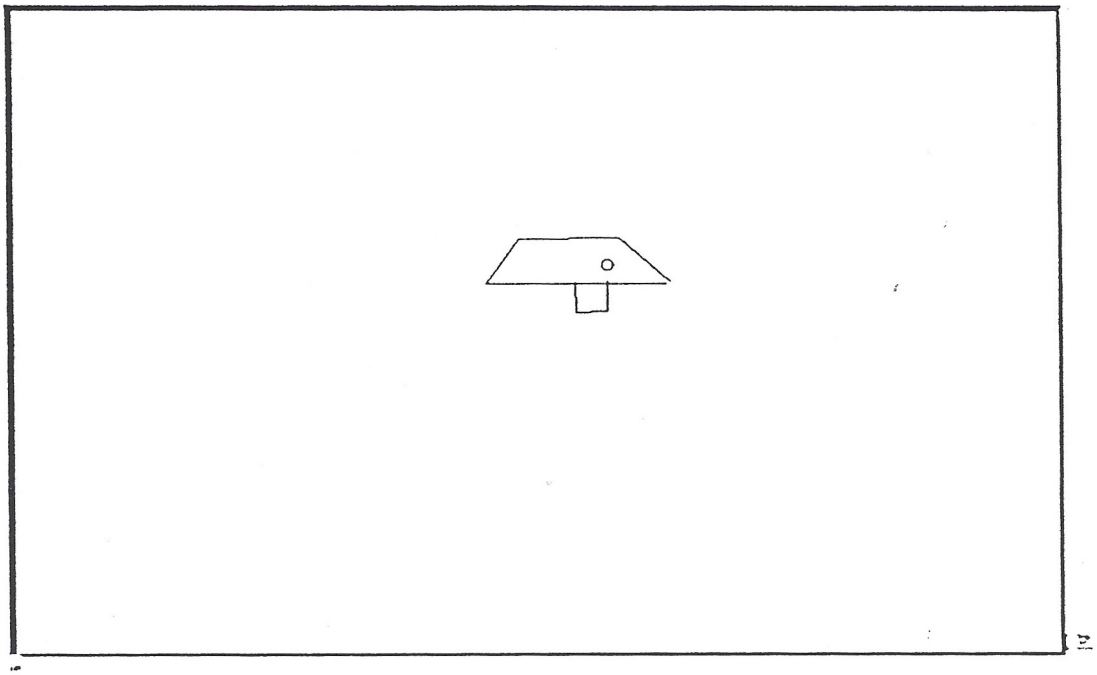


Fig. 2.3

: >>show elmer-montgomery (19)

[*See Fig. 2.4*]

DONE

: >>the form of a board is xboard (20)

PARSED

: >>b2 is a board (21)

PARSED

: >>b2 is at 2 6 inches on the screen (22)

PARSED

: >>b2 has and-1, and-2 and or-1 as parts (23)

PARSED

: >>and-1 is at 1 -1 inches relative to b2 (24)

PARSED

: >>the form of and-1 is xand (25)

PARSED

: >>and-2 is at 1 -2 inches relative to b2 (26)

PARSED

: >>the form of and-2 is xand (27)

PARSED

: >>or-1 is at 1 -3 inches relative to b2 (28)

PARSED

: >>the form of or-1 is xor (29)

PARSED

: >>the form of a port is xport (30)

PARSED

: >>port-1, port-2 and port-3 are parts of and-1 (31)

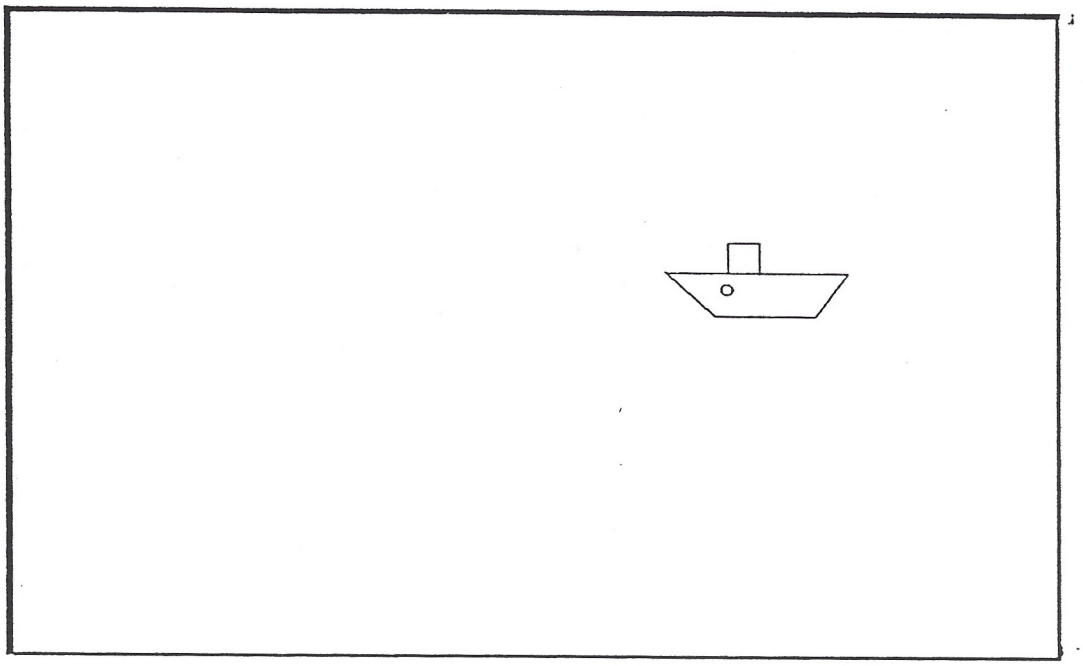


Fig. 2.4

PARSED

: >>port-1 and port-2 and port-3 are members of port (32)

PARSED

: >>port-1 is at -30 -5 relative to and-1 (33)

PARSED

: >>port-2 is at -30 -25 relative to and-1 (34)

PARSED

: >>port-3 is at 60 -15 relative to and-1 (35)

PARSED

: >>erase the screen (36)

DONE

: >>display 2 levels of b2 (37)

[*Fig. 2.5*]

DONE

: >>erase the screen (38)

DONE

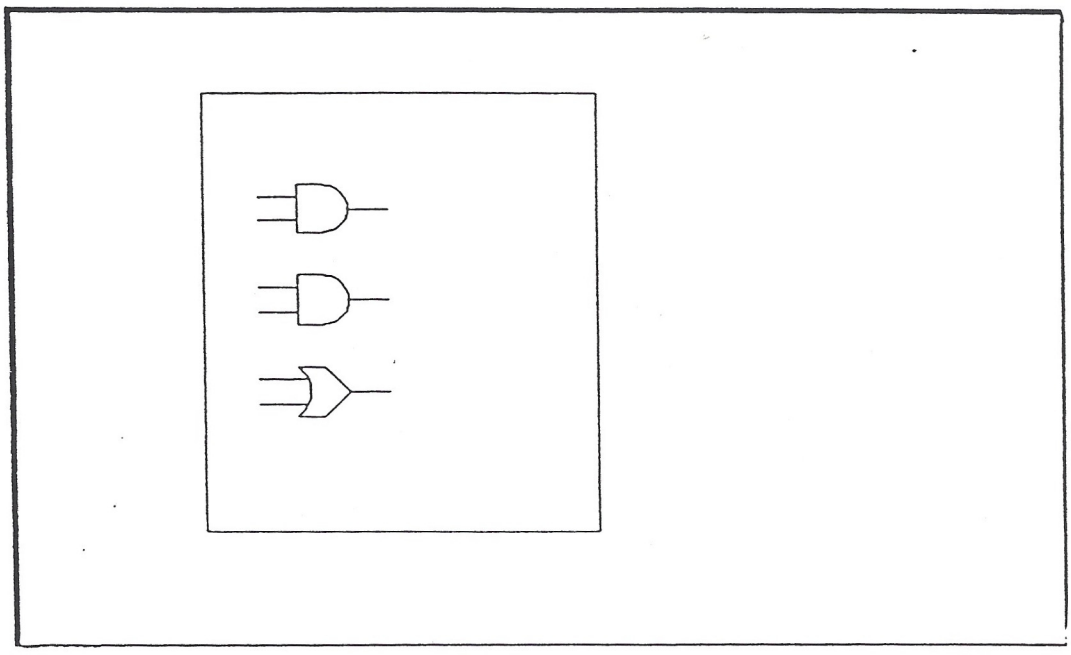


Fig. 2.5

: >>display 3 levels of b2 (39)

[*b2 with all its parts and the parts of its parts are displayed. In other words, this time the parts of and-1 are displayed. Fig. 2.6 shows this.*]

: >>clear the screen (40)

DONE

: >>fill 100 100 300 300 with and-1 (41)

[*Fig. 2.7*]

DONE

: >>clear the screen (42)

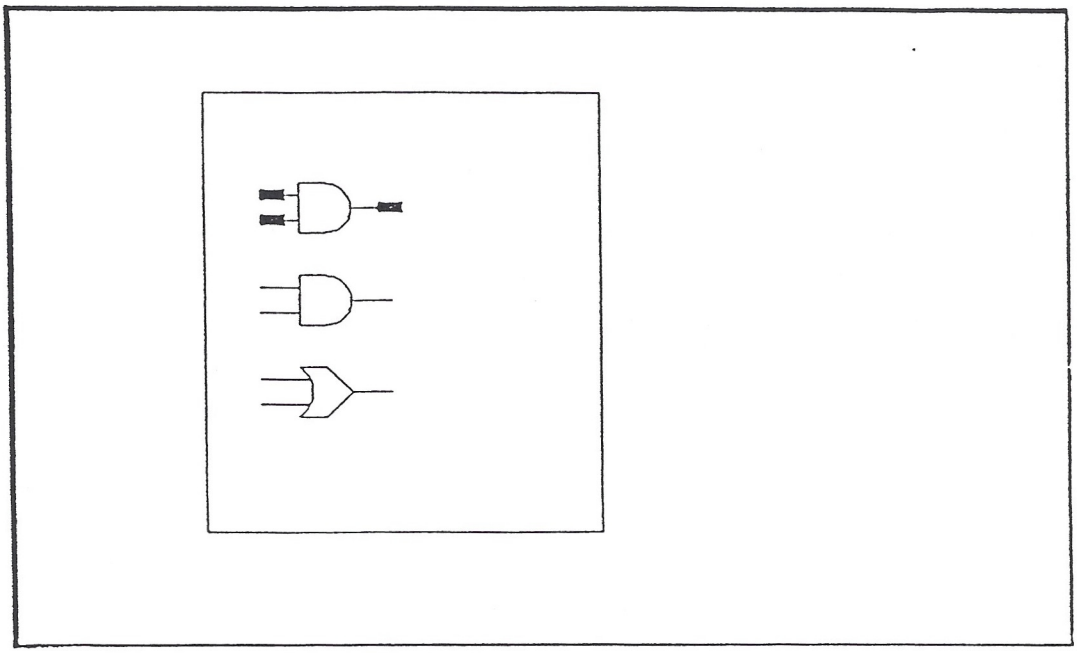


Fig. 2.6

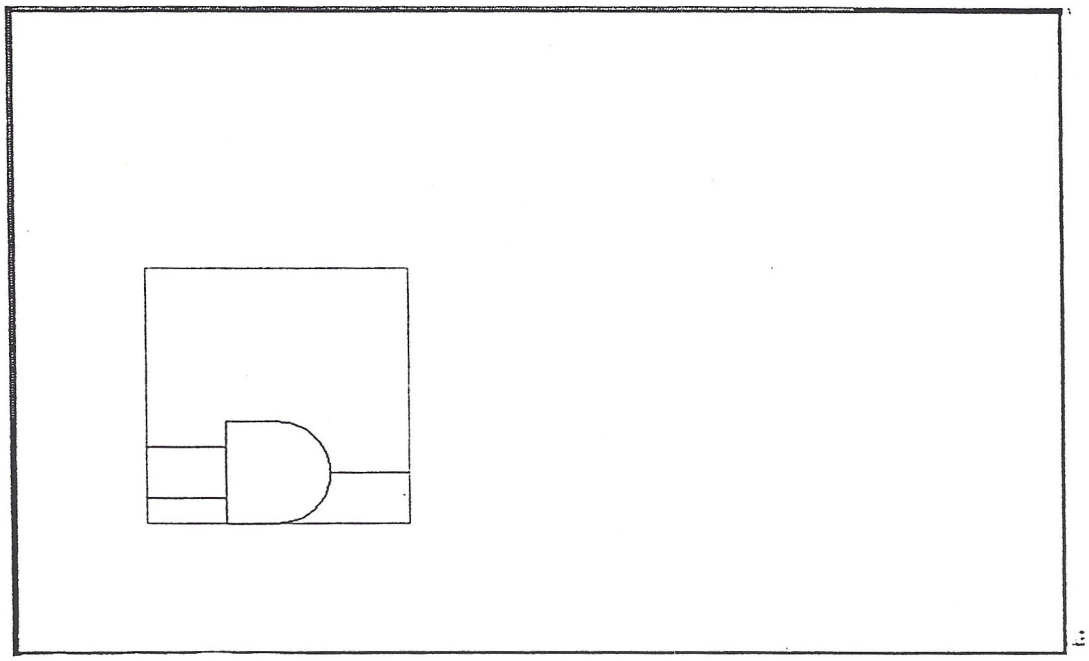


Fig. 2.7

DONE

: >>show and-1 with its environment (43)

[*Fig. 2.8*]

DONE

: >>clear the screen (44)

DONE

: >>what are all the parts of b2? (45)

(AND-1 AND-2 OR-1 PORT-1 PORT-2 PORT-3)

: >>the form of and-11 is xand (46)

PARSED

: >>the form of my-or-gate-99 is xor (47)

PARSED

: >>the form of the-second-not is xnot (48)

PARSED

: >>and-11 is at 3 4 inches on the screen (49)

PARSED

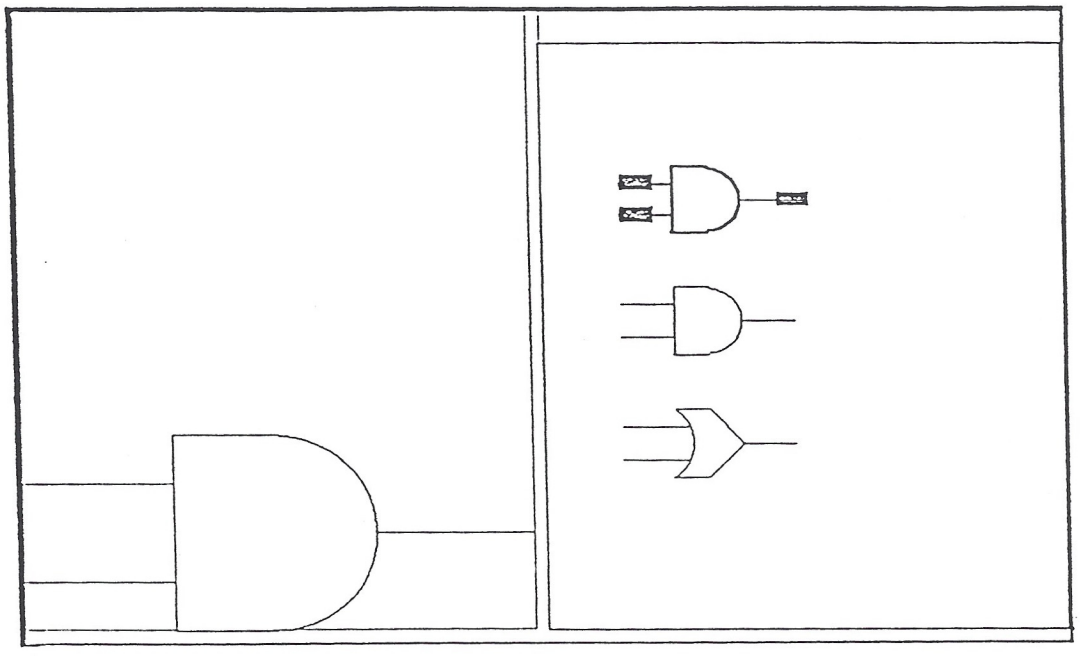


Fig. 2.8

: >>my-or-gate-99 is left of and-11 (50)

PARSED

: >>the-second-not is above and right of and-11 (51)

PARSED

: >>show my-or-gate-99, and-11 and the-second-not (52)

[*Fig. 2.9*]

DONE

: >>clear the screen (53)

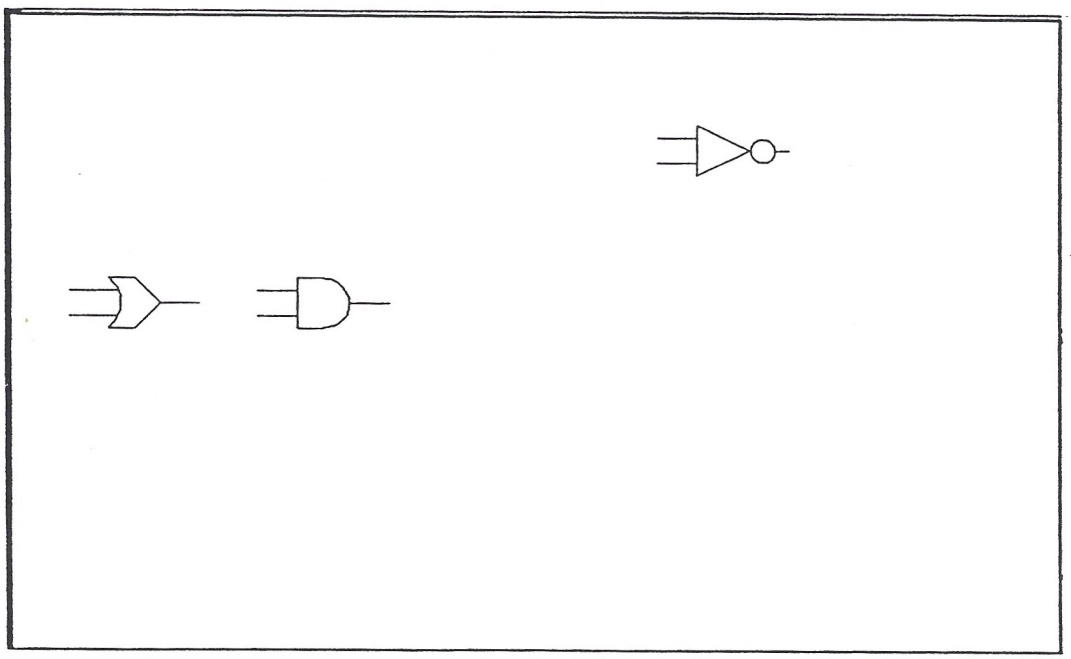


Fig. 2.9

DONE

: >>fill 400 100 600 200 with and-1 (54)

DONE

: >>where is and-1 (55)

(219 292 " relative to" SCREEN-CENTER)

: >>where is the picture of and-1? (56)

[*Fig. 2.10*]

(460 180)

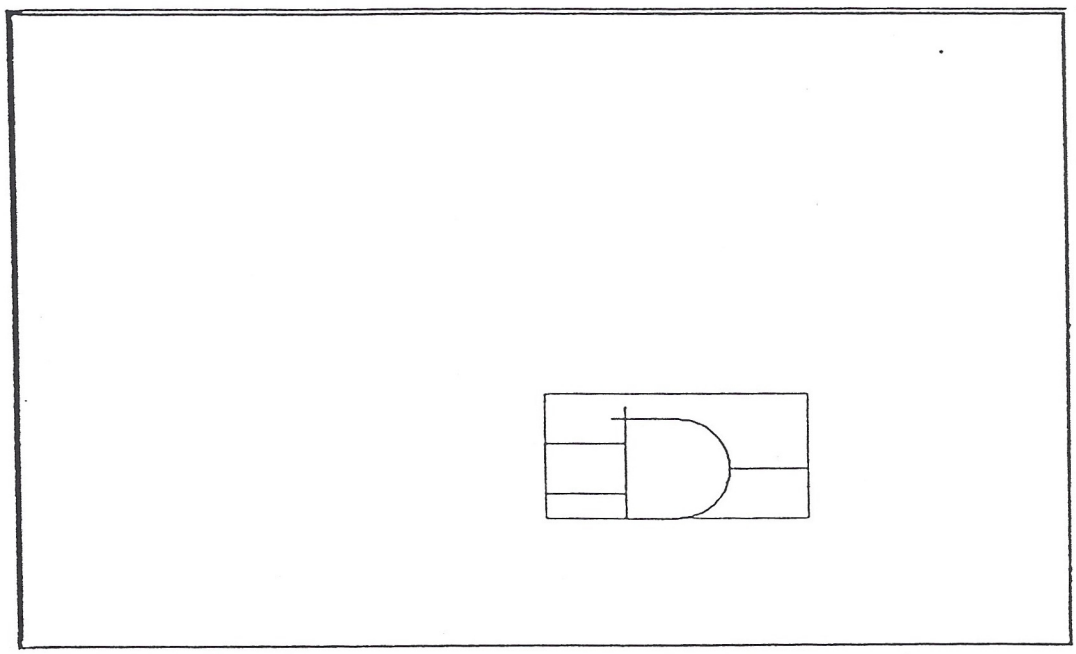


Fig. 2.10

: >>the form of icon1 is form-1 (57)

PARSED

: >>icon1 is at 200 300 on the screen (58)

PARSED

: >>the form of icon2 is form-2 (59)

PARSED

: >>icon2 is behind icon1 (60)

PARSED

: >>show icon1 and icon2 (61)

[*Fig. 2.11*]

DONE

: >>clear the screen (62)

DONE

: >>the form of model-4 is xand (63)

PARSED

: >>model-4 is at the top of the screen (64)

PARSED

: >>show model-4 (65)

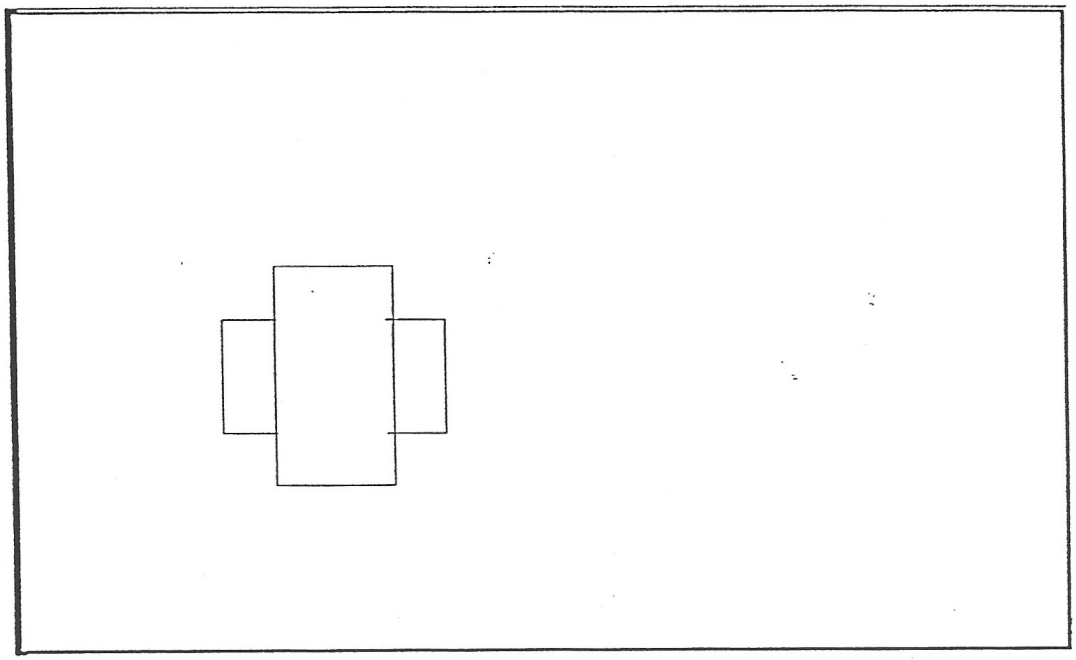


Fig. 2.11

[*Fig. 2.12*]

DONE

: >>clear the screen (66)

DONE

: >>another-and is at 2 2 inches on the screen (67)

PARSED

: >>the form of another-and is xand (68)

PARSED

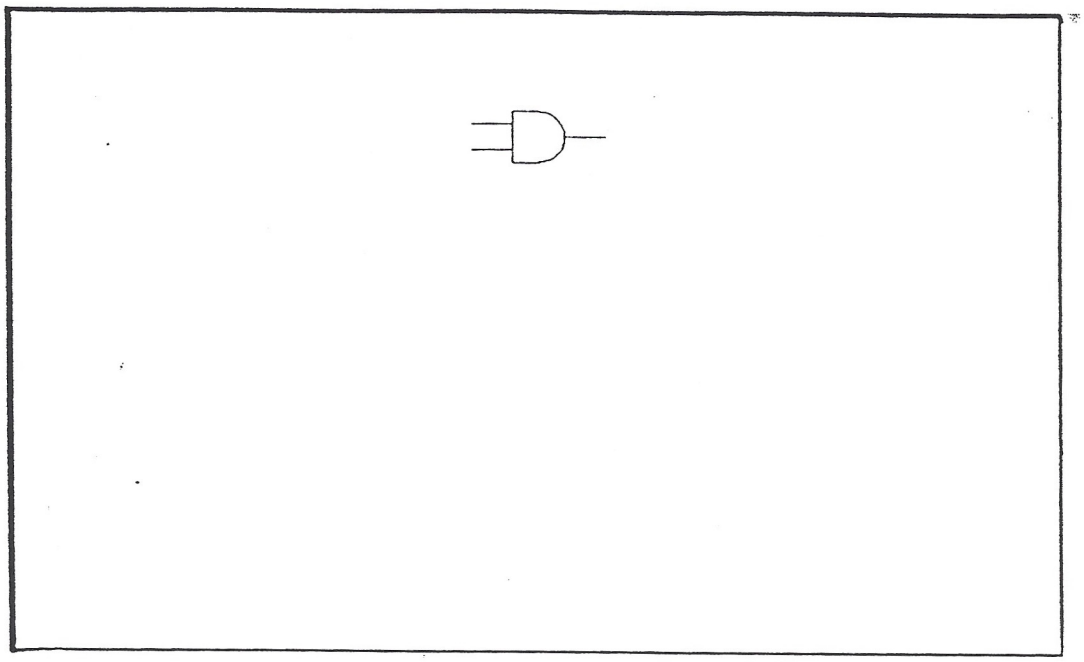


Fig. 2.12

: >>port-1, port-2 and port-3 are parts of another-and (69)

PARSED

: >>port-1 and port-2 and port-3 are members of port (70)

PARSED

: >>port-1 is at -30 -5 relative to another-and (71)

PARSED

: >>port-2 is at -30 -25 relative to another-and (72)

PARSED

: >>port-3 is at 60 -15 relative to another-and (73)

PARSED

: >>the form of a port is xport (74)

PARSED

: >>display 2 levels of another-and (75)

[*Fig. 2.13*]

DONE

: >>the size of another-and is large (76)

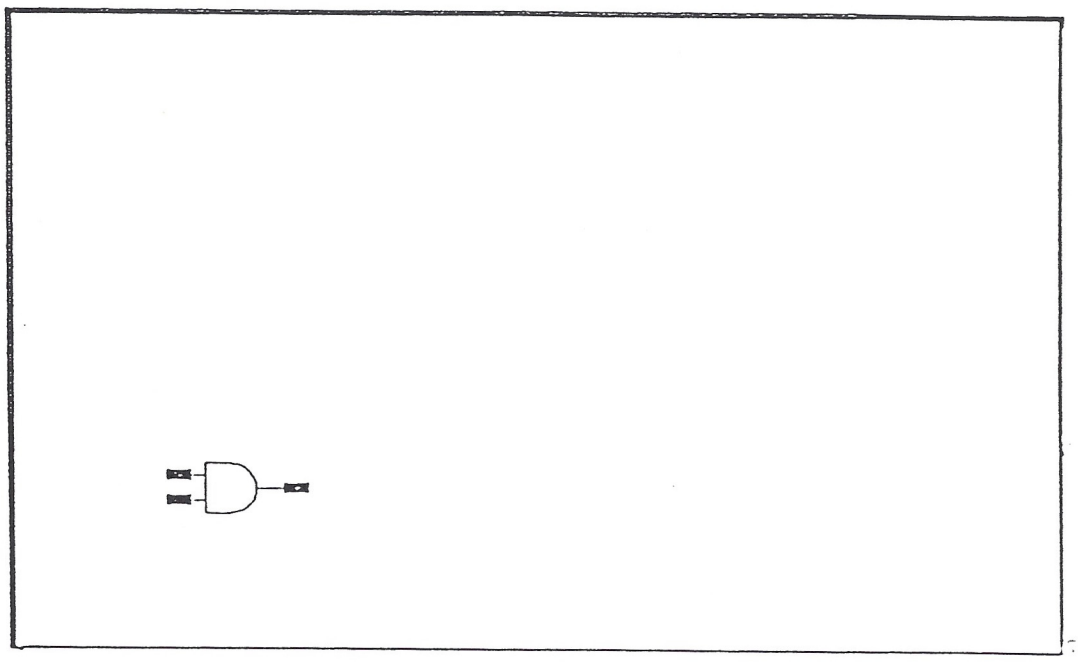


Fig. 2.13

PARSED

: >>size is expressed by scale and 2 represents large (77)

PARSED

: >>clear the screen (78)

DONE

: >>show 2 levels of another-and (79)

[*The somewhat surprising result is due to the fact that the system does not know that the size attribute should be inherited. Fig. 2.14*]

DONE

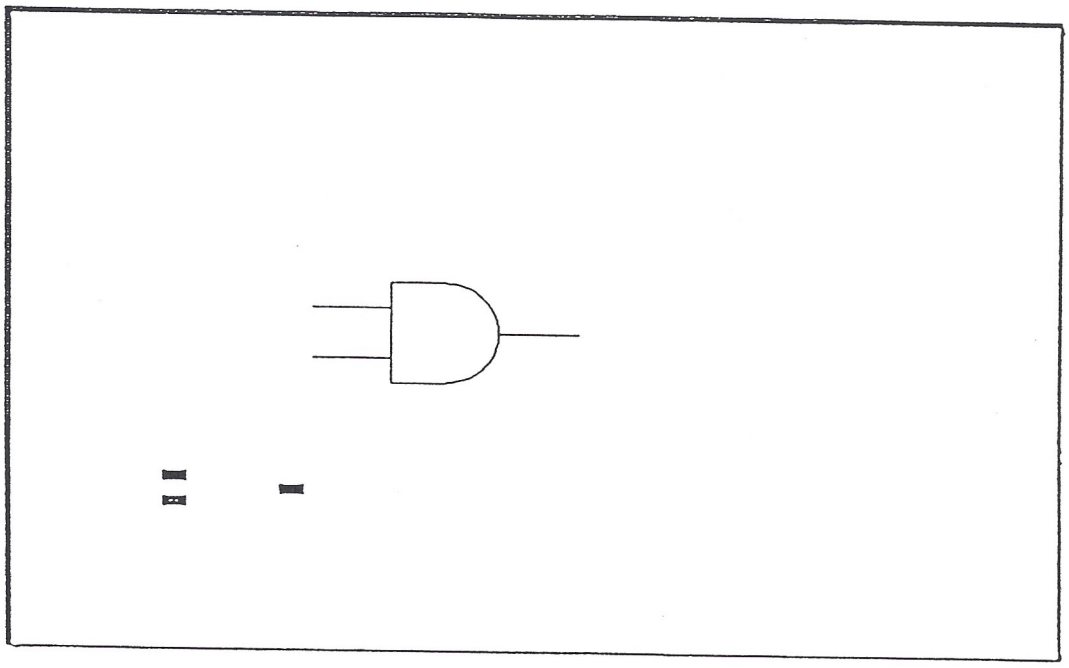


Fig. 2.14

: >>size is inheritable (80)

PARSED

: >>erase the screen (81)

DONE

: >>show 2 levels of another-and (82)

[*Now this is what we wanted! (Fig. 2.15)*]

DONE

(1) associates "ship-1" as an individual with the class "ship". At this point in time the system has never before seen either one of the two words "ship" and "ship-1". (2) assigns a form to the class "ship". For this example dialogue we assume that "ship-form" is a known concept with an associated icon. (3) assigns a position to ship-1. (4) requests to draw ship-1. Note that besides the prior defined icon the system had no other information whatsoever about ship-1, and all information necessary for drawing is completely derived from the knowledge that was built into the system by natural language. Note also that the form for ship-1 was derived by inheritance from the class "ship".

(6) assigns a state to ship-1. As before, these are completely new terms to the system. Neither "state" nor "non-operational" are known in the knowledge base. However, the format of the sentence permits to interpret "state" as an attribute class, and "non-operational" as an attribute value. (7) now binds the "invisible attribute" state to a graphically realizable attribute "rotation". It also maps the attribute value "non-operational" into a parameter for the graphics function that has been predefined and is stored in the function cell of the concept "rotation". (8) requests redisplay of ship-1. The new display shows ship-1 with a changed icon. This demonstrates that the state attribute is correctly interpreted in a graphical manner.

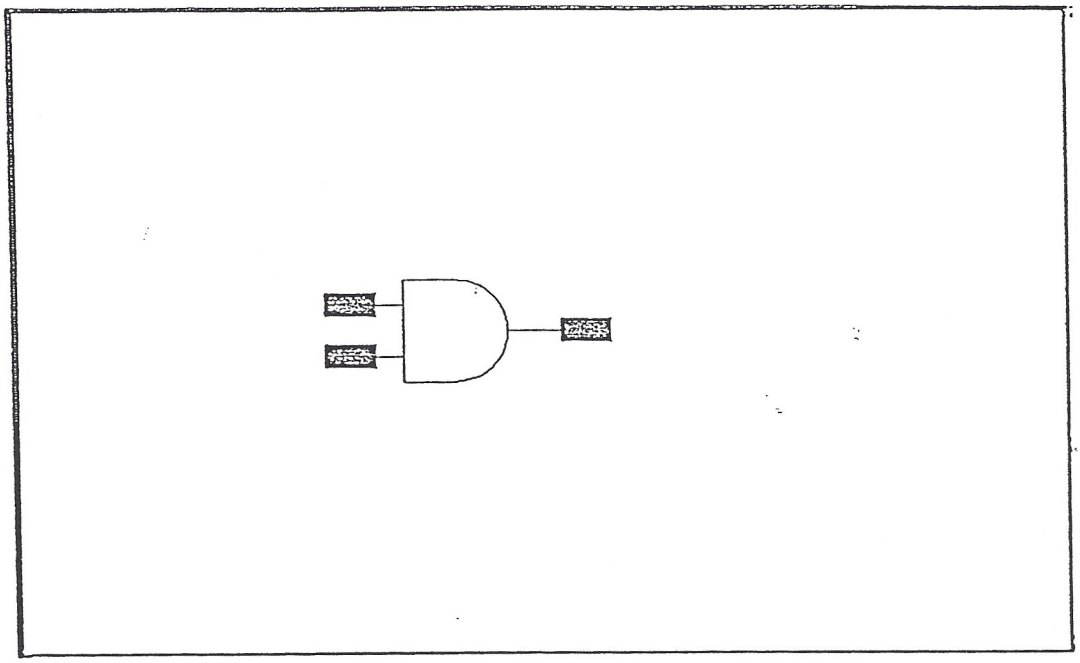


Fig. 2.15

(9) shows that the information passed on to the system by natural language is accessible in the knowledge base and can be queried. In this particular case the form is retrieved and returned by the system. (10) demonstrates that the same is true for attributes, (11) that it is true for positions, and (12) that the members of the class hierarchy built can be retrieved also. (14) adds a member to an existing class. (15) shows simple interaction between language and pointing, and (16) verifies the correct extension of the given class.

(17) - (19) demonstrate a limited ability to do hypothetical reasoning. (20) - (35) build up a little board consisting of an object b2 that has three parts (asserted in 23) and one of the components (and-1) has itself three parts (asserted in 31). (37) shows a display command that uses the level as a selection mechanism what to show. Because only two levels are requested, the parts of and-1 are not displayed. (39) in contrast is a complete display, including all three levels of the part hierarchy.

(41) demonstrates the possibility to display an object in a sub-window, such that it is stretched to optimally fill this sub-window. (43) shows an object in a similar sub-window, and also shows the device where the object is located in a different window. The connection between the two windows is created by "thickening" the corresponding object in the overview window. (45) shows the ability of the system to do reasoning based on parts. This was implicitly already demonstrated with the previous display commands.

(46) - (52) demonstrate the ability of the system to represent fuzzy positions and to instantiate them with reasonable values. (54) - (56) show that the system can discriminate between icon positions and object positions. (57) - (61) show an example for the use of "2 1/2 - d" representations. Although the underlying coordinate system is two dimensional, the system is nevertheless able to operate with an expression like "behind", and to achieve correct display by selecting the order of display. (63) - (65) show "absolute" fuzzy positions which are interpreted relative to the screen. (67) - (74) creates another and-gate with three ports as parts. (75) displays it. (76) - (77) assigns an attribute of "size" to the new and-gate and defines that "large" is expressed as a scale operation. (79) shows a redisplay. Because the system does not know that size is an attribute that should be inherited to parts, the picture shows the parts at their original size. Scaling is done relative to the center, therefore the scaled object is also relocated, but the parts are at their original position. (80) asserts the inheritability of "size", and (82) reactivates display, resulting in a correct diagram.

In summary the above dialogue presents a system that interactively builds a propositional knowledge base and immediately makes use of it for question answering as well as for display purposes.

CHAPTER 3

WHAT PHILOSOPHY HAS TO OFFER TO NLG

Say unto wisdom, Thou art my sister; and call understanding thy kinswoman.

Proverbs, 7:4

3.1. Grice's Maxims of Co-operative Communication

One of the most outstanding achievements in the field of pragmatics has been the philosopher Paul Grice's [Gri75] formulation of "the co-operative principle" which he has explicated as a set of nine maxims, sorted in four groups. This system of maxims has been the basis for criticism as well as for much work in pragmatics, but it has not yet been challenged by any alternative system. In this chapter we will present a short overview of Grice's work and then show that it is applicable to our own research. We will first list the nine maxims in their appropriate categories.

Maxims of quantity:

- Make your contribution as informative as required.
- Do not make your contribution more informative than is required.

Maxims of quality:

- Do not say what you believe to be false.
- Do not say that for which you lack adequate evidence.

Maxim of relation:

- Be relevant.

Maxims of manner:

- Avoid obscurity of expression.
- Avoid ambiguity.
- Be brief.
- Be orderly.

In the following sections each one of these maxims will be analyzed concerning its applicability towards graphical representations and what, if any, contributions it supplies for the construction of NLG systems. Many of the examples given will be derived from the domain of circuit board display and maintenance which has been the driving force behind much of this research.

- **Make your Contribution as Informative as Required**

One major attribute that differentiates language from graphical displays is that language is transient and sequential. Only one word can be spoken at a time, and after it was spoken, it is not accessible any more. The opposite is true for graphical representations. One screen can be split into several windows, and each one of them can represent several propositions simultaneously and statically. There are at least two ways to make use of this facility, and they will be presented as two sub-maxims of Grice's first maxim.

Sub-maxim 1: Show all pertinent views of the same phenomenon.

For the display of a circuit board a user might be interested in showing a physical representation of the board, or a functional diagram. In a physical representation the actual location of components on a circuit board is shown. This is helpful in identifying the location of a component on the board itself. In a functional representation components are arranged to express the overall functional structure of the displayed device. For instance in a situation where a technician tries to identify the function of an apparently damaged component on a board, the first Gricean maxim is served best by displaying functional and physical representation at the same time. The physical representation can be used to derive a component name or position number which then acts as an index into the functional representation.

Sub-maxim 2: Show a complete view of one phenomenon.

When dealing with a complicated system, a person normally concentrates on a single component or a small group of components. Nevertheless it is helpful to know where this component or group is located, relative to the overall system. This can again be achieved by using two windows, one that shows the component or group in question, and another one that shows these components in a larger surrounding. By using a technique like blinking or highlighting, the common components in both windows can be made easily identifiable.

A similar technique has been used in map creation, where maps of a small country often are augmented by an insert showing a whole continent and the country itself highlighted in this insert.

- **Do not Make your Contribution more Informative than is Required**

Paper representations often suffer from the problem of showing *too much* of a system. The dynamic qualities of computer systems make it possible to create completely new reduced views that do not overburden the viewer with unwanted information. There are several approaches to "not making one's contribution more informative than required".

The first approach which is standard in computer graphics and window systems deals with purely geometrical features and selects a certain viewing area. Changing this area is referred to as zooming or panning. Another approach selects parts to be shown according to the part hierarchy of the displayed object. This requires an explicit part representation as maintained by the TINA system. The selection can be done by showing only one or two or a few levels of this hierarchy. If the system has an element/connection structure (like graphs, semantic networks, or circuit boards), then one might want to display all and only the elements that are connected directly to a certain user selected focus component.

Another selection method is to display all and only the components of a certain class, like leaf nodes in a graph, molecular nodes in a semantic network, or transistors in a circuit board. This requires the existence of a class hierarchy in the knowledge base of the representation system.

A fourth selection method is to display all and only objects with a certain attribute. This requires the explicit representation of attributes, as well as retrieval mechanisms to select components based on their attributes.

- **Do not Say what you Believe to be False**

This maxim does not (yet?) apply to computer communication, although the notion of "bias" has been mentioned in the AI literature [Jam83]. The basic idea of biased information is that in a situation when a user expects an exaggerated claim (like from a sales man) he might draw wrong conclusions if he is *not* presented with such an exaggerated claim.

- **Do not Say that for which you Lack Adequate Evidence**

An application for this maxim arises if one uses a program like the "Multiple Belief Reasoner" (MBR) [MaS83]. MBR makes a difference between established facts and hypotheses. If this program is used for hardware maintenance one could follow its activities by displaying propositions graphically. Doing this, the representation of a hypothesis should differ from the representation of an established fact.

For instance in doing maintenance on a circuit a reasoning system will create different hypotheses about which parts are faulty. If faultiness is shown by a signal color like red, then one might use a different line style to distinguish established faultiness from hypothesized faultiness.

- **Be Relevant**

Grice in his original treatment of the pragmatic maxims noted in the section on relevance that "I find the treatment of such questions exceedingly difficult..." [Gri75, p.16].

Presenting examples of relevance that relate to graphical representations is correspondingly difficult, maybe because relevance really should be seen as a super maxim to the other maxims. One way to convince oneself of this fact is to replace the names of the first two maxims by two other names. One can replace the first maxim by "say all relevant things" and the second maxim

by "do not say irrelevant things". In short one can summarize these two maxims as "say all and only relevant things". Something that is not true is necessarily irrelevant, and in the same way one can capture all the maxims below as being variations of the relevance super maxim.

As a side note we would like to mention that the term "relevance" has been used heavily in the AI literature in a number of different meanings some of which are not related to Grice's sense at all. [AnB75,MaS83], [Car70], [Kad86], [Sal72,Sal73], [MiR85]. In Grice's sense the term has been used for example in [SpW86] and [Her86].

- **Avoid Obscurity of Expression**

This maxim is explained by Grice with an example of intended obscurity where two parents talk in a way to prevent a present child from understanding them. From this maxim one can derive the requirement to use standard symbology whenever it exists.

Looking at this maxim in a language context, one finds that in order to be obscure or not obscure one first needs the power to generate all required information in different ways. At the current state of the art in language generation one is usually happy if one can generate a sentence in *one* syntactically correct and semantically coherent way. In other words, one does not have a choice *how* to do something, if one is barely able to do it all.

Nevertheless it helps exposing how far AI is away from its ultimate goals, by imagining how a graphics system *could* be obscure if it should wish to do so. A display program might start to use a font like Old English for text, or put superfluous ornamentation around the lines it is drawing. Behavior like this belongs, as of this writing, into the realm of science fiction authors.

- **Avoid Ambiguity**

In the context of our theory we permit the use of colors to symbolize attributes of objects. If a system has only a limited set of colors available for its task, then a user might be forced to assign the same color for two different types of attributes. This will introduce ambiguity for the viewer. From this one can derive a requirement to use hardware that has a wide range of

expressive facilities, like many colors, blinking, different line styles, different line widths, etc. and to make full use of all the facilities given.

An aspect that makes diagrammatic representations easier to disambiguate than natural language is that they usually comprise some form of *designed* language as opposed to a natural language, and designed languages are less susceptible to the ailment of ambiguity. In addition to that, graphical languages do not suffer that much from the need of natural language to be concise because there are two dimensions of representation available, while language is linear in nature. This eliminates another major source of ambiguity.

- **Be Brief**

Like for the "avoid obscurity" maxim, this maxim makes only sense for a system that has *the ability* to behave in two different ways, in this case to be brief or not to be brief. A graphical analog of a behavior that does not observe this maxim would be if a program draws solid lines by chaining dots even if it has a "line" primitive available. At the current state of the art, AI programs usually do not have the competence to make such decisions. The programmer will hopefully ensure that the display will be constructed according to the briefness maxim.

- **Be Orderly**

The concentration in this section is on the literal meaning of the maxim "be orderly" namely on doing things in the right order. In graphical representations there is almost no limit to the number of different orders in which parts of a picture can be drawn. If there are n icons, there are n! orders of drawing. This differs from natural language (at least English) where there is basically a fixed word order given.

If one takes a piece of text and scrambles it, the understandability of the whole structure will be severly impeded. As opposed to language, parts of a graphics display are not transient, but stay around after initial creation, so that the final result does not depend on the (temporal) order of presentation (except for overlays). It is, however, clear that presentation order can be helpful in

several ways. This will be demonstrated again with examples from our domain of circuit board maintenance.

(1) If a component is found faulty by a maintenance system, then it is of interest to display this component first. In this way necessary measures can be initiated by the maintenance technician, without waiting for the complete display to come up.

(2) Circuit boards often have a structure consisting of functional units that are interconnected. In a functional display the components of these functional units tend to be organized as clusters. One way of creating such a diagram orderly would be to outline the overall structure first by empty boxes such that one box represents one functional unit, and then continue by replacing or filling these boxes. Such a "black box", structure is often used in addition to a functional diagram, and the dynamics of a graphics system permit to integrate both diagrams by the representation order, leading the viewer from the overall structure to the detail representation. Clearly this method can be applied to any structured representation, not just circuit boards.

(3) Assuming a structuring of a system into black boxes, one can then raise the question in what order to fill them with or replace them by components. One possibility is to explicitly represent the attribute of importance in combination with our representation of comparatives and superlatives. If importance information is given, then drawing should be done starting with the most important component. Again, this method can be applied to other diagrammatic representations also.

(4) When filling in components in a black box, after establishing a main part, electrical connectivity should control the farther development of the picture. It is much harder to follow the process of drawing a picture if the location of current growth is changing in an arbitrary manner. ("Don't jump around while you are drawing.") Components should be added in a natural order of connectivity. Using connectivity there are at least three different ways to advance. One way is to follow the signal flow through the system. The next possible

approach is to follow a policy of maximum connectivity. This policy would add a component which connects to the maximum number of pending wires. It seems desirable to use this last policy only if the signal flow does not give enough information how to continue. Finally one can grow in a breadth first manner from the main object. This is certainly the least desirable method, but it is still better than random growth. These ideas can be applied to any node/link system, like to semantic networks, flow charts, or syntax diagrams.

With the availability of high speed frame buffered graphics terminals drawing speed has become less of interest. However, AI programs are notoriously slow, and while the graphics hardware might be able to display a whole complicated device in 1/60 of a second, the underlying program that creates the necessary picture might be so slow that the discussed ideas about ordering become important.

Concerning the first presentation of a complicated diagram, we are inclined to claim that the slow serial presentation in an *intelligent order* can be seen as a feature, not a problem. Diagrams, as opposed to text, do not impose a clear order of reading. If the system draws the diagram according to an importance order as opposed to "left to right, top to bottom", it will help the reader to understand the "inner logic" of the displayed mechanism.

3.2. A special Maxim for Technical Languages

In comparing natural language with diverse symbolic graphical languages a new pragmatic maxim was discovered that applies specifically to scientific/technical languages. This maxim is called the *maxim of unnecessary variation.*

The Maxim of Unnecessary Variation:

Do not represent two objects or two relations differently, unless you want to express a difference between them.

This maxim comes in two versions, as unnecessary intra-variation and as unnecessary inter-variation.

Intra-variation refers to differences between two representations that occur in the same narrow context, usually on the same screen or page.

Inter-variation refers to differences between two representations of the same well defined class of representations, however in different contexts.

The first Sub-Maxim of Unnecessary Variation (Consistency):

Avoid intra-variation at all costs.

The second Sub-Maxim of Unnecessary Variation (Conservativity):

Avoid inter-variation unless you have a good reason to introduce it, i. e. follow the conventions established for this class of drawings.

Some examples will elucidate these sub-maxims and show that they present a very reasonable choice in dealing with symbolic graphical representations.

It is an explicit style guideline for drafting of circuit board diagrams that one should not represent transistors/tubes in the same drawing by more than two different sizes [Ren71]. We claim that one should draw all transistor symbols on one page in the same size. Not doing so would create unnecessary intra-variation that would be confusing. The less knowledgeable technician might suspect that the different sizes express something, maybe different power consumption.

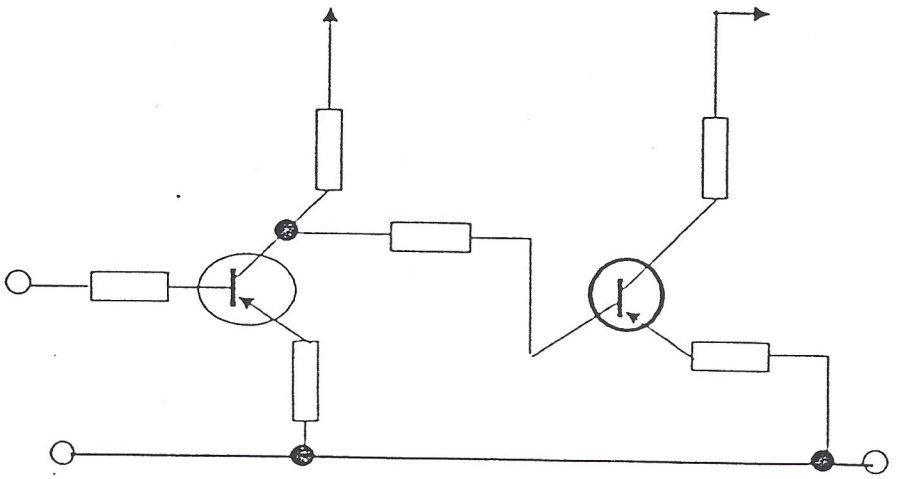


Fig. 3.1: A transistor amplifier with unnecessary variation.

But even the experienced technician will probably feel irritated by this unnecessary variation. The actual size of the transistor is limited by practical considerations like visibility on one hand and space consumption on the other hand. Fig. 3.1 shows two identical cascaded amplifiers, however unnecessary variation makes it hard to understand the diagram. Biesel has reported the same effect [Bie84], however without analyzing the pragmatic reasons for it.

A good example for inter-variation can be derived from another representational system, namely from function graphs (Fig. 3.2). There is no reason why the coordinate axes of a function graph have to be drawn parallel to the page the drawing is done upon, with x pointing to the right. Nevertheless anybody that breaks this *convention* has to have a good reason, for instance he might need a very long x axes, so he could switch to the format in Fig. 3.3. Anybody using the format in Fig. 3.2 where the x-axes is tilted against the page boundary by approximately 45 degree will engender utter protest from his colleagues, although the figure itself is presumably correct.

Conventions of the described nature are usually not arbitrary but often help to eliminate unnecessary perceptual features. For instance there is a strong convention not to have any connections cross any components. In semantic networks this means that arcs should not run over nodes, and in circuit representations it means that wires should not run over transistors or any other

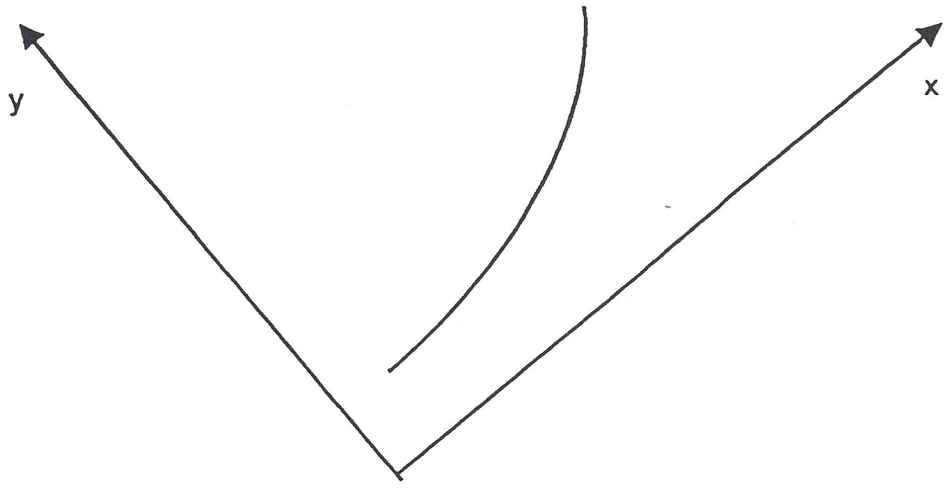


Fig. 3.2: A Function Graph with unnecessary variation.

components. Mutual intersections of connections are usually not avoidable, but they add only one feature. Intersections of wires with components add at least two features, namely entrance point and exit point. In addition, intersections are perceptually more significant than e. g. parallel lines, assuming line segments of equal length [Wal87].

One can demonstrate the effect of unnecessary variation with a text written with a printer that constantly changes the font used. Meaningless variation is added to the text which has a distracting effect on the person trying to read it. For the reader who doubts these claims Fig. 3.4 contains an example of a font cataclysm as test reading.

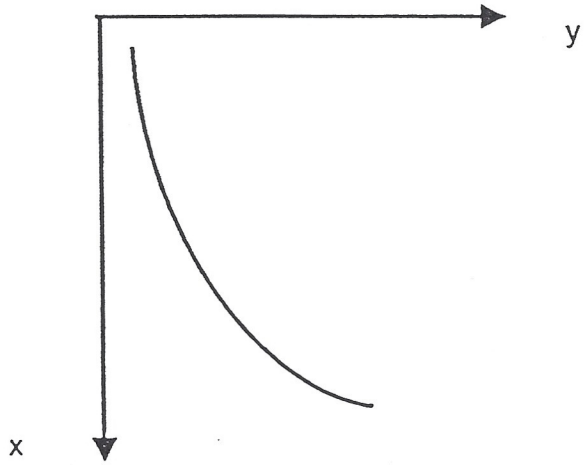


Fig. 3.3: A function graph with an acceptable layout.

The MAXIM OF Unnecessary Variation
is demonstrated by adjoining *text*
of several fonts **which we claim is**
hard to read

Fig. 3.4: A font cataclysm.

3.3. Feature Analysis of Symbolic Graphical Representations

In the previous section perceptually accessible high and low level features of graphical representations have been categorized implicitly into three different classes. (An example for a high level feature would be overall arrangement and an example for a low level feature would be line intersection.) These feature classes will now be discussed explicitly.

Semantic Features.

A perceptually accessible feature is considered "semantic" if any change of an instance of it would result in a change of meaning for the picture containing it.

Pragmatic Conventionalized Features.

A perceptually observable feature is considered a pragmatic conventionalized feature if a change of it would not change the meaning of the picture, but there are well established guidelines how this feature should be selected. An important aid for recognizing conventionalized features is that one can usually verbalize what the convention is ("draw the x-axis parallel to the page boundary").

Accidental Features.

A perceptually observable feature is considered an accidental feature if a change of it would not change the meaning of the picture containing it, and where a range of acceptable values for the feature exists.

It should be noted that accidental features can vary between different contexts, but not in the same context; this is due to the maxim of unnecessary variation. Pragmatic conventionalized features should not even vary between different contexts.

Unfortunately, designers of technical graphical languages virtually never make the effort to compile a list of perceptually recognizable (high and low level) features of their representations and to categorize them into semantic features, pragmatic conventionalized features, and accidental

features. By **not** supplying such a list designers imply that the only way to acquire their languages is by example, and one would rather have a formal semantic specification for a formal language which then may be augmented by examples.[1]

The point being made here is important for an NLG system, because it will (at the current state of the art) **not** acquire the meaning of e. g. circuit board diagrams by example. If we want the system to share the knowledge with the user that a line intersections is electrically significant (or not!) we have to worry about representing this fact explicitly.

One can add a fourth redundant list to describe a graphical language. Due to the reductionist character of representations which applies to most technical languages as well as to knowledge representation systems (we will return to this observation), every domain will have a number of features that are not represented by any perceptual feature of the technical language. Although it is usually impossible to supply a complete list of such features, it is desirable to mark some of them as not represented, if the danger of wrong interpretation exists. In other words, if there are features that are not represented, but somebody might think that they are represented, it is beneficial to state *explicitly* that they are not represented. This is done for instance in some maps of the solar system that state explicitly that planets are not drawn to the same scale as the rest of the map. So the feature of planet size is not represented.

Semantic features, i.e. features that cannot be changed without changing the meaning of the display, can be either representative or symbolic features. If a yellow object is displayed yellow, then the color is a representative feature. If a faulty object is (always) presented in red, then the color is a symbolic feature. In the more common case of a symbolic feature a responsible display designer should supply an annotation explaining the corresponding domain feature.

We do not expect that from today on every e. g. electrical circuit diagram will have four such lists of features associated with it. This would be unnecessary and take more space than the

---

[1] It has puzzled me since my days in engineering school that nobody made an attempt to teach the language of circuit boards in a formal way.

diagram itself, and it would not even help a beginning "circuit diagram reader". However, two environments where such lists would be immensely helpful have already been suggested: in *teaching* a graphical language, and in designing an NLG system. The latter is the reason why we have developed this whole theory.

One could raise an objection that listing accidental features is never necessary, because one can use a closed world assumption, declaring that every feature that does not show up in the list of semantic or conventionalized features is accidental. This of course presupposes that one has compiled two lists of all these other features without forgetting one. It is therefore preferable to describe salient accidental features also, because a reader (viewer) might suspect that a certain feature was only forgotten in the other lists. Should a user ask an NLG program about the meaning of an accidental feature, it would be beneficial if it is known as such.

In order to exemplify the descriptive methodology suggested here, a number of graphical representation systems will be reviewed. Before getting to this, an overall list of features will be presented that has been found useful. This will shorten the detail lists for the different representations.

3.3.1. Features Common in Symbolic Representations

A basic distinction will be made between *icons* which usually have a fixed form, OR appear as a closed figure, and *lines* which are open and used often to connect icons. The third type of elements permitted are descriptive labels. The following is a list of perceptually accessible low and high level features that we have found useful in describing graphical representations.

- Form of the icons used.
- Connectivity between icons by an uninterrupted line that impinges on two or more of the icons.
- Line style (bold, dotted, etc.).
- Simple line intersections.

- Line intersections with marking points or avoidance arcs.
- Size of icons used.
- Relative spatial arrangement. This is very general and subsumes some of the other features.
- Orientation of icons. This is independent of the relative positions.
- (Prohibited) crossings between icons and lines.
- (Prohibited?) overlay of icons.
- Density of the icon distribution in the given space.
- Overall arrangement of icons describing a flow from left to right or top to bottom (sometimes with a few exceptions).
- Proximity association of labels and icons.
- Proximity association of labels and lines.
- Containment association of labels and icons.
- Endmarker of lines (e. g. arrowheads).
- Fonts of labels.
- Relative positions (i. e. distances, as opposed to arrangements).
- Area color of icons.
- Boundary color of icons.
- Hatching of icons.
- Angle.
- Line orientation.
- Scales (linearity or not).
- Containment of icons.
- Permitted overlay of icons (by assigning the overlay area a different color or hatching pattern than the separate icons).
- Intersections. This has several sub cases already mentioned before.
- Touching relations between icons.

In the following sections the occurrence of above features in different graphical representation systems will be investigated. We will start with a look at (what else?) graphical displays of circuits and a class of semantic networks that will be heavily used in later chapters, so called SNePS networks. For these representational systems semantic features, pragmatic conventionalized features and accidental features will be given. If appropriate some features that are not represented will also be mentioned. The reader will notice soon that there are a number of "borderline" cases. Also semantic features are sometimes negative, in that a certain representational feature is never used in a representational framework. In that case the inclusion of that feature would create doubt whether one is in fact dealing with a wire plan/ SNePS net, etc. therefore they are considered as (negative) semantic features. Also the degree to which variation of accidental features would create confusion, if varied, is different for different features, depending on their perceptual salience.

3.3.2. Logical Wire Plans

Semantic features:

The form of the icons is fixed (specified by technical norms). Every icon form represents a specific type of component. The connectivity between icons expresses electrical connectivity. This applies to solid lines only. Coaxial switches are sometimes marked by connected dotted lines, so the line style used is meaningful. Line intersections that are marked with a little dot express electrical connection between wires. Labels describing components or wires have to be nearer to the described component than to any other component. Wires are undirected, i.e. there are no specific endmarkers; a consistent change of this (negative) feature would make it doubtful whether one is dealing with a wire plan at all. Areas are not colored or cross-hatched, except for some icons that contain small parts filled in with the drawing color. No scale is associated with the picture. Icons are not contained in other icons or overlaid over them.

Conventional features:

Relative spatial arrangements should place components belonging to one functional unit near each other. Icons should be arranged such that the overall flow of the signal processed follows the conventional reading direction, i.e. from left to right and top to bottom. Icons should be in a standardized orientation (upright), or at most rotated by 90 degree or flipped horizontally or vertically. Wires should be horizontal or vertical whenever possible. Crossings between lines and icons are prohibited. Overlaying of icons is prohibited. Overlaying labels with icons is prohibited. Intersections between lines should be avoided as much as possible, but they are meaningless, if they occur. Angles are meaningless, but wherever possible right angles are used (except inside of icons themselves). Icons are supposed not to touch each other.

Accidental features:

Icons should have a constant size, however one has some flexibility in selecting this size. All lines representing wires should be drawn in the same line width. All icons of the same type should have the same line widths used. The same functional unit should be displayed with the same layout pattern. The density of icons in the presentation area should be approximately constant. Fonts are meaningless. Relative distances between components are meaningless. Icons and wires are usually drawn in black or blue (blue prints).

Unrepresented features (selection):

Very little of a real circuit board is preserved. Forms and relative positions of components are not represented, and this excludes most other features.

3.3.3. Physical Wire Plans

Semantic features:

Connectivity is a semantic feature (assuming a single layer board with no wires passing under components, an assumption that will not hold for many circuits). Line style is meaningful, one might indicate wires running under components by dotted lines. Simple line intersections are meaningful. They express real electrical connections. Relative spatial arrangement is meaningful. It means: that's how components are arranged on the board. The same applies to icon orientation and derivatives of it, like density of distribution, overall arrangement, touching relations, and relative positioning. Proximity or containment association of labels with icons and lines is semantic. There are no endmarkers, color is not used.

Conventional features:

The rigid limitations that are the result of the need to correctly mirror a real circuit limits the number of conventions used drastically. Conventions are used where there is a choice, and there is not a lot of choice if one tries a quasi analog depiction.

Accidental features:

The form of the icons used seems to be an accidental feature in the following sense. Two equally looking components (in reality) should look the same way on the plan. However there are no strong conventions, and one might also vary degree of precision of the icons, so they might actually look different on different wire plans. On the other hand, very different components might physically look the same. Size of the icons is accidental, but no variation of scale is permitted. The same applies to line widths.

Unrepresented features (selection):

Colors of components are usually not represented.

3.3.4. SNePS Networks

And thou shalt make for it a grate of network of brass; and upon the net shalt thou make four brasen rings in the four corners thereof.

Exodus 27:4

SNePS semantic networks are used as the major tool of this investigation and will be explained in later sections, where diagrams will also be supplied as examples. They are only included here to maintain the logical structure of this chapter. The reader not familiar with SNePS is invited to skip this section and to return to it after familiarizing himself with the graphical network notation. The word "semantic" does not have the same meaning in SNePS and in the chapter heading "semantic features".

Semantic features:

Connectivity between icons is meaningful. Solid lines express arcs, dashed lines express auxiliary arcs. Labels of lines must be next to the lines (such that there is no other line that they are nearer to). Labels of icons must be inside the icons. Lines must be marked on one end with an arrow head. The icon that the head is pointing at is called a dominated node. Areas are not colored or cross-hatched. No scale is associated with the picture. Icons are not contained in other icons or overlaid over them.

Conventional features:

Icons are circles, ellipsis, or sometimes rectangles. The choice is based on the length of the label associated with it, and all icons are identical in meaning (independent from the form). Intersections between arcs are meaningless, but they should be avoided whenever possible. Icon sizes are meaningless, but the convention is to draw them according to the size necessary for labels. The relative spatial arrangement of icons is meaningless, but it is preferred to have molecular nodes above base nodes. Icons are always horizontal. Intersections between icons and lines are prohi-

bited. Overlays between icons are prohibited. Icons are supposed to fill the given space about equally and not to touch each other.

Accidental features:

Line style is accidental. Fonts are accidental. Relative positions between icons are accidental. Icons and lines are usually drawn in black. Angles are meaningless and can span a wide spectrum of values. This does not disagree with the maxim of unnecessary variation, because the angles are enforced by label sizes and do not give the feeling of being varied randomly.

Unrepresented features (selection):

Given the highly abstract nature of semantic networks and of what they represent, there are very few unrepresented features. However if one wants to talk in terms of an implementation, then the networks do not express the implementation language (e. g. Common LISP versus Franz LISP), implementation technique, or implementation environment (hardware, operating system).

3.3.5. Maps

A specifically interesting representational system for our purposes are maps, because they are a borderline case between symbolic representations that we are dealing with here and analog representations. For instance some maps express a specific three dimensional character by shading. Nevertheless there are some factors of abstraction in drawing maps [Arn86] and many relations are expressed by symbolic icons. The rigid format of feature categorization used in the previous sections will be relaxed at this point, because many of the descriptions are repetitive.

Maps have icons of variable form, namely country boundaries, and normal icons that mark villages, mountain tops, etc. There are no lines in the sense of the other representational formats. The relative spatial arrangements are significant, and so are the orientation of variable icons. Sizes are not preserved. The density of fixed form icons should not become too large. Labels have

to be as near to the described icon as possible, however there are also area describing labels (for mountain chains), and labels may have different fonts for cities, countries and mountains, which permits to relax the distance condition. Fixed form icons are contained in variable form icons. Area color expresses political division or height above zero, or sometimes vegetation or population distributions. Hatching is used sometimes for large cities. Non-political boundary lines are sometimes expressed with a color different from political boundary lines.

Angles are not preserved (like sizes). A scale is given, but certain objects like villages are often exaggerated relative to this scale.

Different types of maps preserve different features, but no map preserves the original size of the displayed objects. Most maps do not preserve any three dimensional structure, but there are some exceptions (for instance globes).

3.3.6. Pie Charts

We will limit ourselves to simple as opposed to hierarchical or exploded pie charts. Such pie charts represent one single numerically measurable feature in an analog way. The icons of pie charts have the form of slices. The angle of a slice in the chart is the meaning-carrying item. Labels must be assigned uniquely to slices. There is no concept of connectivity, intersection or end markers. The relative arrangement of the slices is not meaningful. Different slices are usually marked in different colors or different hatching patterns. The radius of the area is not meaningful, but this is an accidental feature with strong pragmatic limitations, varying the radius would be extremely confusing. Usually no scale is supplied although it would be possible to do so. Icons don't overlay but touch each other along their straight lines. Some pie charts indicate the convention that segments are ordered according to size.

3.3.7. Bar Graphs

Bar graphs are interesting because they behave differently in two different dimensions. The form of the icons used is constant, except for variation of one measure. In this respect there is only one icon used (except for coordinate axes). There are no connecting lines. The line style used is not meaning carrying, neither is the color. Size is interesting because it is a semantic feature in the vertical direction, but not so in the horizontal direction. A scale is used that may be logarithmic instead of linear. It is a strong convention that the vertical direction be parallel to the drawing plane boundary.

Bars may either touch each other, or not, but this must be consistent, a strong pragmatic limitation. In simple bar graphs there are no intersections or inclusions of icons. Labels must follow the proximity condition and are usually of a single font. Icons may be hatched or their areas colored, and if this is done then color is meaningful, such that every different color describes a different feature.

3.3.8. Euler Circles

Icons are all closed shapes with no inner lines, and usually with no corners and approximately elliptic form, referred to as blobs. Sometimes icons are circles or ellipsis. There is no line structure, and labels are usually contained in the icons. Pens are normally not semantic. The size of icons, as well as relative positions are not meaningful and there is no scale given. Icons may intersect, may be contained, and are sometimes shown as overlay. Icons may be colored and or hatched.

3.3.9. Graphs of Functions of one Variable

This representation consists besides the coordinate axes and their labeling of no icons, only of a "line". If several functions in one diagram are displayed, then line style might be used for differentiation. Line intersections are meaningful (for instance the intersections with the x-axis).

Labels have to be near the coordinate axes points they describe. Finally there is a scale given for both coordinates x and y, and they might be different. The coordinate axes are supposed to be parallel to the drawing plane, by strong convention.

3.3.10. Flow Charts

Icons are limited to a few regular forms (boxes, diamonds,...). Lines are connecting icons, there are arrow heads that express a direction of the lines. Pen and width of the lines are accidental features. The size of icons is not meaningful, but they have to leave enough space to contain their descriptive labels. Icons have conventional orientations. Lines may not cross icons, neither may icons contain or overlay each other. Lines have no labels associated, except for TRUE/FALSE (or sometimes yes/no or a small variation of these). Fonts are accidental. Hatching is not used. Color is accidental, but usually black. Lines are preferably parallel to the drawing surface boundaries (if possible). There is no scale specified. Intersections of lines are not meaningful but should be avoided whenever possible. Overall arrow orientation usually goes from top of the page to bottom of the page, this is a weak convention, because loops make it necessary to run backwards.

3.3.11. Syntax Diagrams

Syntax diagrams are quite similar to flow charts in their structure, although they are different in meaning. They differ in that different fonts are often used meaningfully, and that the overall flow is rather from left to right then from top to bottom. The icons used vary slightly, with boxes with rounded edges replacing diamonds as second main icon.

CHAPTER 4

GRAPHICAL DEEP KNOWLEDGE AND INTELLIGENT INTERFACES

One of the most important growth areas of computer science which happens to be directly related to NLG is the design of user interfaces. The sixth (sic!) generation project of the Japanese computer industry [GaS86] is witness as well as a successful journal, the "International Journal of Man Machine Studies", and relevant publications in a number of others. Cognitive psychology, human factors, artificial intelligence, operating systems, and even hardware design (mouse!) have helped in designing better user interfaces.

The user interface is nowadays seen as a major part of every system, because acceptance of a new device is dependent on the quality of the interface. Unfortunately it is not always possible to "program" a success in interface design, and well planned and designed interfaces have suffered utter rejection by the user.

In the recent past the knowledge revolution has been catching up with user interfaces [NeK86a, NeK86b]. It has been argued [All86, ScE86, ShG86b] that knowledge is as necessary for intelligent user interfaces as for any other intelligent system. In this section a few short comments will be made on the positive influence of knowledge based systems on user interfaces. We will preface these with presenting a problem which has almost become part of interfacers folklore.

SYSTEM:	Select one of the following countries: Belgium, France, England, Sweden, Norway.	(4.1)
USER:	Belgium	
SYSTEM:	Invalid Command "Belgium".	

What's wrong here is that the system obviously does not know what it is talking about. It expects probably a command like "select" followed by a country name, or just a position number, but it does not inform the user about this fact. The exact same type of thing can happen with a mixed

graphics natural language interface. Imagine the following dialogue:

		(4.2)
SYSTEM:	[displays Fig. 4.1]. Please select one of the objects.	
USER:	triangle	
SYSTEM:	Invalid input "triangle".	

In this case the system might expect an input by means of a pointing device, but it fails to inform the user about this fact. The word "triangle" is useless to a system that does not know about triangles, only about lines and points.

We will not argue for the advantages of multi-media interfaces, this has been done ably at other places [PAC85,RPK85]. But we will argue that a multi-media interface will be useless, unless the system shares some knowledge with the user about what the user is currently seeing. In other words, it is not enough for the system to draw something, but it has to know what is currently visible on the screen, and it has to know the most important perceptual attributes of objects on the screen that people might use to refer to certain objects.

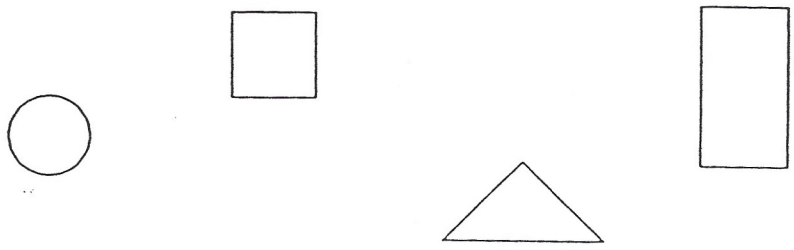


Fig. 4.1: An arrangement of simple geometrical objects.

The approach to permit this common knowledge consists in maintaining separate information on the objects to be displayed and on the pictures that are visible. A user might request the display of an object PCM-BOARD1 in all its details. As will be elaborated later on, the system should react with following a stored part hierarchy, to locate all the parts of PCM-BOARD1, their forms, their positions, their attributes, etc. Then all these parts have to be displayed, and this is how far somebody would go that is interested in a nice display. But it is not enough, because a user might refer to an object currently visible, or even worse, to an object visible on *the last screen*, therefore the system must maintain all this information in a propositional format, so that it can answer questions.

It is clear that knowledge about screen arrangements will not be available initially, and the system will have to create the structures describing the screen by itself. This will permit a user to refer to to the position of an object on the screen, although this position is different from its position in the world and has just been created dynamically. We will discuss the inner workings of this mechanism later on in the section on knowledge compilation (Section 5.1.4).

CHAPTER 5

GRAPHICAL DEEP KNOWLEDGE AND REASONING FOR NLG

5.1. Knowledge Representation

He that planted the ear, shall he not hear? he that formed the eye, shall he not see? He that chastiseth nations, shall not be correct? even he that teacheth man knowledge?

Psalms 94, 9-10

Knowledge representation is widely considered one of the most important fields of Artificial Intelligence, and it has even been called the "glue" that keeps the whole field together [McC83]. In chapter 2, describing a scenario for the use of NLG systems, it has been argued that certain knowledge structures are necessary to achieve intelligent behavior of a graphics oriented program. The major part of this chapter consists of the formal description of a collection of such knowledge structures. We will refer to the class of all such structures as "Graphical Deep Knowledge".

Before dealing with necessary representational constructs for NLG systems, this chapter will give a characterization of graphical deep knowledge as well as a precise formulation of what we consider the goal of NLG research. Terminology necessary for this endeavor is worked out on the way. Later on it is found that inheritance of attributes makes sense along part hierarchies, not only class hierarchies, and an argument against representation systems built on IS-A hierarchies will be derived from this finding. In this way the specific domain helps to improve knowledge representation theory as a whole.

The work presented here makes use of the SNePS [Sha79b] Semantic Network Processing System as a notational formalism as well as an implementation "language". Information and clarification about SNePS will be supplied whenever necessary.

5.1.1. A Goal Statement for NLG

It is the major purpose of this chapter to develop a precise goal statement for NLG. On the way some interesting terminology can be developed. For the benefit of the reader the chapter has been divided into a few subsections, but it should be seen as one long argument.

5.1.1.1. The Reductionist Character of Representations

In this section we will address three basic properties of knowledge representation systems and talk in some detail about one of them, the "reductionist character" of knowledge representation. This discussion has been inspired by work of Woods who has discussed fundamental issues of knowledge representation in [Woo87].

The first (and trivial) basic feature of a representation is its characteristic of being a function. It maps an entity or a group of entities into another entity or group of entities. For instance a map is a representation of a piece of land. The second characteristic of a representation is that it is considered to be a representation by *somebody*. As Minsky would say, "the model relation is inherently ternary" [Min68]. Every representation has it in its nature that it can be considered an object by itself, and it is an act of interpretation that makes somebody not look at a map as a piece of paper but as a representation of a piece of land.

Any object that is seen as a representation of another object can nevertheless itself be viewed as an object. The term *primary object* will be used to designate an object that is *not* regarded in its function as representing another object. So the same physical object can either be viewed as a primary object or as the representation of another object. For instance when somebody cuts off the white margin around a map, then he performs an operation on the primary object and does not refer to its meaning as a representation at all. On the other hand if one corrects the borderline of a country that was changed by war, then this is an operation that changes the representational quality of the map.

The important aspect of representations that we want to focus on is that they are usually *reductionist* in order to be of any value. That means that the representation is missing certain features of the original which makes the representation more "manageable" than the represented object. For instance a map is missing the original size and three dimensionality of a piece of land which makes it possible to fold it and store it in a drawer, while it still preserves enough of the features of the original to be an interesting representation.

There are sometimes exceptions to the reductionist character of representations. When a jet fighter of a hostile nation is captured, then one can consider this fighter as a primary object as well as a representation of how the other fighters look, and one would then have a non-reductionist representation. But even in this extreme case one could not rely on totally identical properties of represented objects (any other fighter) and representation (the one at hand). Cars of the same type and make often show very different behavior, and some turn out to be "lemons" without anybody knowing why they should be different from all the other cars that came from the belt on the same day. In conclusion one can say that most representations are reductionist by design, but even if a representation is meant to be perfect, like one car seen as the representation of another car of the same make and type, it is still never possible to accomplish a complete representation of all features.

These observations on representations apply to knowledge representation, and unless one tries to model abstract objects with very few properties, knowledge representation will be a reductionist mapping. Woods [Woo87] (p. 47) refers to three "fundamental limitations" of modeling the world by an intelligent system, namely (1) changes of the world may not be captured, (2) there is too much to be learned, and (3) the world is too rich to even conceptualize certain things. If one uses a fully *intensional* knowledge representation system like SNePS, then things become even more complicated. Because intensional knowledge representation does not try to model a world, but a cognitive agent, one has to consider *two* potential steps of reduction. The first step of reduction happens when knowledge about a world is represented in an incomplete fashion by a cognitive

agent, while the second step occurs in modeling this agent based on incomplete information about his internal structure. These facts will be summarized in what we want to call the "First Fundamental Conjecture about Knowledge Representation (First FCoKR)".

> **The First Fundamental Conjecture about Knowledge Representation:**
>
> Every knowledge representation for any non-trivial world is a *reductionist* mapping. "Reductionist" hereby means that the mapping contains less information than the mapped entity.

Besides the problem of potentially non-accessible information, there are two different limitations that contribute to the reductionist character of KR systems. First of all, the complexity of any non-trivial world will surpass the *representational capacity* of all known intelligent systems. Secondly, for any representational system there is a limitation in grain size. At some point it becomes possible to make a change to a represented domain that is so small that no corresponding change in the representation is possible due to grain size limitations. This is a limitation of the *representational adequacy* of the representation system.

5.1.1.2. The Need for an Internal Relevance

Any function that performs a reductionist mapping from a given world state into an internal representation of a cognitive agent will be called a *knowledge acquisition mapping (KAM)*. This mapping will normally be based on some perceptual mechanism. However we are not interested in details of this mechanism.

We will, for now, notate the functionality of the KAM by

$$K = KAM\ (W, M) \qquad (5.1.1.2.1)$$

where W is a perceptually accessible world, M is the internal state of the cognitive agent, and K is the resulting knowledge structure of the acquisition process.

Cognitive agents are able to influence the knowledge structure K that they are building up by *manipulating* their own acquisition mapping. In other words, people are able to let their

internal goal structure influence what type of knowledge is assembled for a given world. For example, a person might very well be a chess master who has been using his home chess set for many years, and nevertheless he may not be able to give a complete description of the form of one of the figures. This is the case because he has manipulated his acquisition mapping towards supplying relations between figures, but not towards analyzing the shapes of them in detail.

One could say that a cognitive agent is able to focus his attention on relevant information. Unfortunately, as has been pointed out in the chapter on philosophical contributions to NLG research (Section 3.1), the term "relevance" has been used in the literature with a large number of different meanings [MiR85, Sal72, Sal73, ShW76] [AnB75, Car70, Kad86, SpW86] [Gri75, Gri78] which neither agree with each other nor with the notion currently of interest. In order to discriminate the currently intended sense of relevance from previous use in the field, the term "internal relevance" will be used. The notion of internal relevance differs from Grice's relevance in that he describes a phenomenon of communication between two agents. Internal relevance on the other hand does not require a second agent as source of the communication, it is limited to a single agent.

Even so there are two different ways that internal relevance can come into play. Either the agent can select information from an environment according to his criteria of internal relevance, or he can, in the process of reasoning, generate new information and test it against his criteria of internal relevance. If the new information does not conform to his criteria he can eliminate it from further elaboration.

Definition 5.1.1: Internal Relevance Criterion:

> An internal relevance criterion is a set of goals that can be used by a cognitive agent as a filter in his knowledge acquisition mapping.

This results in adding an additional argument to the knowledge acquisition mapping, leading to a modified version of (5.1.1.2.1).

$$K = KAM\ (W, M', RC) \qquad (5.1.1.2.2)$$

where RC is a relevance criterion. The original M has been modified to M' because M comprises all of the cognitive agent's knowledge, and therefore also his relevance criteria. M' eliminates this redundancy. So the use of an internal relevance criterion helps cognitive agents to deal with the problems imposed by the First FCoKR.

5.1.1.3. Differentially Adequate Knowledge Representation

There is an obvious relation between internal relevance and the First FCoKR. If one were not forced to do a data reduction, one would not need a criterion how to do it. However, the First FCoKR also results in the generally accepted practice of everyday work in knowledge representation. Charniak & McDermott [ChM85] describe the design of a knowledge based system as a repeated attempt to stretch a given representational mechanism to the breaking point. If the breaking point is reached, then amendments to the representational system become obvious and will have to be made, constituting the effective research progress.

This practice is frustrating, because knowledge representation is by no means done for its own purpose. Most knowledge representation systems are part of a larger unit. A parser might supply input by translating natural language into knowledge structures, or a natural language generator might be driven by stored knowledge. The representation itself can be considered an intermediate medium that is used to manage the complexity of the natural language parsing and generation problem. Even in the theoretical setting of an AI research lab one will have to try for a parser. After all, a successful parser / representation / generator system is a proof of the effectiveness of the representation.

To really appreciate the difficulty of this task, it pays to take a comparative look at one of the simplest parsing problems in computer science, namely that of parsing an integer number into a word of memory. A number of different binary codes has been designed (BCD, Huffman, Hamming, Grey,...), and in order to write an appropriate parser, the correct representation has to be known. However, let us assume that it is not known how many bits the device has, nor is there a

complete list of all possible input numbers, nor a list of what binary patterns correspond to what numbers for the majority of the input numbers. Nobody would seriously attempt to write such an ill defined procedure.

A look at natural language parsing unfortunately shows that one is in exactly this unpleasant situation. There are no generally accepted constraints on the representation except maybe finiteness and discreteness. The existence, number and quality of system primitives, comparable to the number of bits, is usually a research issue, not a known fact. Given the limitations of the First FCoKR one cannot even hope to achieve a complete correct description of a given world. The representations for most possible natural language utterances are not trivially known. There is not even a complete list of all possible natural language utterances that one would like to parse, because there are infinitely many possible utterances. But even a weeker descriptive mechanism like an acceptor or a grammar that exist for many formal infinite languages does not exist for natural language.

Some people might seriously think that the task could not be done at all, were we not confronted with living proof all the time.

So the KR researcher takes single natural language utterances, or small classes of them and creates corresponding knowledge items using what he thinks is an approximation of his own mental algorithm. His main goal is to make sure that two utterances which are obviously different to the human observer are never forced by limitations of the representational system to be mapped into identical representations.

We would like to capture this desired property of knowledge representation systems by a definition:

Definition 5.1.2: Differentially Adequate KR Systems:

> A knowledge representation system is differentially adequate if any two worlds with an interesting difference are representable by different representations.

The key problem in the definition above is the word "interesting", which we could as well have replaced by "relevant". It turns out that researchers sometimes do not agree on what would constitute an interesting difference. In other words, the KR researcher is using his own private internal relevance criterion all along.

There is nothing inherently wrong with the use of one person's relevance criterion, if a representational system is accepted by more than one researcher. In this sense knowledge representations become a social phenomenon, as much as a proof in mathematics is only a proof if a number of educated people agree on it. Unfortunately, only very little effort has been expended towards making the implicit internal relevance criterion that went into some representational system explicit. So we conclude that the best we can achieve in Knowledge Representation is to find differentially adequate representations, parameterized by the shared (often implicit) internal relevance criterion of a researcher community. This limits the goals that we can set for NLG research.

5.1.1.4. A Second Fundamental Conjecture about Knowledge Representation

In this section we will discuss a second conjecture about knowledge representation which is not directly related to the rest of this chapter but builds on the ideas discussed earlier.

It has been argued in the cognitive psychology literature [BMR82] that vision processes require the use of top down semantic information. Biederman writes that "Access to the semantic relations among the entities in a scene is not deferred until the completion of spatial and depth processing and object identification. Instead, an object's semantic relations are accessed simultaneously with its physical relations as well as with its own identification" (p. 144).

Kosslyn[1] [KoS77] was already previously quoted saying that "conceptual information can be used in image generation..." (p. 271). Kosslyn's paper describes a program that creates "mental images" from a "propositional representation" and uses them for a number of operations. The

[1] We continue to ignore the imagery debate. We simply take the standpoint of the imagists.

claim of Kosslyn and fellow researchers is that the mental images that people report are not an epiphenomenon of a propositional knowledge representation, but an experimentally demonstrable functional representation. A large amount of supporting experimental evidence is given in [Kos80]. The "propositional representation" used contains among other things *coordinate values*, and the internal images created by Kosslyn's simulation program can be *displayed*.

A system that creates pictures from coordinate values is usually referred to as a "computer graphics program", and the obvious conclusion is that anybody who subscribes to Kosslyn's theory of mental images is claiming that there must be some sort of computer graphics system in the human brain which we want to refer to as a *mental graphics system*. (We hurry to stress that we do not claim the existence of real pictures in the brain, we are only drawing a simple conclusion from Kosslyn's implementation which does not disagree with the notion of a functional representation.)

We argue that the propositional knowledge for "vision" and "mental graphics" consist of shared parts as well as parts that are dedicated to each one of these activities. Research in computer vision and computer graphics would be considerably more unified, if all of the knowledge for vision and graphics were shared.[2] On the other hand, both tasks deal with a number of common high level concepts like shapes, relative arrangements, and colors, and we subscribe to localized and unique representations of concepts. Therefore both tasks will have to access the same shared semantic representations concerning shapes, etc. We will say that conceptual knowledge that can be used for vision has the attribute of "receptive adequacy". Semantic knowledge used for (mental) graphics is "projectively adequate".

Semantic knowledge of the described kinds has to be acquired, and here the First FCoKR comes into play. An internal relevance criterion is necessary for both types of knowledge. Nevertheless, all the acquired knowledge will be resident in one semantic memory which is highly

---

[2] Of course we cannot exclude that whatever is happening in the brain is unified and completely different from our current scientific approaches.

interconnected. This might pose difficulties at access time. Minsky [Min75] has argued that one of the most important parts of intelligence is the retrieval of the right piece of information at the right time. This attitude disavows the separation of intelligence into knowledge store and processor. Recent developments in the philosophy of psychology also criticize such a separation.[3]

One possible way to retrieve the "right knowledge at the right time" is to organize it according to the expected task. However, we would not want to reorganize knowledge after it had been acquired. It would be much more efficient to classify knowledge according to its use from the very beginning. This is the content of our Second Fundamental Conjecture on KR (Second FCoKR).

The Second Fundamental Conjecture about Knowledge Representation:

A task oriented knowledge representation system should make use of the same internal relevance criteria for knowledge acquisition and for task related knowledge access.

In other words, it is more parsimonious to organize knowledge at acquisition time according to the expected use, than to first acquire it and restructure it later on for use. Even more importantly, when goals change, and therefore internal relevance criteria are adapted, it is more parsimonious to adapt only one mechanism, as opposed to adapting the knowledge acquisition and the knowledge retrieval part of the system. (Prior knowledge will need some restructuring anyway).

5.1.1.5. Definition and Goal of Graphical Deep Knowledge Research

In the previous section we have mentioned knowledge bases that can be used for generating diagrams and have called them "projectively adequate". In addition we want to do propositional reasoning about knowledge bases describing the physical structure of diagrams, and a knowledge base that permits to do this will be said to exhibit "deductive graphical adequacy". While the terms "visual knowledge" and "graphical knowledge" are standard AI terminology, the literature reports no good name for the type of knowledge we are referring to. This is why we have intro-

---

[3] John Haugeland, talk given on 4/23/87 at SUNY at Buffalo

duced the term "Graphical Deep Knowledge" [GeS87].

Definition 5.1.3: Graphical Deep Knowledge:

> A knowledge base is said to contain graphical deep knowledge if at least part of its knowledge exhibits deductive graphical adequacy, and part of its knowledge exhibits projective adequacy.

While the utility of designing a differentially adequate representation system as a first step in creating the whole complex of representation / parser / generator / reasoner has been well established in natural language oriented knowledge representation, especially in the analysis of belief systems [ShM82, WiR86, WiB83], we think that graphical deep knowledge has not been analyzed in the same way. Therefore a research program that focuses on a task domain analysis of graphical deep knowledge in a comparable way to the analysis of belief systems is a worthwhile endeavor.

Creating a differentially adequate representation system is however not good enough. A knowledge representation by itself is meaningless, unless the semantics of the representation is given. Descriptive semantics (i. e. a good natural language explanation) or procedural semantics are two well known possibilities. We will give syntax and descriptive semantics for each introduced structure of graphical deep knowledge. Whenever useful, we will informally describe how the given structure influences the behavior of an abstract graphics machine that processes display requests.

5.1.1.6. Semantic Primitives and Differential Adequacy

Existing knowledge representation systems differ with respect to their use of semantic primitives. Some systems supply the user with a fixed set of primitives, while other systems supply him with tools for creating semantic primitives. Neither AI nor philosophy have so far succeeded in defining a reasonable set of primitives that are sufficient in all situations. Researchers whose goal it is to identify such a set are therefore often forced to use ad hoc extensions of their theory if unexpected difficulties arise in an application.

Two examples of theories that have been created with a set of primitives in mind, which later on had to be extended are Schank's primitive actions and Fillmore's deep cases [Fil68,ScA77]. Schank's work was later extended to include additional primitive (social) actions. Concerning Fillmore's deep cases, no compilation of a generally accepted set of cases exists to date [BrM87]. Problems with finding the ultimate set of semantic primitives have therefore led researchers to design systems that permit the user to specify his own set of primitives. Such systems can be seen as tool kits to design and test knowledge representation systems, in effect making them knowledge representation design systems (KReDeS). This approach is used in the SNePS (Semantic Network Processing System) [Sha79b] system which we will introduce at a later time in more detail. However, it is necessary to anticipate some of the SNePS ideas in order to analyze the relation between differential adequacy and semantic primitives.

SNePS is a semantic network system with arcs and nodes where the user is required to define the arc labels that he wants to use. (There is a small set of predefined arc labels which are used by the reasoning system, however, this is not meant to be in any way exhaustive). Our work with SNePS has advanced under the "assumption of asymptotic domain coverage". According to this assumption it is possible to define a small domain of world knowledge, and by analyzing it thoroughly to converge towards a set of semantic primitives (arc labels) which are sufficient to completely describe this domain.

At no point in time can one be sure that a complete set of primitives has been found, but the number of new labels necessary asymptotically approaches zero. Attempts to reach asymptotic domain coverage have been made by other researchers using SNePS in the area of belief sentences [WiR86] natural language syntax and semantics [Nea85] and temporal expressions in narratives [Alm87].

This permits us to define the goals of graphical deep knowledge research in a more precise way. We are aspiring to reach asymptotic domain coverage for the domain of graphical deep knowledge. An understanding of the overall goal of the NLG research program requires more

analysis of the natural language side and will be given at the end of the next section.

5.1.1.7. The Linearity Principle of Knowledge Representation

Differential adequacy is an important goal in designing a knowledge based system, however it is by itself not sufficient. If a knowledge based system achieves asymptotic domain coverage, then we are only guaranteed that every interesting difference in the world can be represented, but it is not clear how "simple" this representation will be.

It seems unsatisfactory if a cognitively simple concept can only be expressed by a large number of primitives. This does not say that it is per se bad if a short sentence is expressed with many nodes and arcs, because one might have a very rich representation system that attaches inferred knowledge to the actual representation of the sentence. But if this really happens, then we would like to have it happen consistently. In other words, we do not want to represent most five word sentences with three or four semantic network nodes, but have one five word sentence of this language that can only be represented with 25 nodes.

It is difficult to define complexity measures for natural language and for knowledge structures, but one would like to have a more complex language utterance expressed by a more complex knowledge structure, and vice versa. In other words, one wants a linear relation between the complexity of an utterance and the complexity of its image in the knowledge base. This idea has been guiding our work on the language side of NLG systems. We will refer to it as the "linearity principle".

The Linearity Principle (comparative form):

> Of two knowledge representation systems of comparable and satisfactory differential adequacy the better system is characterized by better approximating a constant with its quotient of knowledge structure complexity and complexity of the corresponding sentence for a wide range of sentences.

We need to talk about two systems of satisfactory differential adequacy, because we do not want to invoke the linearity principle for systems that are not differentially adequate. Otherwise one might use the "empty mapping" which assigns the empty knowledge structure to every sentence.

This will result in perfect linearity with a quotient of 0. However this system would not be satisfactory in its differential adequacy.

The linearity principle (LP) can be seen as a generalization of Shapiro's work on non-standard connectives [Sha79a]. Shapiro has argued that the connectives of standard first order predicate logic are not very pleasant for certain applications. For instance if one would want to represent the sentence "Two of chip-1, chip-2, chip-3, and chip-4 are multipliers", the resultant structure might look like this:

```
is-a( multiplier, chip-1) & is-a( multiplier, chip-2) OR                (5.1.1.7.1)
is-a( multiplier, chip-1) & is-a( multiplier, chip-3) OR
is-a( multiplier, chip-1) & is-a( multiplier, chip-4) OR
is-a( multiplier, chip-2) & is-a( multiplier, chip-3) OR
is-a( multiplier, chip-2) & is-a( multiplier, chip-4) OR
is-a( multiplier, chip-3) & is-a( multiplier, chip-4)
```

Had we chosen an example with four multipliers in a set of ten chips it would be unthinkable to assume that such a structure is built by a person upon hearing the corresponding sentence. Therefore Shapiro has introduced connectives that directly can represent sentences like "more than i but less than j of the following n assertions ... hold true".

Before we present an example application of the linearity principle it is necessary to remind the reader that the arc labels in SNePS networks (Fig. 5.1.1.7.1) are seen as system primitives. The number of different arc labels is not fixed and can be extended by the user [ShG86a].

If people can describe a simple arrangement of objects by a short sentence then it should be possible to describe it with a reasonably simple SNePS structure. If this is not the case then the number of user defined primitives has to be extended to accommodate the sentence. (Of course new primitives will also have to be used if the sentence is not representable at all).

For instance if two people are sitting in front of a graphics terminal displaying the Adder-Multiplier (which has been used in maintenance research, Fig. 5.1.1.7.2), and one of them asks:

"Tell me the names of all multipliers."

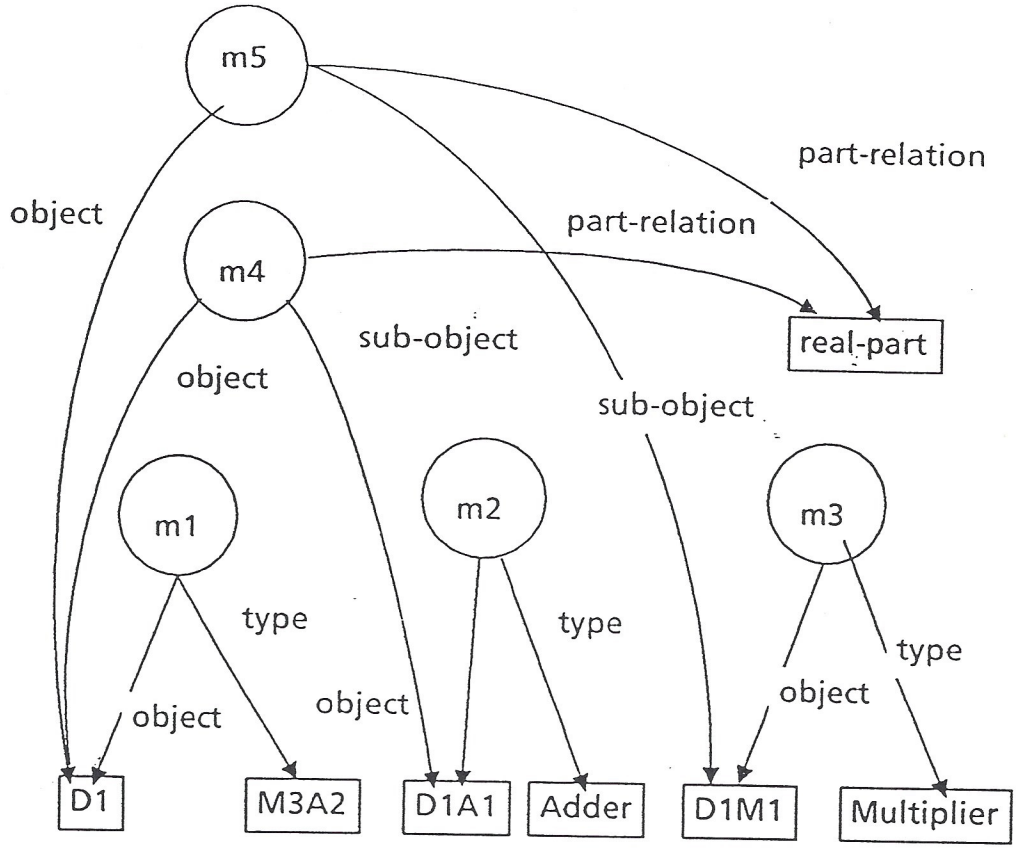


Fig. 5.1.1.7.1: A SNePS network.

then the other person will presumably be able to do that. Therefore one would want an NLG system to be able to do the same thing. Also the knowledge base should contain information on all multipliers in a format approximately linear in size with respect to the answer given by a person. This leads directly to an old idea, the implementation of a class hierarchy. (Less obvious examples will offer themselves naturally at later occasions).

Armed with the linearity principle a comprehensive description of the goal of NLG systems based on graphical deep knowledge can finally be given.

THE GOAL OF NLG RESEARCH:

Given the language L of all utterances that create, describe, modify, or query a world of symbolic graphical representations. NLG as we want to understand it has the following goals:

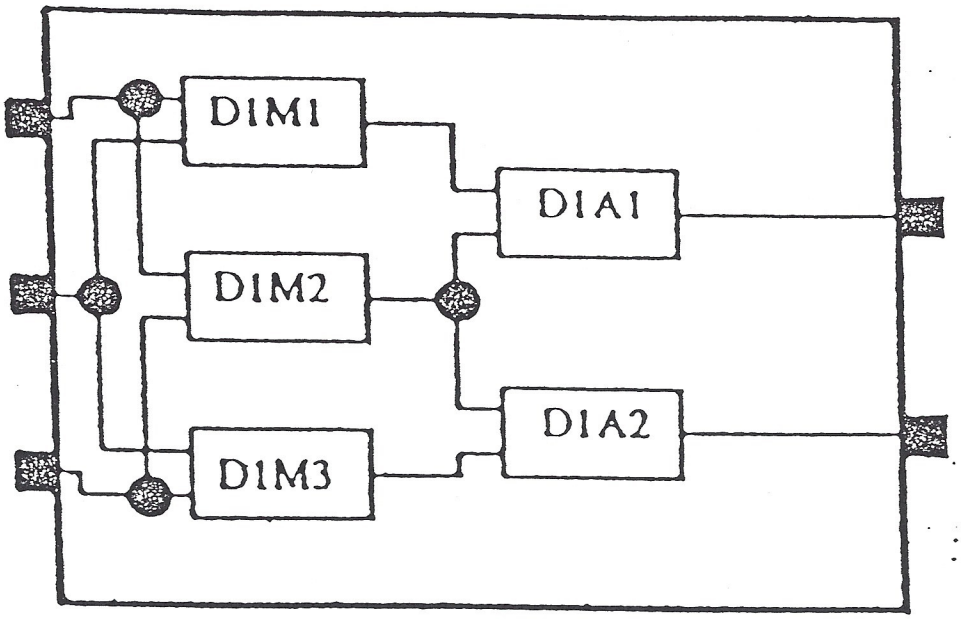


Fig. 5.1.1.7.2: The adder-multiplier

(1) To achieve asymptotic domain coverage for Graphical Deep Knowledge, such that members of the set of created knowledge structures, called S, relate to sentences of the language L by virtue of the linearity principle.

(2) To design a generator that maps structures of S into pictorial representations.

(3) To design a parser that maps sentences of L into structures of S.

(4) To design a function that generates sentences of L from structures of S.

The parts of this goal statement are given in order of importance, so our focus is on the description of correct knowledge structures for graphical purposes. The use of NLG that has been called before *criterial* concentrates on the parsing and language generation algorithms.

5.1.2. The SNePS Knowledge Representation System

5.1.2.1. Introduction

The SNePS semantic network processing system has been aptly described in a number of places [Sha79b,ShS83,ShR86]. SNePS networks are labeled directed graphs consisting of nodes and arcs, but not all such graphs are legal SNePS networks. Nodes represent concepts, i. e. entities that the system can reason about. Arcs represent non-conceptual binary relations between nodes. The major building blocks of SNePS networks are *propositions*. This makes SNePS a

propositional network as opposed to an inheritance network like KL-ONE, NIKL, KL-TWO, LOOM, KRYPTON, or other members of the KL-ONE family [BFL83,BFL85,BrS85,MaB87].

The normal way to represent a conceptual relation is to create one node for each argument and one node for the relation itself. The SNePS system automatically generates an additional node that represents the proposition itself, i. e. a reification of the relation. This node is named by SNePS with a label of the form mXX, such that XX is a unique id number. Once such a proposition node is built, the user is not permitted to add any nodes under it. (The specific meaning of "under" will be explained shortly).

If the user decides that the relation itself is of no interest to him, i. e. he does not expect his system to reason about the relation, then he may omit the node corresponding to it. So the semantics of a SNePS structure is not dependent on the relation node, but on the combination of arcs under the proposition node. In other words one can view combinations of arcs emanating from a common proposition node as a case frame and assign this frame a meaning. Individual arc labels do not have a semantics. The interpretation of case frames as the meaning carrying units in SNePS is based on the ideas of Fillmore's [Fil68] case grammar. This ancestorship shows in the treatment of missing information by SNePS. If the filler for one or more cases (slots) is not available it is permissible to omit the corresponding slot. A structure that has a set of arc labels that are a subset of the labels of another structure is not a completely different case frame, but a reduced version adapted to the available information. This assumes that the arcs in both structures are at the same level, i. e. they are emanating from corresponding nodes.

In graphical network representations of SNePS networks it is customary to draw proposition nodes above their corresponding arguments, and this is expressed by saying that they *dominate* the argument nodes. (See Fig. 5.1.1.7.1). However there is no absolute necessity to maintain the domination relation in the picture. The directionality of the arcs is sufficient to determine the proposition node. Nodes that do not dominate any other nodes are called base nodes and are used to express elementary (often individual) concepts. A node that is not a base node is called a molecu-

lar node (therefore the labels of the form mXX).

The arcs displayed in graphical representations are called "descending arcs". They point from proposition nodes to the arguments they dominate, or possibly from a proposition A to another proposition B if A expresses something about B. Arguments of propositions may themselves be molecular nodes that are not propositions. Such nodes are called *structured individuals.*

The interpretation of a node as proposition or structured individual is the responsibility of the user (or the program interpreting the knowledge structures). The following rule of thumb helps: top level structures and structures representing SNePS rules are always propositions. Dominated molecular nodes are propositions if an intensional action ("believe") is expressed about them, otherwise they are likely to be structured individuals. The differentiation between propositions and structured individuals corresponds to the differentiation between sentences ("The house is red") and attributive phrases ("The red house..."). Therefore, when in doubt, one can interpret everything as a proposition, potentially loosing readability.

For every descending arc there is an ascending arc that connects the same two nodes but points in the opposite direction. Ascending arcs are labeled by appending a minus sign to the label of the corresponding descending arc. They are not shown in network pictures, but they are necessary to make connected pairs of nodes mutually accessible. Because of the universal use of antiparallel ascending and descending arcs, any concept in a net can be reached from any other concept, a property that we consider desirable.

SNePS is a "neat" knowledge representation system that has the whole power of predicate logic, and in addition features a number of sophisticated non-standard connectives, and it is possible to build rules that correspond closely to natural language terms expressing rules. This is, for instance, not the case for first order predicate calculus and is an implicit application of the linearity principle. SNePS rules can be used in forward, backward, and bidirectional reasoning modes. This power is avoided by other representational systems because of its penalties in run time efficiency [ChM85].

SNePS features and the right way of reading SNePS networks will be introduced as necessitated by the ideas that will be developed. However, it is important to counter some methodological criticism that might be leveled against this modus operandi.

The principal ideas of the SNePS system have been stable since 1971 [Sha71]. Changes to implementation, new host machines, new LISP dialects, and sometimes new interpretations of subtle details have occurred, but the basic ideas of the system are unchanged. So the fact that different features of the system are introduced as necessary does not reflect the ad hoc creation of these features in order to deal with a specific problem, but it reflects a pedagogic attitude towards introducing ideas where they are needed.

5.1.2.2. Notational Conventions for SNePS Networks

Fig. 5.1.1.7.1 shows an example of a typical SNePS network. The nodes m1, m2, m3, m4, m5 represent propositions. m1 expresses the fact that the object D1 is of type M3A2. m2 expresses the fact that the object D1A1 is of type Adder. m3 expresses the fact that the object D1M1 is of type Multiplier. m4 expresses the fact that the real-part relation holds between D1 and D1A1. m5 expresses the fact that the real-part relation holds between D1 and D1M1. An equivalent first order predicate calculus representation for Fig. 5.1.1 would be the following one:

type(m1, M3A2) & object(m1, D1) (5.1.2.2.1)
type(m2, Adder) & object(m2, D1A1)
type(m3, Multiplier) & object(m3, D1M1)
sub-object(m4, D1A1) & object(m4, D1) & part-relation(m4, real-part)
sub-object(m5, D1M1) & object(m5, D1) & part-relation(m5, real-part)

This representation is unnecessarily redundant, and we will introduce a pseudo-predicate notation according to the following formal scheme. Given a conjunction of a number of binary predicates with identical first arguments, transform the first argument into a pseudo predicate. Transform all binary predicates into arguments at odd numbered positions, and insert all second arguments at even numbered positions. In symbols:

$$\mathop{\&}_{i=1}^{n} p_i(a_0, a_i) \rightarrow a_0(p_1\ a_1\ p_2\ a_2\ \ldots) \qquad (5.1.2.2.2)$$

This transformation is syntactic sugar and has no influence on the meaning of the representation which depends on the combination of system primitives (arcs). As an example we will show the translation of (5.1.2.2.1).

```
m1( type            M3A2                                   (5.1.2.2.3)
    object          D1)

m2( type            Adder
    object          D1A1)

m3( type            Multiplier
    object          D1M1)

m4( sub-object      D1A1
    object          D1
    part-relation   real-part)

m5( sub-object      D1M1
    object          D1
    part-relation   real-part)
```

Whenever we want to abstractly describe a class of structures we will use a case frame notation instead of a linearized network representation. In this case dominated nodes will be replaced by bracketed terms describing classes, and dominating nodes will be eliminated. If it becomes necessary to refer to a whole structure, for instance because it is used as a sub-structure at some other place, then we will name the structure with a bracketed term followed by a colon. If a structure may occur in two alternative formats we will separate them by an exclamation mark. We will now show the previous example in case frame notation. The case frame for the fifth structure is a duplication of the case frame for the fourth structure and may therefore be omitted. Similarly, the case frames for the first, second, and third structures are identical, therefore only one of them needs to be shown.

```
type            <device-type>                              (5.1.2.2.4)
object          <object-1>

sub-object      <object-2>
object          <object-3>
part-relation   real-part
```

Introducing a name for the first structure we get:

<membership> : (5.1.2.2.5)

type	<device-type>
object	<object-1>

This representation might look like BNF notation, but it is different from it, because the pairs of slots and fillers may occur in any order and are not constrained by the order in which they are shown here. However, for each individual pair, slot and filler may not be exchanged. In other words (5.1.2.2.5) is identical to (5.1.2.2.6), but not to (5.1.2.2.7).

<membership> : (5.1.2.2.6)

object	<object-1>
type	<device-type>

<membership> : (5.1.2.2.7)

type	<device-type>
<object-1>	object

In continuous text case frames are sometimes simply shown as lists of arc labels. In this notation (5.1.2.2.6) would be represented as (object type) or (type object).

Pictures have been looked at with suspicion in parts of the scientific community. For instance in AI Hayes [Hay77] states that "If someone argues for the superiority of semantic networks over logic, he must be referring to some other property of the former than their meaning (for example ... their attractive appearance on a printed page)" (p. 561). Although we have seen some reversal of this attitude lately [Riv87], we have introduced above representational conventions to give a precise linear representation of the syntax of knowledge structures.

5.1.3. Representational Constructs of Graphical Deep Knowledge

5.1.3.1. Form Knowledge

A number of different scientific subfields and fields have been interested in the representation of forms. Among these are the already mentioned computer vision, computer graphics, and imagery, but also solid modeling [Req80], computer aided design (CAD), and character recognition. It turns out that no representation in any of these fields satisfies the requirements for

graphical deep knowledge.

Computer vision has developed the largest number of different form representations [BaB82]. All representations are however insufficient because they are not projectively adequate. At this point it is useful to divide the notion of projective adequacy into a narrow sense and a wide sense. A form representation is *projectively adequate in the narrow sense* if it can be fed "immediately" into a graphics processor which will then create a picture. A representation is *projectively adequate in the wide sense*, if it contains all the information necessary to create a display, but an intermediate step of assembly is necessary to transform it into graphics code. "Assembly" hereby refers to any process requiring arithmetic or rearrangement, but not to a translation that can be handled by one step table lookup.

No computer vision representation is projectively adequate in the narrow sense for a vector graphics system, although polyline representations [BaB82] come near to this requirement. However, a number of representations used in computer vision are not even projectively adequate in the wide sense. A typical example would be Fourier descriptors [BaB82]. This representation is based on the approximation of a curve by an infinite convergent series of trigonometric functions. The perfect reconstruction of the given form would require to compute this infinite sum which is normally impossible. Therefore Fourier descriptors are an example of a representation used in computer vision which does not preserve complete form information. (Fourier descriptors have a few nice properties though, for instance they can be formulated to be invariant to some geometric transformations). Computer vision representations might not be projectively adequate, because to be projectively adequate would inhibit their ability to solve the computer vision problem.

Computer graphics representations are by definition projectively adequate in the narrow sense and are organized to permit efficient projection. However they usually fail at the other requirement for graphical deep knowledge in that they don't show deductive graphical adequacy, at least not in human terms. This requires further clarification.

Most graphics systems operate on primitives at the level of points, lines, arcs, and polygons. They do not operate on conceptual objects. Given a picture of a chair the system might know things about the lines that make up the chair, but not about the chair as such. Only model based computer graphics systems [FoD83] store interesting information about the world. (We already pointed this distinction out). However, we do not want to refer to a world model as a knowledge base and to operations on such a model as propositional reasoning.

Our conception of knowledge, namely declarative propositional knowledge, is certainly different from the data structures used in model based graphics. We are hereby in line with Smith's knowledge representation hypothesis [Smi85]. This widely quoted (e. g. [LeB85]) definition says that

> Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behavior that manifests that knowledge.

Robins [Rob86] expresses the differences between knowledge and data in the following way. He says that "... knowledge is not the same as data. Knowledge has a certain depth to it, while data is rather linear in nature; knowledge tends to be abstract and symbolic, and may contain rules and other information from which inferences may be made, while data tends to be composed of simple tables of attribute/value lists...." (p. 6).

Frijda [Fri72] characterizes human memory by four properties which Giustini [GLM78] has taken to mean that these properties should be exhibited by an artificial intelligence system. These properties are that memory has to be associative, teachable, inferential, and able to retrieve information given input which differs from the the structure used to learn it.

In summary, we think that knowledge is characterized by the following attributes. (1) Knowledge is symbolic and propositional. (2) Concepts in a knowledge base are highly interconnected. (3) Any two concepts in a knowledge base are mutually accessible. (4) Knowledge is in an appropriate format for a reasoning engine that has access to it. ("Knowledge in a book locked

into a drawer is not knowledge.") (5) Knowledge is relational, with relations defined over domains of small dimensions. (6) Knowledge is uniform, in the sense that meta-knowledge is represented with the same formalism and in the same space as knowledge.

While every form of mechanical knowledge representation is ultimately implemented as a data structure, the opposite does not hold, and a data structure that does not conform to (1) - (6) above cannot be referred to as knowledge. In this sense computer graphics systems do not exhibit deductive graphical adequacy, because they are not based on a knowledge representation, not even if they have a world model.

Character recognition [Har72] is probably the field that most completely ignores projective adequacy. Many character recognition systems [KeB81] as well as word recognition systems [Hul85] rely on systems of features. The totally enclosed space in an "o" would be a possible feature to distinguish it from the open "c". Feature models are obviously not formulated for projective adequacy and fare poorly in that respect. The other major model of character recognition is template matching. Template based character recognition appears to be better for projective purposes, but here there is no information about parts of characters stored. Parts are a conceptual structuring tool that is used in human reasoning and should be contained in a graphical deep knowledge based representation. So templates are better in their projective adequacy but worse in their deductive adequacy.

CAD systems [Ger85] and solid modeling as an integral part of it [Req80] come closest to the requirements of graphical deep knowledge. A solid modeler is always connected to a graphics interface. Therefore it is able to draw pictures which therefore proves that it is projectively adequate. Some steps have been made towards deductive graphical adequacy. For instance so called constructive solid geometry models (CSG) of solids are based on the combination of three-dimensional primitives with boolean operators. The combination of two primitives with "OR" and "AND" operations creates a representation that contains important hierarchical information, structured in a way close to the human reasoning process [Req80].

Requicha [Req80] has pointed out that solid modeling systems profit from redundant representations. This observation has to be carried over to the realm of graphical deep knowledge. He notes that CSG representations support the use of a number of important algorithms, while surface representations are preferable for actual display creation. Therefore hybrid models are redundant but more powerful than simple CSG representations.

Before presenting the approach to form representation advocated here, a short repetition of the requirements is helpful:

- The representation should be projectively adequate in the narrow sense.
- The representation should be deductively adequate.
- The representation should be based on conceptual primitives which seem natural to the human observer.
- The representation should support relations between primitives which are natural to humans.
- The representation may contain redundant information.

To fulfill these requirements a representation with the following properties is used. The representation consists of basic forms (icons) and asserted relations. The basic forms are (supposed to be) meaningful to human observers. Every basic form is represented as a procedure that has three properties. (1) The procedure consists of calls to graphics primitives. (2) Executing a procedure of the name <name> results in the drawing of an object that is described by <name>. (3) The procedure name is accessible as a concept in the knowledge representation system, i.e. it functions simultaneously as a node in a semantic network. The representation of a basic form is therefore projectively adequate and also a conceptual unit.

Relations between icons are represented propositionally. A large number of different proposition types is permissible, and every type will be dealt with in its own chapter.

The SNePS system is used in the following way to accommodate the described form representation. The name of every basic form in the system is a base node in the SNePS semantic network. The SNePS inference machine treats it as a conceptual unit and permits reasoning about

it. At the same time every SNePS node is also an uninterned LISP atom[4]. This uninterned atom is accessible through a so called node-access which is an interned node of the same print-name. As is well known, LISP assigns a number of different cells to every atom. Franz LISP in particular, which has been one of the SNePS implementation languages, supplies a function cell, therefore it is possible to store a LISP function in the function cell of the interned node-access. This function is made up of calls to graphics primitives from a LISP graphics package and, if called, will create the picture described by the corresponding concept.

So far forms have been explained, but nothing has been said about objects. Objects and forms are separate concepts, linked by an asserted proposition. This conceptual separation of forms and objects opens the doors to a number of knowledge representation techniques which have been used widely. For instance it becomes possible to associate a form with a class of objects, instead of a single object. Many objects can then be made members of this class.

Before presenting the actual knowledge structures for form assertion, an additional argument for redundant representations will be derived from a concrete example. Objects have a number of features that people seem to realize immediately when looking at them. An example of such a feature is symmetry [EaL86]. The knowledge that an object is symmetrical is obviously not necessary in order to draw it. A representation that is projectively adequate but has no redundancy would therefore lack this information.

An NLG system asked whether a certain object is symmetrical could run a symmetry detection procedure on its form. However, this does not seem to be the case for humans, because symmetry is realized at a very early stage of the recognition process, indicating that symmetry probably is itself helpful in recognizing the object [Bie87]. Therefore it is justified to explicitly represent information about symmetry, even if this information is redundant given that the whole form is stored. The actual representation for symmetry will be given much later, near the end of this chapter.

---

[4]Recently the internal representation of nodes has been changed to structures. This has no influence on the user level described here.

One should remember at this point that we are not concerned with solving the image understanding or the knowledge acquisition problem, therefore the question is not raised how symmetry information was extracted in the first place, or when redundant spatial information is created. All we are interested in is how to represent the information in a consistent and deductively adequate way once it is already given.

We are finally ready to show the knowledge structure that links the concept of an object to the concept of its form.

5.1.3.1.1. Individual Form

```
(define  form                                          (5.1.3.1.1)
         modality
         object)

(build   form      xand                                (5.1.3.1.2)
         modality  logical
         object    chip1)
```

The above two list structures are commands of the SNePS User Language (SNePSUL). The "define" command defines the arc labels "form", "modality", and "object". This command also automatically defines three more arcs, namely form-, modality-, and object- which are inverse arcs to form, modality, and object. As noted before, these inverse arcs which are required by the SNePS system guarantee universal accessibility of every node from every other node in the network and are automatically maintained without user request. By specifying the three arc labels form, modality, and object they become *primitives* of a user defined network.

The build command creates a network structure as in Fig. 5.1.3.1.1. It links an object concept "chip1" to a form concept "xand" and a modality concept "logical". Together these three arcs could be read as: "the object chip1 has the form xand under the modality logical". In other words, the meaning of the structure results from the combination of the form, modality and object arcs. The impact of the modality arc will be explained in a separate subsection. According to the notational conventions established earlier, the two commands given will result in the creation of

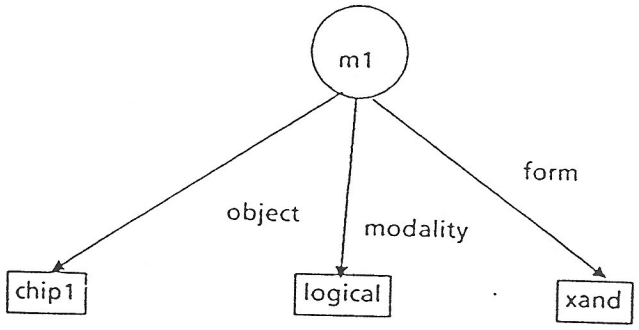


Fig. 5.1.3.1.1: Object chip1 has the form xand.

the following network:

```
m1( form        xand                                        (5.1.3.1.3)
    modality    logical
    object      chip1)
```

In the future the SNePSUL commands will be omitted and only the actual network structures shown. The form function "xand" that draws the icon displaying an and-gate is coded as follows:

```
(def xand                                                   (5.1.3.1.4)
    (lambda (x y)
      (setq CENTER (list x y))
      (mapcar (function draw-wse)
        '((xplylnrel-wse 0 0 20 0 *b)
          (xarcrel 20 -20 0 20 -180 *b)
          (xplylnrel-wse 20 -40 -20 0 *b)
          (xplylnrel-wse 0 -40 0 40 *b)
          (xplylnrel-wse 0 -10 -30 0 *b)
          (xplylnrel-wse -30 -30 30 0 *b)
          (xplylnrel-wse 40 -20 30 0 *b)))))
```

The def-statement is the Franz LISP method to define a function. The function defined is named "xand" and consists mostly of calls to the two graphics primitives xarcrel and xplylnrel-wse. xarcrel draws an arc and xplylnrel-wse a polyline (a train of line segments) respectively. The picture created is shown in Fig. 5.1.3.1.2.

Abstracting from the given example a syntactic description of an individual form would consist of the following case frame:

```
form        <form>                                          (5.1.3.1.5)
modality    <modality>
```

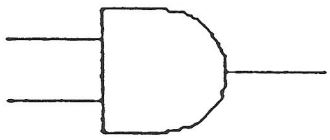


Fig. 5.1.3.1.2: An AND gate.

object <object>

We will supply a descriptive semantics for every given case frame. (5.1.3.1.5) represents the proposition that the object <object> has the form <form> under the modality <modality>.

This structure can be used to answer questions like "What is the form of <object>?" or for identifying all objects that have the form <form>, or for asserting that some agent believes in the fact that <object> has the form <form>. Assuming our model of an abstract graphics machine receives the request to draw the object <object> under the modality <modality> and at a location (x, y), then the function denoted by <form> is applied to the arguments x and y. The location (x, y) will be the location of a privileged point of the object <object> called the reference point. How x and y can be determined, their meaning relative to the screen and the nature of the reference point will be explained below.

Two notes about the epistemic status of the syntactic variables <object>, <form>, and <modality> have to be made. (1) If one of these variables is used again later on in the context of this dissertation, even if it is modified by an integer number (e. g. <object1>, <form-1>), it refers to the same syntactic variable as defined here. (2) It has been pointed out repeatedly in the literature (e. g. [McD81]) that one cannot rely on a label to express a meaning. This was said in the context of a computer processing these labels, but it applies as well to semantics specifications as given here. Therefore we will not hesitate to give a semantics specification like "<modality> represents a modality". After all, we could have used a term like <g0001> as a syntactic vari-

able. Nevertheless we will strive to use self documenting names for syntactic variables, and to supply additional information about them, wherever possible.

<form> stands for the concept of a form, i. e. of an entity that can be visualized by a person, and that can be projected by the abstract graphics machine. <object> denotes the concept of an individual object. <modality> stands for the concept of a modality, whereby a modality can best be understood as one of several possible "views" of an object. We will devote the whole next section to modalities.

5.1.3.1.2. The Problem of Display Modalities

The association of modality information with other representational structures has the following significance. Objects like gates of electronic circuit boards usually show up in different types of technical drawings. Wire plans that contain logical structures of circuits are one common form of representation. Another common form are physical representations. In general the form of an object in its logical representation is different from the form in its physical representation. Fig. 5.1.3.1.2 shows the representation of an AND gate in a wire plan. The three short line segments signify two input wires and one output wire. In reality an AND gate would have at least two more wires which are necessary for power supply, and in order to achieve an even number of *legs* a sixth unused pin. So a physical structure would look like Fig. 5.1.3.1.3.

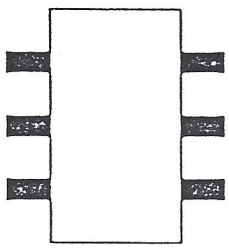


Fig. 5.1.3.1.3: The physical structure of an AND gate.

Commercially available AND gates usually come in dual-in-line packages, with four gates per package, so Fig. 5.1.3.1.3 is still a simplification of a realistic structure. This introduces the interesting problem of logical parts that do not correspond one to one to physical parts.

Even ignoring these additional problems it is clear that a single object will have different forms and different positions if displayed logically or physically. We refer to these two different ways of display as the *logical (= functional) display modality* and the *physical (= structural) display modality*. Any assertion in the network therefore has to be qualified by the modality for which it is valid.

Example:

```
m1( object      gate-1                                   (5.1.3.1.6)
    form        andgate
    modality    logical)
```

One could raise the question whether other display modalities might exist, and in fact this is the case, even in a very limited domain like circuit board display. Components like coils create magnetic fields which are larger than they are themselves. Other components (most notably other coils) might be sensitive to magnetic fields. A display of a circuit board in the *magnetic* modality would therefore be a variation of the physical display, with the following changes:

- Components creating a magnetic field will be displayed with a form that shows an approximation of the boundaries of their respective magnetic fields. This will result in showing them larger than they really are. It might even result in overlap between such components.
- Components sensitive to magnetic fields will be displayed in their correct sizes.
- Components that are neutral with respect to magnetic fields will not be displayed at all.

Another comparable display modality is the *thermal* modality. Such a display would show all the heat sources and all the heat sensitive components in two different colors, ignoring all other components. In general we will permit all the representation systems that were introduced in chapter 3.3 as additional modalities.

5.1.3.1.3. Modalities versus Partitions

Fig. 5.1.3.1.4 shows 4 case frames in two different modalities. One question that arises immediately is whether the representation with modality arcs is not redundant, and whether some sort of partitioning of the network [Hen79] would not be more appropriate, as shown in Fig. 5.1.3.1.5. It will now be argued that this view differs only graphically from the view in Fig. 5.1.3.1.4.

In SNePS every arc connecting two nodes has a converse arc associated with it which connects the same two nodes, however in the opposite direction. The label of the converse arc is identical to the label of the original arc, except for a *minus* (-) appended to the name. Normally the

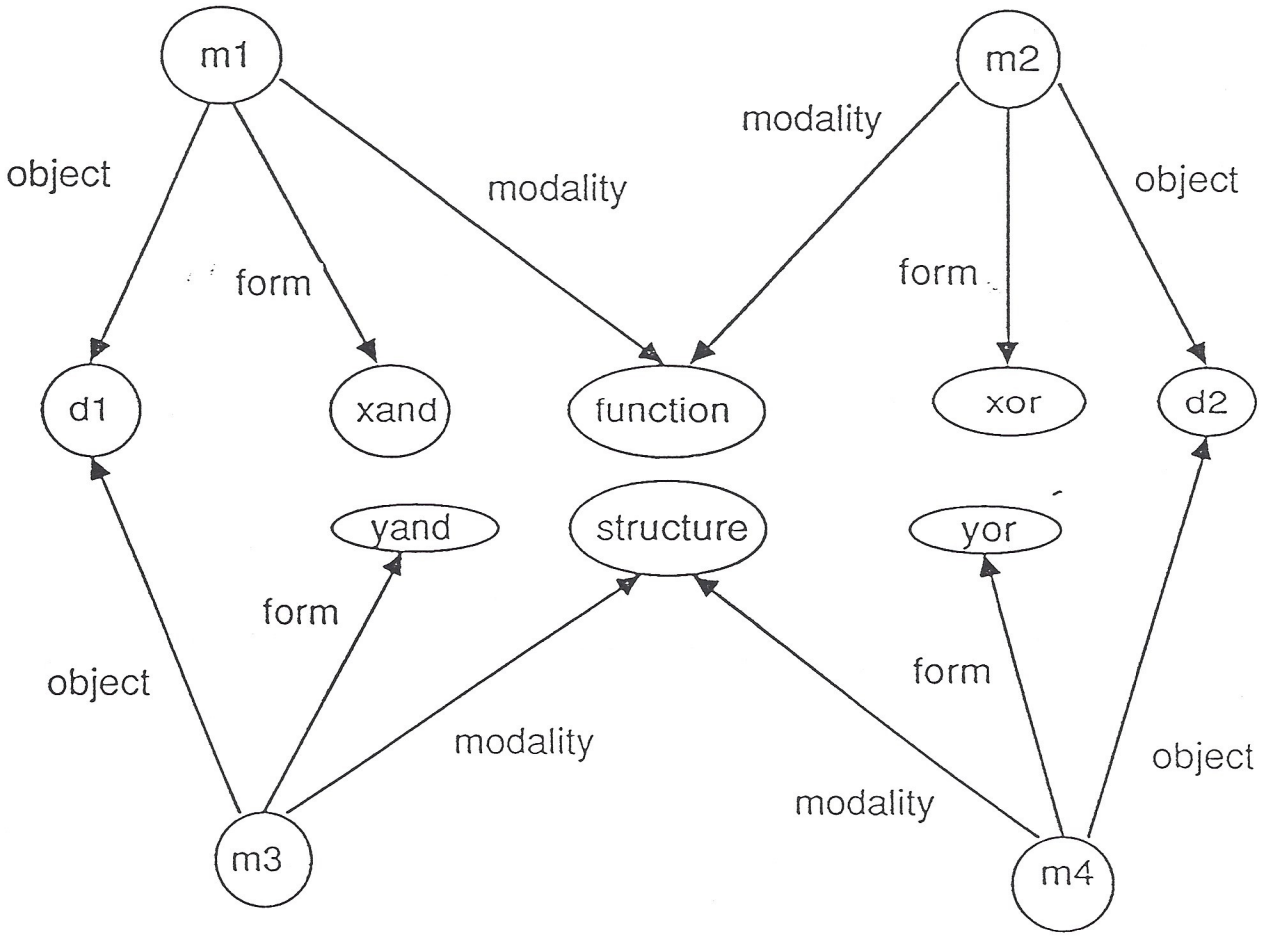


Fig. 5.1.3.1.4: Four case frames in two modalities.

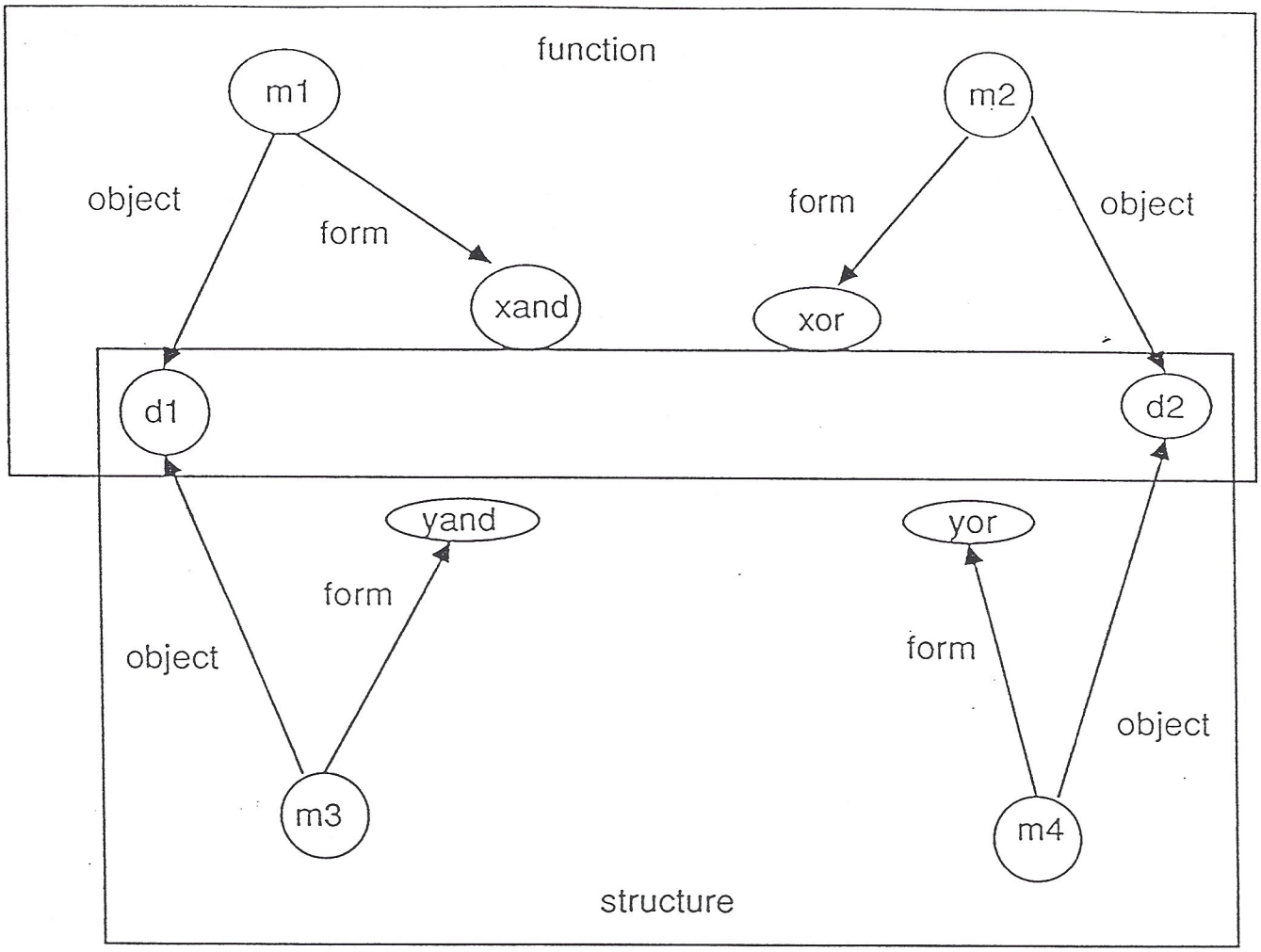


Fig. 5.1.3.1.5: A partitioned representation of modalities.

user is not confronted with this arc at all, not in build operations, not in screen displays and not in drawings of the network. Nevertheless this converse arc makes every node in the network accessible from every other node and is used in knowledge base retrieval operations [Sha79b].

In Fig. 5.1.3.1.6 the same picture is shown as in Fig. 5.1.3.1.4, however the modality arc is now replaced by the inverted arc *modality-*. Also the graphical layout has been changed such that a structure similar to Fig. 5.1.3.1.5 is created. In short, it becomes clear that the modality mechanism is isomorph to a partitioning of the network into separate areas, and although Fig. 5.1.3.1.5 seems more intuitive to some people, Fig. 5.1.3.1.4 captures the intent and the implementation equally well. As soon as several independent partitionings are introduced, diagrams of the form of Fig. 5.1.3.1.5 often cannot be maintained any more, while diagrams like Fig. 5.1.3.1.4 can

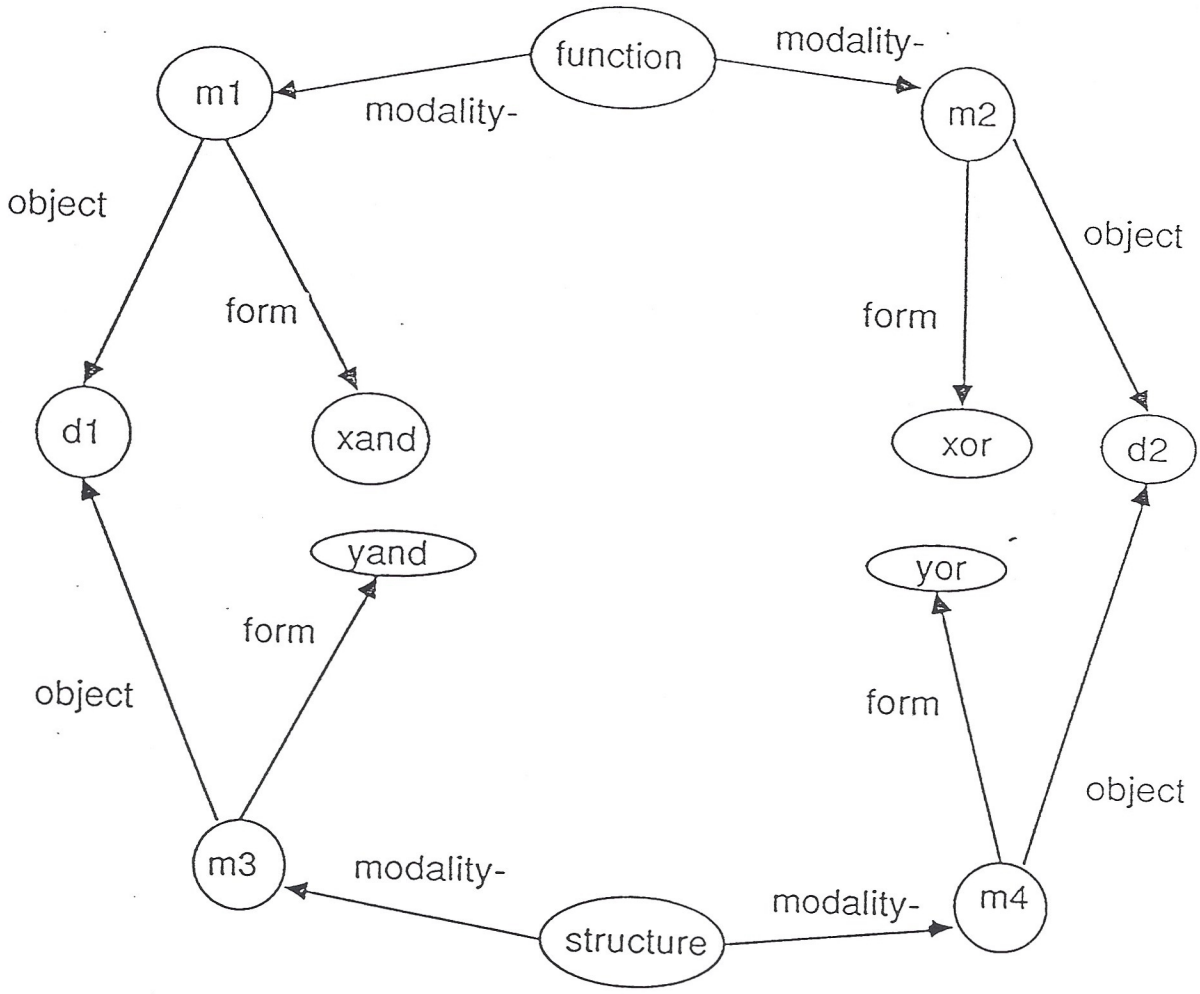


Fig. 5.1.3.1.6: Alternative representation of 5.1.3.1.4

be extended easily. Besides that, using modalities to achieve network partitioning is more parsimonious than to invent a completely new structural building block called a "partition".

5.1.3.1.4. Class Form with n Step Inheritance

As mentioned before, an object might inherit a form along a class hierarchy. Unfortunately one needs to present forms before classes to show the use of classes by a graphics processor, and classes before forms to give the descriptive semantics for form inheritance. We have decided to indicate the use of class structures in this section, but a separate section (5.1.3.5) will be reserved for the formal introduction of classes.

Syntax:

object	<object>	(5.1.3.1.7a)
type	<class>	
modality	<modality>	
sub-class	<class>	
class	<class-2>	
modality	<modality>	
sub-class	<class-2>	
	:	
	:	
class	<class-n>	
modality	<modality>	
class	<class-n>	(5.1.3.1.7n)
form	<form>	
modality	<modality>	

Semantics:

(5.1.3.1.7a) describes a simple class membership, meaning that <object> is a member of <class> under the modality <modality>. (5.1.3.1.7n) asserts that every member of the class <class-n> that does not have its own form has the form <form> under the modality <modality>. The intermediate structures will be discussed in the section on class hierarchies. <class> stands for the concept of a class, i. e. an entity that is by itself not displayable, but which has members[5] that are (potentially) displayable. All other syntactic variables in the above structure have been defined in the previous sections.

Assuming that there is no other network structure present that could have an influence on the display of <object>, then we can summarize the procedural effect of above structures in the following way. If the structure (5.1.3.1.7) is asserted, and the structure of (5.1.3.1.5) is not asserted, when requested to draw the <object> at location (x, y), apply the function <form> to x and y.

Example Network:

---

[5] We are not interested in empty classes or infinite classes.

```
m1( object     chip2                                   (5.1.3.1.8)
    modality   logical
    type       c-d2)

m2( sub-class  c-d2                                    (5.1.3.1.9)
    modality   logical
    class      ci-d2)

m3( sub-class  ci-d2                                   (5.1.3.1.10)
    modality   logical
    class      cf-d2)

m4( class      cf-d2                                   (5.1.3.1.11)
    modality   logical
    form       xor)
```

The node m1 represents the membership of chip2 in the class c-d2. The node m2 represents the proposition that c-d2 is a sub-class of the class ci-d2. The node m3 represents the assertion that ci-d2 is as sub-class of cf-d2. Finally the node m4 stands for the proposition that all members of the class cf-d2 have the form xor. All these assertions are valid for the logical display modality only. (Side note: the letter x is used to indicate a concept that also figures as a form primitive. An XOR gate would have the concept name xxor. Similarities with the terminology of the X window system are purely accidental). The nodes m2 and m3 are available for subclass and membership reasoning.

An example of a class form without intermediate steps would look as follows:

```
m5( object     chip3                                   (5.1.3.1.12)
    modality   logical
    type       nand)

m6( class      nand                                    (5.1.3.1.13)
    modality   logical
    form       xnand)
```

A drawing request for object chip3 with a requested display modality "logical" would result in the execution of the function xnand which will draw a nand-gate. m5 expresses the class membership of chip3 in the class xnand, and m6 represents the proposition that all members of the class nand have to be displayed using the form corresponding to the concept xnand.

A few final notes on forms follow now. All forms in this investigation are assumed to be rigid. No attempt has been made to deal with "prototypical forms" as found by Rosch for base level categories of objects [Ros78]. Forms for classes will be mentioned again in the chapter on classes (5.1.3.5). Most objects in the real world are constructed from parts, and we will devote a chapter to this subject (5.1.3.4).

5.1.3.2. Positions

5.1.3.2.1. Requirements for Position Representations

Position information always relates to a reference frame. How can we represent a reference frame in a knowledge representation system? After having achieved this, how can we then represent positions that refer to this reference frame?

People who talk about a graphics display might either talk about the objects in the world that are depicted, or about the icons on the screen. Assuming a three dimensional world, displays have to be created by projection on a plane. This constitutes a third space that a knowledgeable user might want to refer to. We would like to express positions in all these spaces with a unified mechanism.

A major problem in position specification is that people use two different methods to refer to space. They either refer to positions based on numerical coordinates, or they use what we want to call "fuzzy" or "conceptual" coordinates. Two examples of such conceptual coordinates would be "left" and "near". So a representation is necessary that captures numerical coordinates as well as conceptual coordinates in a unified framework. Numerical coordinates themselves might refer to different types of reference frames. Besides cartesian coordinates there are polar coordinates in 2-d, and cylindrical and spherical coordinates in 3-d. Whatever knowledge structure one uses, it should not be necessary to make a commitment concerning the coordinate type.

Another problem in interpreting spatial utterances is that people use absolute and relative position specifications. We argue that there are no absolute positions, therefore it is necessary to express absolute positions with the same mechanisms as relative positions. What was said before about two types of coordinates carries over to absolute positions. "Left" and "near" can be considered as relative conceptual coordinates, but terms like "top" or "center" express fuzzy absolute coordinates. Therefore, whatever structure is introduced to describe fuzzy relative coordinates must also express fuzzy absolute coordinates.

Although it was stated that we do not insist on a redundancy free representation for GDK, we would still like to find ways to eliminate repetitive items from the knowledge base, if we can find a well defined and algorithmic method to recover this information. Two accepted KR candidates for such knowledge economy are inheritance and default assumptions. We would like to find a knowledge structure for position representations that lets us make use of these methods in a way that agrees with cognitive ideas about object representation.

Another requirement that we want to raise is that our representation should help in identifying spatial invariants and, more importantly, describe positions in a way that maintains these invariants. This is an interesting problem because ranges of size ratios seem to be important in object recognition as well as in practical graphics applications. A good example for such a graphics application is the use of a viewport (window) on the viewing surface of a work station. Window managers usually permit to relocate and scale windows, but the result of scaling a window down is most often to limit the physical range of the document that is viewed in the window instead of scaling the window appropriately together with its contents.

An interesting language related problem is the use of terms like "behind" even when referring to a display of 2-d icons. How can a position representation sensibly interact with a reference frame representation to decide about the meaning of such a term, and how can it do this without keeping track of complete 3-d information?

We will use the next section of this chapter to discuss all the problems that are connected with reference frames. After that we will present our solutions for the other problems mentioned in this section.

5.1.3.2.2. Reference Frames

In understanding utterances about spatial relations the identification of a correct reference frame is often the first problem that has to be solved [Son76]. Computer vision research also has been looking at the problem of reference frame identification in some detail [Pin84]. Questions have been raised about the correct type of coordinate system, as well as about the correct handedness of the coordinate axes.

For NLG the problem of reference frame identification represents itself in the following manner. The screen that a person looks at naturally induces a coordinate system with axes parallel to the screen edges. There are a few reasonable choices for the center position, and we will assume a privileged screen coordinate system, having a horizontal right pointing x axis intersecting a vertical upward pointing y axis in the leftmost lowest pixel on the screen. A person may then use screen coordinates to describe the location of an object. Unfortunately screen coordinates alone are an insufficient device for NLG.

A normal graphics device has a certain addressing range. Typically one has pixels from 0 to 1024 in the horizontal direction, and from 0 to 800 in the vertical direction. If a person refers to a position which is beyond this range, for instance by using negative coordinates, she makes it clear that she is not interested in screen coordinates, but in plane or world coordinates. So reference frame identification means to determine the relation between the world coordinate system and the screen coordinate system.

To project an object of the world onto a screen two conceptual steps are necessary. First a projection plane is selected, and the object is projected onto the plane with beams orthogonal to it. Then an area on the projection plane needs to be selected and mapped onto the screen or onto a

viewport[6] on the screen.

The previous paragraph implies a number of different conceptual objects that we are operating with, and which therefore deserve consideration for being represented in the network. There can be several different world coordinate systems. One world coordinate system can be projected on several different projection planes. Although we have assumed a privileged screen coordinate system, we can nevertheless not prohibit the user to think in terms of a screen coordinate system that divides the given space into four equally sized quadrants. Because of the different "semantics" of plane and screen coordinates we need to associate each coordinate system with its type. In order to compute the projection from a world coordinate system to a projection plane we need to represent the plane in the world coordinate system. Given a projection plane, there might be different coordinate systems in it. For instance one might want to describe the plane with a cartesian or a non-cartesian coordinate system. So it is necessary to associate each coordinate system with a type tag. To permit arbitrary names for the axes, which is necessary if there are several systems of the same type, we need to give the correct order for the axes. Sometimes somebody wants to conceptualize one projection plane with two different types of coordinate systems, e. g. one cartesian, and one non-cartesian. In this case one plane has to be associated with two different coordinate systems. For the second step of display generation we might want to consider to represent the window that is projected, and the viewport it is mapped into, or alternatively we might want to talk about the scale factor and the shift vector that performs the necessary coordinate transformation.

For the first step of projection we will now show the most general possible conceptualization represented with case frames. First the representation of a coordinate space.

Syntax:

<space-description> : (5.1.3.2.1)
space <space-descriptor>

---

[6] It is now customary to refer to an area on the screen of a terminal as a window. Strictly speaking this is wrong, the correct term is viewport.

space-type <space-type>

Semantics:

The space denoted by <space-descriptor> is of the type <space-type>. <space-descriptor> denotes any concept of a coordinate space. <space-type> is one member of the set {world, plane, screen}. This definition does not make any statements about the coordinate system used in <space-descriptor>.

Syntax:

<coordinate-system> : (5.1.3.2.2)

space	<space-descriptor>
coord-sys	<coord-sys-descriptor>
coord-type	<coord-type>
first-axis	<axis>
second-axis	<axis-2>
third-axis	<axis-3>

Semantics:

The coordinate system <coord-sys-descriptor> represented by an atomic node is resident in the space <space-descriptor> and is of the coordinate type <coord-type> and has the three axes (<axis> <axis-2> <axis-3>) in exactly this order. <space-descriptor> denotes any concept of a coordinate space. <coord-sys-descriptor> denotes any concept of a coordinate system. <axis>, <axis-2>, and <axis-3> denote concepts of coordinate axes. <coord-type> is one of the atomic nodes {cartesian, polar, cylindrical, spherical}. <axis> will be interpreted as an X axis, if the <coord-type> is cartesian, and as an R axis otherwise. <axis-2> will be interpreted as a Y axis if <coord-type> is cartesian, and as a Φ axis otherwise. <axis-3> will be interpreted as Z axis, if <coord-type> is cartesian or cylindrical, ignored if it is polar, and interpreted as a τ axis otherwise. If <space-descriptor> is defined as a two dimensional space by its <space-description>, then the third axis will be ignored. We will consider the origin of the coordinate system as its reference point, therefore any reference to the position of the coordinate system is a reference to its origin.

Having shown methods to define a space and a coordinate system in it, it becomes possible to specify a world coordinate system as well as a plane coordinate system for projection purposes. To permit the computation of a projection we need to represent the plane in the world. One can represent the plane coordinate system by a vector to its origin and two vectors in the plane, denoting the directions of the plane coordinate axes. The plane itself is then spanned by its axes. Alternatively one can represent the plane independently of its coordinate axes, and represent the axes themselves with the same three vectors. The second representation is redundant, something that does not disturb us, as repeatedly noted,[7] and we will argue that it is a conceptually better representation.

Given that we permit one plane to have several coordinate systems, it becomes clear that the plane is on a lower level in the ontology that we are designing than the coordinate system embedded in it. It is therefore unnatural to define the plane by means of the higher concept of a coordinate system in the plane. We will therefore prefer the second representational solution. A good way to represent a plane in a cartesian coordinate system is by giving its intersections with the three coordinate axes. The three vectors that describe the plane coordinate system can be expressed as triples of numbers with units.

We will not show either one of these two structures, because in this investigation only a few special purpose positions of planes and plane coordinate systems will be used, and the general purpose representations are not necessary. We will also assume without loss of generality that all world coordinate systems have axes parallel to the direction of gravitation designated as the Y axis.

For notational purposes we will refer to world coordinate axes as X, Y, and Z, to plane coordinate axes as x_p, y_p, and to screen coordinate axes as x and y (See Fig.5.1.3.2.1). More precisely, x and y are used generically or with reference to the privileged screen coordinate system. The discrimination between these two choices will always be possible by context. In the program the

---

[7] Purists may eliminate appropriate components from the vectors.

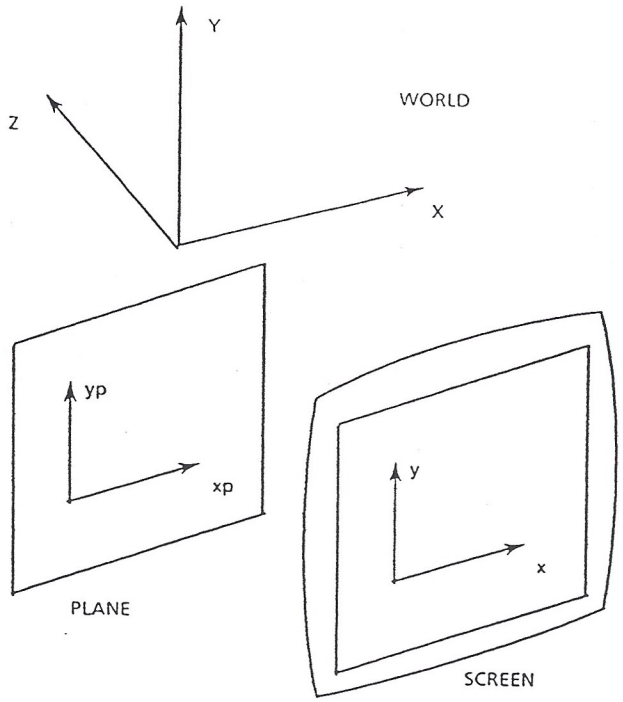


Fig. 5.1.3.2.1: Three different coordinate systems.

names of coordinate axes will be identical to the concept labels for them, and these, as we have shown, can be randomly chosen and asserted with the <coordinate-system> structure.

The following positions of projection planes relative to world coordinate systems will be permitted. (1) X parallel to x_p and Y parallel to y_p; this will be called a "front view". (2) X parallel to x_p and Z parallel to y_p; this will be referred to as a "top view". (3) X antiparallel to x_p and Y parallel to y_p; this is a front view with leftness defined by the computer. (4) X antiparallel to x_p and Z parallel to y_p; this is a top view with leftness defined by the computer.

The reasons for these choices are as follows. It has been stressed repeatedly that this work only concerns 2-dimensional NLG. Nevertheless it turns out that people often look at a 2-dimensional diagram with a preconception that this is really a projection of a 3-dimensional world. Imagine a screen with a vertical arrangement of two circles (see Fig.5.1.3.2.2). If one assumes that this scene represents a front-view, then one circle is clearly *above* the other circle. But if a person looks at the diagram as a map and at the two circles as two trees and pictures herself "below" the lower circle, then she will think of "upper" circle as being "behind" the lower circle! So some

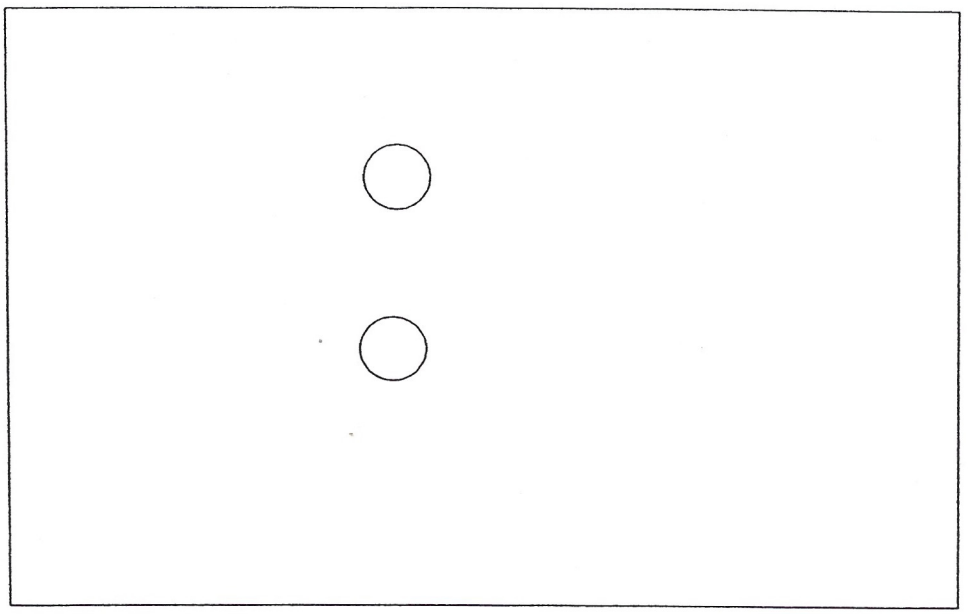


Fig. 5.1.3.2.2: An arrangement of two circles can be described in two ways.

natural language utterances can only be interpreted if one starts with the idea of a 3-dimensional world coordinate system. Luckily people do not seem to assume arbitrary projection angles, so it will be sufficient for us to use (1) and (2). Interestingly, if one does not want to specify "behindness" by concrete (= numeric) terms, the world coordinate system is not necessary at all, and one can talk about one object being behind another object even in reference to a screen coordinate system!

Choices (3) and (4) are due to the ambiguity of the terms "left" and "right" in conversations of two agents facing each other. Two dimensional coordinate systems are traditionally drawn such that the negative half axis corresponds to the left hand of the viewer, and the positive half axis to the right hand. Because a person could imagine that the computer is "looking at the screen" from behind, she might be tempted to interpret the term "left" as seen from the computer. This implies a reverse assumption about the x axis, which corresponds to a multiplication factor of -1. Therefore the x axis becomes antiparallel to the X axis.

The choice of the vector from the center of the plane coordinate system to the center of the world coordinate system as well as the mapping from an area on the projection plane to an area on the screen is of little importance from the knowledge representation point of view. The reason for that is as follows. Under normal circumstances NLG wants to display an object or a group of objects. This is different from "normal" computer graphics, where an area in the world (the window) is specified and mapped into a viewport on the screen.

An NLG user rather expects the program to find its own window that includes all the objects he wants to see, and map it correctly into a selected viewport. The necessary translation factor and the necessary shift vector are dynamically adapted at every single display request. In this computation the vector from the center of the plane coordinate system to the center of the world coordinate system functions only as an "uninteresting" constant. Choosing a vector orthogonal to the plane eliminates this constant. Note that our assumptions are still more rigid than e. g. Hinton's [Hin79] opinion that in vision reference frames might not have a center at all, just a set of preferred directions.

We will now present the knowledge structure that covers all the projection choices that we have considered interesting.

Syntax:

<projection> : (5.1.3.2.3)

coord-sys-w	<coord-sys-descriptor>
coord-sys-p	<coord-sys-descriptor-2>
view	<top-or-front>
leftness	<user-or-system>

Semantics:

The case frame (5.1.3.2.3) describes a projection. It asserts that a projection from the coordinate system <coord-sys-descriptor> to the (plane) coordinate system <coord-sys-descriptor-2> is done such that the following holds true. (a) The origins of the two coordinate systems are connected by a vector orthogonal to the plane. (b) The plane is located such that the projection

amounts to a <top-or-front> view. The only values that can be taken on by <top-or-front> are "top" and "front". In other words, for a front view the projection plane is parallel to the plane defined by the [X Y] plane, and for a top view the projection plane is parallel to the plane defined by the [X Z] plane. (c) The x axis in the plane is oriented such that a lateral term ("left") is interpreted from the view of <user-or-system>. The only values that can be taken on by <user-or-system> are "user" and "system". It is permissible to omit the slot containing <coord-sys-descriptor>. In that case the <view> structure is used only to interpret fuzzy natural language terms.

Without such an assertion the following questions would be meaningless to the system: "Do you mean left for me, or left for you?" and: "Is this a top view or a front view?". No other possible projections than the ones derived by combining "top" or "front" and "user" or "system" are permitted.

Above structure has the following effect on display requests. If required to display an object A "behind" an object B, and <top-or-front> is "top", then A will be displayed vertically above B. If the view specified is "front" then A will be displayed at the same location as B, such that B is drawn later and overdraws A. "In-front", "above" and "below" are interpreted analogously. If required to display an object "at the left", and <user-or-system> is "user" ("system"), then the object will be displayed in the left ("right") area of the screen seen from the user's point of view.

Example:

```
m1( coord-sys-w  world-1                                    (5.1.3.2.4)
    coord-sys-p  plane-2
    leftness     user
    view         front)
```

m1 asserts that projections from world-1 to plane-2 will be done such that a front view is achieved, and such that the term left is defined according to the user's sides. (5.1.3.2.4) exemplifies the default assumption that we will make in most of the rest of this dissertation.

5.1.3.2.2.1. Concrete Units in Reference Frames

Sizes and distances in the world will be specified in units like "inch" or "centimeter". Sizes on a graphics terminal are device dependent and specified in pixels. In order to correctly map from world units to pixels the system must know how many pixels correspond to one unit of length. To permit one knowledge base to work with several different devices it is necessary to include the device in the case frame.

Syntax:

unit	<unit-name>	(5.1.3.2.5)
pixels	<pixel-number>	
device	<device-name>	

Semantics:

A unit <unit-name> is graphically represented by drawing <pixel-number> pixels on a device <device-name>. <unit-name> is the concept of a linear measuring unit. <pixel-number> stands for the concept of an integer number. It is considered to be a special case of <value> (see Section 5.1.3.2.3). <device-name> stands for a node that represents the current graphics device. Any size specification in a unit like inch has to be transformed, using this mapping, to compute the correct size of pixels on the screen.

5.1.3.2.2.2. Fuzzy Units in Reference Frames

In natural language, descriptions of distances are usually given by people in terms of expressions like "far" or "near". These expressions are highly ambiguous. If a person A asks a person B whether it is far to the city hall, and the person B replies "yes" then we don't know anything about the actual distance. If A is sitting in a car then the term "far" will obviously have a different meaning from the case where A is walking on the sidewalk.[8] But even the case of the per-

[8] We ignore other sources of ambiguity, like "of which city?"

son on the sidewalk has to be differentiated. For a jogger a distance of two city blocks might be considered near, while for an old person with a walking stick the same distance is quite far.

The ambiguity of spatial distance terms is well known, and the use of norms to describe typical sizes has been reported in the linguistics literature [Bol77]. We will interpret the meaning of the terms "near" and "far" as a fixed ratio of the available space defined by the screen (or viewport) and the position of the reference object.

5.1.3.2.2.3. Modalities Revisited

At this point it becomes necessary to add a short note about the relation between modalities and spaces. Display requests are given by asking for an object. However this object might exist in several different spaces, and each one of these spaces might be projected onto several different planes. So therefore it would be necessary to specify an object together with a space.

We have found that the only common situation where one object occurs in two different spaces is when the spaces are conceptually different, like a logical and a physical space. Also projecting one space on two planes is useless under the limiting assumptions we have made about projection planes. If the two planes are orthogonal, then, because we are dealing with 2-d icons, one of the planes will show only line segments. If the two planes are parallel, then the two projections will only differ by an "uninteresting constant". The effect of this constant will be eliminated by the following window to viewport mapping. Therefore it is sufficient to use modalities to index the correct display.

The other alternative would be to make the modality part of the <space-description> case frame. But that would require e. g. forms to be defined with reference to a space, a quite unnatural solution. We will therefore use modalities as defined earlier on.

5.1.3.2.3. The Representation of Positions

Positions are the most complex phenomenon in graphical deep knowledge. Nevertheless it turns out that one can get by with a single structure to represent every possible case. In this section this overarching representational structure will be discussed. First we have to introduce the representation of a measurement which consists of a value and a unit.

Syntax:

<linear-measure> : (5.1.3.2.6)
value <value>
unit <unit>

Semantics:

<linear-measure> describes a measurement, such that <value> is the value and <unit> is the unit of the measurement. <value> may be any numerical value or an element of a small set of conceptual values, namely {left, right, above, below, near, far, behind, in-front}. We will refer to numerical values as "concrete", and to conceptual values as "fuzzy". <unit> may be any unit of length measurement, including one of a small set of conceptual units, namely {left-right, above-below, near-far, behind-front}, or it may be the concept of an object used as a unit (see section 5.1.3.2.9.1).

Syntax:

<angle-measure> : (5.1.3.2.7)
value <angle-value>
unit <angle-unit>

Semantics:

An <angle-measure> consists of an <angle-value> and an <angle-unit>, such that <angle-value> is any number[9] and <angle-unit> is any measuring unit for angles, like degrees or radi-

---

[9] More precisely <angle-value> stands for a node that represents the concept of a number that can specify an angle. We will somewhat relax the rigor of our specifications.

ans.

In the specification of a position we will make use of linear as well as angular measures, therefore we define the following alternative.

Syntax:

<measure> : (5.1.3.2.8)
<angle-measure> ! <linear-measure>

Later on we will make use of an area measure, so we include here.

Syntax:

<area-measure> : (5.1.3.2.9)
value <area-value>
unit <area-unit>

Semantics:

An <area-measure> consists of an <area-value> and an <area-unit>, such that <area-value> is any number or one of the fuzzy values {small, large}, and <area-unit> is any concrete measuring unit for areas or the conceptual unit small-large.

We now need to assign a direction to a measurement.

Syntax:

<component> : (5.1.3.2.10)
direction <axis>
measure <measure>

Semantics:

<component> describes an oriented measure, such that <axis> is an axis of a known coordinate system, and <measure> is a measure as defined previously. <axis> may refer to a linear axis, as well as to a conceptually bent axis, as they are made use of in non-cartesian coordinate systems.

Syntax:

<vector> : (5.1.3.2.11)

coord-sys	<coord-sys-descriptor>
component	<component>
component	<component-1>
component	<component-2>

Semantics:

(5.1.3.2.11) describes a <vector> that is defined in a coordinate system <coord-sys-descriptor> and that consists of three components <component>, <component-1>, and <component-2> which are all <component>s. In the most general case this vector will be defined in a 3-d space. However, as noted before, it is permissible to omit irrelevant or unavailable information, therefore a 2-d vector will be represented by omitting the slot for <component-2>.

Syntax:

<object-position> : (5.1.3.2.12)

object	<object>
relpos	<vector>
rel-to	<object-or-co>
modality	<modality>

Semantics:

<object> denotes an object concept. <object-or-co> denotes an object concept or a coordinate system concept. <object-position> describes the position of an object <object> by supplying a reference object <object-or-co> and a vector <vector> that has its starting point in the reference point of the reference object <object-or-co> and its ending point in the reference point of <object>. The <object-position> specified this way is valid for the modality <modality> only.

Syntax:

<class-position> : (5.1.3.2.13)

class	<class>
relpos	<vector>

rel-to <object-or-co>
modality <modality>

Semantics:

The <class-position> case frame defined in (5.1.3.2.13) defines the relative position for a class <class>. Every <object> that is a member of <class> and that does not own a position by virtue of an <object-position> case frame and that does not inherit a position from a sub-class of <class> has its position assigned by the <class-position> case frame. The position is specified by the vector <vector> positioned such that it starts in the reference point of <object-or-co> and ends in the reference point of <object>. The <class-position> is only valid for the modality <modality>.

Syntax:

<position>: (5.1.3.2.14)
<class-position> ! <object-position>

Semantics:

A position description <position> is either a <class-position> or an <object-position>. The exclamation mark denotes a BNF-like "or", but remember that this and all previously shown structures are *not* BNF structures, because slot-filler pairs may be arranged in any desired order.

We will now represent an <object-position> in its expanded form by replacing syntactic variables by appropriate sub-structures.

<object-position> : (5.1.3.2.15)

```
object   <object>
relpos   coord-sys    <coord-sys-descriptor>
         component    direction  <axis>
                      measure    value   <value>
                                 unit    <unit>
         component    direction  <axis-2>
                      measure    value   <value-2>
                                 unit    <unit-2>
         component    direction  <axis-3>
                      measure    value   <value-3>
```

unit <unit-3>
rel-to <object-or-co>
modality <modality>

(5.1.3.2.15) represents a proposition that, in modality <modality>, <object> is <value> <unit>s in the <axis> direction and <value-2> <unit-2>s in the <axis-2> direction and <value-3> <unit-3>s in the <axis-3> direction away from the position of <object-or-co> in the coordinate system given by <coord-sys-descriptor>.

Given a display request for <object> in the modality <modality>, <object-position> can be used to derive the position of <object>, if the position of <object-or-co> and the valid reference-frame <coord-sys-descriptor> are known. We will now show a number of examples that demonstrate how this general purpose position description can be used to solve the representational problems that we have set out to attack (see Section 5.1.3.2.1).

5.1.3.2.4. Concrete versus Fuzzy 2-d Descriptions

One of the problems that we have set out to solve was to create a representation that will capture concrete as well as fuzzy spatial descriptions. The following example shows <object-position> for a front view of the object gate-1 positioned relative to the object transformer-4 in the modality "function". Coordinates are concrete, i. e. numbers. The valid coordinate system is world-1.

```
m6( object      gate-1                                                    (5.1.3.2.16)
    relpos      m5( coord-sys          world-1
                    component
                    m4(   direction    X
                          measure      m1( value  1.2
                                           unit   inch))
                    component
                    m3(   direction    Y
                          measure      m2( value  0.8
                                           unit   inch)))
    rel-to      transformer-4
    modality    function)
```

We may read this example by saying that gate-1 is 1.2 inches to the right and 0.8 inches above

transformer-4 in the coordinate system world-1 under the modality "function". Due to space limitations the slot name "component" has been placed in a line by itself. This has no influence on the meaning of the structure. Note that the Z coordinate has been omitted, according to our earlier assumption that irrelevant information may be omitted. Also note that the names X and Y are purely mnemonic. The meaning is assigned to them in an appropriate reference-frame assertion.

Now, as an alternative we will present an example for a fuzzy relative position representation.

```
m8( object      gate-1                                                    (5.1.3.2.17)
    relpos      m7( coord-sys              world-1
                    component
                    m4(     direction      X
                            measure        m1( value   left
                                               unit    left-right))
    rel-to      switch-8
    modality    function)
```

We are assuming a special fuzzy unit "left-right" that has only two values, namely "left" and "right". This choice makes perfectly sense for the following reason. Consider an x axis with grid points at all integer numbers. If one is required to describe the position of a point on this axis and the point does not match any of the grid points, then the reasonable strategy used by most people is to describe the point by the grid point nearest to it. This is exactly what is done in the previous structure, except that the grid consists of only two points and the distances of the two points from the origin are not fixed.

Four additional fuzzy relative positions can be derived by combining measures on the left-right axis with measures on the above-below axis. The following structure represents "above-left".

```
m8( object      gate-1                                                    (5.1.3.2.18)
    relpos      m7( coord-sys              world-1
                    component
                    m4(     direction      X
                            measure        m1( value   left
```

```
                                                    unit    left-right))
                    component
                    m3(     direction       Y
                            measure         m2( value   above
                                                unit    above-below))
    rel-to      switch-8
    modality    function)
```

We will not permit combinations of behind-front and left-right, i.e. spatial terms like "in-front left", therefore we will not represent these combinations, although they are clearly representable in the given case frame.

5.1.3.2.5. Polar Coordinates

We have required that non-cartesian coordinates should be a possible representational alternative for positions. In this section we will show that our general purpose representation for positions can capture polar coordinates as well. Polar coordinates describe the position of a point by a distance from the origin and the angle between the vector from the origin to the point and the R-axis of the system. Polar coordinates are assumed to describe the position of points in the plane only. To describe the positions of points in 3-d space one has to add either a height coordinate above the [R Φ] plane, the Z coordinate, or an angle between the vector to the described point and the [R Φ] plane, commonly referred to as τ.

```
m8( object      LED-4                                                   (5.1.3.2.19)
    relpos      m7( coord-sys               world-r
                    component
                    m4(     direction       R
                            measure         m1( value   2
                                                unit    inch))
                    component
                    m5(     direction       PHI
                            measure         m2( value   60
                                                unit    degree))
    rel-to      resistor-7
    modality    function)
```

This example encodes the information that the reference point of LED-4 is 2 inches away from the reference point of resistor-7, and the vector pointing from the former to the latter has an angle of

60 degrees with the R axis of the system world-r. The information that Φ (PHI) is a non-linear axis is *not* contained in its name which is purely mnemonic, but in the case frame describing the coordinate system, as well as in the unit "degree" that has to be used appropriately if drawing of LED-4 is desired.

Polar coordinates can be used naturally to express one more fuzzy measure, namely near-far.

```
m8( object      switch-1                                              (5.1.20)
    relpos      m7( coord-sys          world-99
                    component
                    m4(  direction     R
                         measure       m1( value   near
                                           unit    near-far)))
    rel-to      transistor-99
    modality    function)
```

Note how beautifully this representations falls out of the combination of the basic assumption that unknown or irrelevant fillers may be omitted in SNePS with the polar conceptualization of space.

5.1.3.2.6. Absolute versus Relative Positions

The distinction between absolute and relative positions is the one that comes to mind first when looking at spatial knowledge. However, it can be argued that there are no absolute positions. Any positional information refers to a reference frame. Elimination of reference frame ambiguity has been mentioned before as a major step in language understanding. A speaker using an absolute spatial term assumes that the hearer knows what the correct reference frame is. The hearer also knows that there is a reference frame and makes an assumption about its identity.

The declaratively present knowledge of the current reference frame becomes obvious when a third person joins a discussion about spatial relations and asks for clarification. Both prior participants will be able to explain the reference frame they had in mind. Therefore the knowledge structure for an absolute position will differ from the structure for a relative position only by a marker that indicates that the given reference frame is at the same time the current reference frame. In the environment of SNePS this can be achieved by making the concept of the reference

frame the value of a SNePSUL variable called for instance CURRENT-REFERENCE-FRAME.

A SNePSUL variable is a variable that can store a node set and that is maintained by the SNePS top level interpreter. If such a variable contains a singleton node set, then one can say that it "points to the node" in the node set. So the CURRENT-REFERENCE-FRAME variable points to the node that represents the concept of the currently valid reference frame. A similar solution has been used by Almeida [Alm87] to mark the current reference time. This has been referred to as the "NOW point". The use of structures that describe absolute positions reported in [GeS87] has to be seen as a convenient abbreviation which is satisfying for engineering purposes, however not as a cognitive model.

5.1.3.2.6.1. Fuzzy Absolute Positions

A sentence like

Put the multiplier at the top. (5.1.3.2.21)

gives the impression of a fuzzy absolute position, because the reference object "screen" is not explicitly mentioned. We will continue to use the term "fuzzy absolute positions", keeping in mind that they are in fact positions relative to the screen and show the representation of the set of fuzzy absolute positions {top, bottom, left, right, upper-left-corner, upper-right-corner, lower-left-corner, lower-right-corner, center}. More precisely we will discuss the positions "top", "upper-left-corner", and "center" separately. All other positions are analogous to these three.

The screen a person is looking at naturally induces the existence of several objects, e. g. the object "upper-edge". We will require that the reference point of this object be in the middle of it. We claim, that an object is at the top of a screen, if it is near to the upper-edge. A representation for "near" has already been introduced before, so the necessary structure follows immediately.

```
m8( object      gate-9                                          (5.1.3.2.22)
    relpos      m7( coord-sys          screen-coord-sys
                    component
```

```
                    m4(   direction    R
                          measure      m1( value   near
                                           unit    near-far))
rel-to      upper-edge
modality    function)
```

Likewise an object in the upper left corner is near an induced object that we want to call the upper-left-corner of the screen. screen-coord-sys refers to the privileged screen coordinate system. But what about an object at the "center"? There are two conceptual ways to deal with the center location. (1) We can assume that "center" is also one of the objects that are induced by the screen, and that in fact every object with a horizontal and vertical symmetry axis induces a (virtual) object "screen-center". In this case the same structure as before will do.

```
m8( object    coil-7                                                  (5.1.3.2.23)
    relpos    m7( coord-sys              screen-coord-sys
                  component
     m4(      direction   R
                          measure      m1( value   near
                                           unit    near-far))
    rel-to    screen-center
    modality  function)
```

Note that the quality of this conceptualization is inversely proportional to the distance of the reference point of the object from its own center. Therefore we have required before that the reference point should not be outside of the object.

The other alternative to represent the center position of an object makes use of four (!) position case frames and two additional case frames. An object is at the center of the screen if it is below the top, right of the left edge, left of the right edge, and above the bottom edge. This description would cover any position inside of the screen. "Near" and "far" are of no help at this point. We would have to introduce four distance tokens for the four distance slots, and assert with two additional case frames that pairs of these distance-tokens are coextensive. This last task is traditionally achieved in SNePS with the "equiv" arc [ShM82]. We will show three of these six structures, the others follow analogously.

```
m8( object    AND-1                                                   (5.1.3.2.24)
    relpos    m7( coord-sys              world
```

```
                    component
                    m4(     direction      Y
                            measure        m1( value   below
                                                unit    above-below))
                    component
                    m6(     direction      R
                            measure        m3( value   distance-1)))
    rel-to      upper-edge
    modality    function)

m18( object     AND-1                                                  (5.1.3.2.25)
     relpos     m17(        coord-sys      world
                    component
                    m14(    direction      Y
                            measure        m11( value   above
                                                unit    above-below))
                    component
                    m16(    direction      R
                            measure        m13( value   distance-2)))
     rel-to     lower-edge
     modality   function)


m(20 equiv      distance-1                                             (5.1.3.2.26)
     equiv      distance-2)
```

The second solution is unpleasant because it requires a large amount of structure, something not in line with the linearity principle; after all language describes the center of a box with a single word. We also have to introduce unspecified distance tags (distance-1, distance-2 etc.), objects that function as place holders and whose epistemic status is at least doubtful. On the other hand, what gives us the right to introduce a virtual object at a point just to save our representation?

Luckily, cognitive science has been investigating virtual objects for quite some time. Some of these phenomena are known as "subjective-contour illusions". In experiments [Wal87] subjects perceive certain locations in a class of diagrams as brighter than other locations, although no real difference in brightness exists. Although we do not know about a perceived brightness difference for the center of a box, we feel justified by these results in making the center of a box a virtual object that "comes with every box".

5.1.3.2.7. Explicit versus Deduced Reference Objects

One of the requirements that we mentioned earlier was to design our position case frame so that it would be possible to eliminate "blatantly redundant" information from it. One candidate for this operation is the reference object. Considering e. g. an electrical circuit board, it is quite clear that all components can be described well by placing them relative to the board they are mounted on. If the explicit representation of this information is eliminated, we need a reliable method to recover the correct reference object.

In a later chapter (5.1.3.4) part hierarchies, a very basic structure of the cognitive system, will be introduced. If an object is placed with relative coordinates and no reference object is asserted, then the logical object to look for is the object it is part of, i.e. the "super-part". This choice is supported by experimental work of Kosslyn [Kos80]. Kosslyn has concluded from work with subjects, that in humans positions of parts seem to be stored in reference to the object they are parts of. Kosslyn himself has used this assumption in his simulation program reported in the same reference.

We would like to add another argument in favor of such an organization of object knowledge. This argument is based on the well known limitations of short term memory [Mil56]. Given a main object M with n parts in a one level part hierarchy. Given also a cognitive agent that tries to retrieve the positions of all the parts, presumably in order to assemble an internal image. Now consider two possible forms of knowledge organization: (1) Each object has a relative position specified relative to the "main object". (2) Each object has its position specified relative to one of the other n-1 objects or relative to the main object, such that every object is anchored directly or indirectly at the main object. No limiting assumptions about these spatial references are made, except that they form a tree (i. e. no circular dependencies).

Assuming that a position has to be kept in STM in order to do a computation, it is necessary (in case 1) to keep the absolute position of the main object in STM, and to move the relative positions of all the other objects there (at least temporary). At no time are there more than two posi-

tions necessary in STM. In case 2 an object B might be placed relative to an object A. A in turn might be placed relative to the main object. In order to compute the position of B it is necessary to maintain 3 position values in STM, one for M, one for A and one for B. If the chain length of relative positions is permitted to grow, a point might be reached where the number of necessary elements exceeds STM capacity. Alternatively, if one wants to assume that only the positions of the two objects that are immediately involved in the computation have to be resident in STM, then the main object would be bumped as soon as object B is loaded. Later on it might be necessary to reload the position of the main object. Under these assumptions the effort of computing all part positions grows due to additional "paging". So organization (2) is temporally or spatially inefficient, resulting in the expected argument for organization (1).

The following structures will be explained without giving details on the part hierarchy which is discussed in its own section.

```
m25( object          board-1                                    (5.1.3.2.27)
     modality        logical
     part-relation   real-part
     sub-object      gate-1)

m8( object    gate-1                                            (5.1.3.2.28)
    relpos    m7( coord-sys        board-world
                  component
                  m4(  direction   X
                       measure     m1( value   2
                                       unit    inch))
                  component
                  m5(  direction   Y
                       measure     m2( value   0.5
                                       unit    inch))
                  component
                  m6(  direction   Z
                       measure     m3( value   0
                                       unit    inch)))
    modality  function)

m18( object     board-1                                          (5.1.3.2.29)
     relpos     m17(        coord-sys   board-world
                    component
                    m14(  direction   X
                          measure     m11( value   10
                                           unit    inch))
```

```
                    component
                    m15(    direction   Y
                            measure     m12(  value   8
                                              unit    inch))
                    component
                    m16(    direction   Z
                            measure     m13(  value   0
                                              unit    inch)))
        rel-to      board-world
        modality    function)
```

m8 represents the relative position of gate-1, but no reference object is given. m18 represents the relative position of board-1 in the board-world. Finally m25 represents the fact that gate-1 is a part of board-1 and therefore board-1 is a super-part of gate-1. Because gate-1 has no reference object specified, its *immediate* super-part will be assumed as the reference object. Remember also that the origin is considered the reference point of a coordinate system, therefore m18 is correctly specified.

5.1.3.2.8. Own versus Inherited Relative Positions

Another way to eliminate information from a case frame is by permitting its inheritance along a class hierarchy. As noted before, a separate section will be devoted to class hierarchies, here only the aspects that relate class hierarchies to positioning tasks are discussed.

Given a circuit board that functions as a main object and has components as its parts, it is natural to assert relative positions relative to the board (see last section). However with several boards of the same type it is not necessary to store this relative position for every gate. If the gate is a member of a class of gates, and if all gates of this class have the same relative position relative to their board, then it is enough to assert the position information with the class. Practically that means that a position vector is only specified once, namely with the class, and omitted in all the object-positions. For the position specification of a class, refer back to (5.1.3.2.13). The position specification of an object degenerates then to the following pleasantly simple structure.

```
m8(  object     OR-3                                    (5.1.3.2.30)
     rel-to     OR-2
```

modality function)

The following example network represents a class membership.

```
m2( type       and-gate                                    (5.1.3.2.31)
    modality   logical
    object     and-gate-0)
```

The class "and-gate" would be placed by an instance of (5.1.3.2.13). "And-gate" is not the class of all and-gates, but it is a sub-class of all and-gates. This is the case because it describes a class of and-gates which *are positioned* in a certain way on a board of the class of which board-1 is a member. The following structure would connect the class "and-gate" with a super-class comprising and-gates in any position and is an example of our notation for sub-class/super-class relations.

```
m5( sub-class  and-gate                                    (5.1.3.2.32)
    modality   logical
    class      all-and-gates)
```

5.1.3.2.9. Coordinate Unit Types

Besides the description of fuzzy absolute positions, the identification of positional invariants has been the most challenging and rewarding area of the analysis of positions. Coordinate units used in scientific as well as practical applications, whether engineering, geography, mathematics, masonry,... make use of a predefined and publicly known base unit like the inch. The base unit is independent of both the objects that are positioned relative to each other. The distance is expressed as a multiple of the base unit. This is not the only way positions could be expressed. One could use the *size* of either one of the involved objects of a relative position specification as the base unit (or even of any object in the system).

There are practical reasons to permit such types of coordinate units. One such reason is the scaling problem for objects with parts. As is well known from computer graphics [FoD83] an object which is not at the center of the current coordinate system will not only be scaled but also relocated by a scaling operation. This is undesirable for the following theoretical reason. We view the scaling of an object as an operation that expresses a size attribute of the object. (Attributes

will be treated in Section 5.1.3.3). This attribute applies to each part separately and is independent of its position. More precisely scaling is supposed to be done for each object relative to its own reference point.

In order to correctly scale a whole group of parts it is therefore necessary to recompute the position of each part, which makes the display algorithm unwieldy. Body coordinates come to the rescue. If the relative position of a part is already given in *multiples of its own size* then any scaling of the part will not change its relative position! In other words, we can maintain our theoretical claim about scaling as being an attribute of an object, without paying a penalty in terms of the complexity of scaling a group of parts.

This observation is especially useful in connection with view-ports on a screen, because if all positions are given in multiples of the size of the viewport, then the resizing of the viewport does not require any special scaling algorithm for the positions of all displayed objects. We consider this approach to viewport resizing far more interesting than the computationally easier solution of displaying a different, potentially smaller segment of the document under the window.

Another argument for using an object-size as a base unit for specifying distances between parts is the importance of proportions in object perception and recognition. The recognition of objects and especially of characters depends not on absolute sizes, but on ratios. A large L can be recognized as well as a small L, but shortening only the horizontal segment will lead to the danger of a confusion with an I. In the same way if people see a giant, then they have no problem recognizing him as human, "just bigger". If one would compute the ratio of arm length to the distance of the arms from the spine for a large segment of the population, then a certain range of values would result. A giant or a midget would have their ratios somewhere in this range, and therefore can be recognized as humans. (Of course many other ratios will be used for recognition too).

In short, absolute information about part positions, does not have to be preserved for recognition purposes; it is more interesting to look at proportions. Relative positions between parts, which are measured in body units of one of the involved objects therefore express a set of valuable

invariants, and it is of interest to create a graphical deep knowledge structure that preserves them. (Only one value is preserved for each relative position, not a whole range.)

We have opted to permit any object in the system as base object, but coordinates based on the size of an object and coordinates based on the size of its reference object show a pleasant side-effect, they actually simplify the position representation used! This is the case, because no separate node for a unit is necessary, rather one of the object concepts itself functions as unit.

The crucial difference between absolute units versus the use of the object or the reference object as unit is *not* that a third object is involved for the first type, but that this third object *is never scaled* together with the other objects. We will first show the case frame for body coordinates, i. e. coordinates that use the object itself as unit.

					(5.1.3.2.33)
object	<object>				
relpos	coord-sys	world			
	component	direction	X		
		measure	value	<value>	
			unit	<object>	
	component	direction	Y		
		measure	value	<value-2>	
			unit	<object>	
	component	direction	Z		
		measure	value	0	
rel-to	<object-or-co>				
modality	<modality>				

Reference object body coordinates follow now.

					(5.1.3.2.34)
object	<object>				
relpos	coord-sys	world			
	component	direction	X		
		measure	value	<value>	
			unit	<object-2>	
	component	direction	Y		
		measure	value	<value-2>	
			unit	<object-2>	
	component	direction	Z		
		measure	value	0	
rel-to	<object-2>				
modality	<modality>				

The simplification of the structure comes about by eliminating one node, namely the independent unit node (like the "inch" node). Clearly in (5.1.3.2.34) the reference object has to be constrained

towards being a real <object>. Procedurally the extent, i.e. the smallest enclosing rectangle with sides parallel to coordinate axes is used to interpret body coordinates. An x coordinate is interpreted as multiple of the x-length of the extent, and a y coordinate is interpreted as multiple of the y-length of the extent of <object> (or <object-2>, whichever is specified).

5.1.3.2.9.1. Fuzzy Coordinate Unit Types

One problem we have set out to solve was to capture concrete as well as fuzzy coordinates with the same representation system, and the introduction of objects as size units forces us to look for the applicability of this idea to fuzzy units. It turns out that in fact fuzzy body coordinates can be used to model expressions like *too near* and *too far*, i. e. expressions that have to do with the reach of a human arm or mechanical device. Consider e. g. the reach of two boxers. A boxer stands *too near* to his opponent in a defense situation, if the other one can reach him, while he is *too far* if he can't. On the other hand, in an attack situation, a boxer is near if his own arms can reach the opponent and far otherwise. Therefore the meaning of "too near" and "too far" depends on the arm length of the two fighters.

If one boxer has longer arms than the other one, than the two boxers might be standing too near from one's point of view and too far from the other's. Given that the absolute distance between them is identical, the fuzzy units they are using must be different, each man using his own arm length. The same is true for the reach of a robot arm or the distance planes from an air craft carrier can fly. As well known, something that is "too big" might still be very small, and something that is "too far" might actually be pretty near, so the following structures do not model nearness per se. Our interpretation of "too near" will therefore be as follows. If an object is nearer than the size of the unit-object measured both in the same dimension, then it is too near, otherwise it is too far.

```
m8( object      arm-1                                          (5.1.3.2.35)
    relpos      m7( coord-sys              world
                    component
```

```
                    m4(   direction     R
                          measure       m1( value   far
                                            unit    arm-1)))
rel-to      face-2
modality    physical)
```

In the above example arm-1 is too far away from face-2, relative to its own length.

5.1.3.2.10. Screen, Plane and World Coordinates

We have required that the same knowledge structure can be used for screen, plane and world coordinates. This is in fact the case. The type of coordinate system is contained in the case frame describing it, but not in the position representation. The three different spaces are interesting for the following reason. If a position description applies to an object in a world coordinate system, then it is possible to create different views depending on the projection plane that is selected. If a position is specified in plane coordinates, then one has no more choice concerning the view, and plane coordinates are by nature two dimensional. However, a display on the projection plane might be too large to fit a given viewport on a screen. It is therefore permissible to do scaling or shifting to move the display into the viewport. If a position is given in screen coordinates, then no more coordinate transformation for viewing purposes is permitted. So each coordinate space behaves differently with respect to the operations necessary and permissible for display purposes.

While concrete coordinates are permissible for any type of coordinate system, we have limited ourselves to describing fuzzy position representations in relation to the screen. The reason for this is that the screen itself supplies the necessary reference frame that gives meaning to a term like top. The limited size of the screen also makes terms like "near" less ambiguous. In the real world Venus is quite near from us, compared to Pluto. On the other hand, two atoms are quite far away from each other, compared to the distance between protons. The real world just supplies too many orders of magnitude, each one having its own "near" and "far".

5.1.3.2.11. 2 1/2 Dimensional Representation

Although we have been assuming 2-dimensional structures in most of this dissertation, we do not want to exclude a user from talking about an object "behind" another object, even in a front view. One way to accommodate this is to use a graphics technique called 2 1/2 - d display. This does not refer to Marr's 2 1/2 - d [MaN78], but can best be imagined as a number of sheets on top of each other. It is therefore necessary to assign different 2-dimensional objects to different sheets. This is achieved by using the Z coordinate of a world coordinate system.

Our abstract graphics machine will make the following assumptions to deal with different layers. (1) Objects are in principle plane. Therefore, if the position of the reference point of object A has a Z value less than the Z value of the position of the reference point of object B, then *all* points of A have positions with Z values less than the Z values of the positions of all points of B. (2) The layering of objects decides about the order of drawing them. An object C with a large Z coordinate is considered to be far away from the viewer, and therefore will be drawn first. An object D with a small Z coordinate will be drawn later, and if it shares part of the [X Y] space with C, then D will overdraw C and effectively create the impression of an object "behind" another object. This is the point where our assumption of permitting only a few special projection plane positions becomes most crucial. Effectively we are expecting the projection plane to be parallel to the sheets that are displayed. This trivializes the projection algorithm. On the other hand, it is precisely the "behind" phenomenon that forces us to maintain three dimensions.

The layer model, with all its limitations, is quite useful for practical applications. For instance physical (structural) circuit board displays can be modeled by one layer containing the wiring on the surface of the board, while the components can be assumed to be in a layer above the board. The following structure shows a concrete example that expresses two objects in two different layers. This assumes again that the projection plane is parallel to the [X Y] plane, something expressed in the reference-frame descriptions.

m8(object wire-1 (5.1.3.2.36)

```
relpos      m7( coord-sys          world
                component
                m6(     direction  Z
                        measure    m3( value  1
                                       unit   inch)))
rel-to      chip-4
modality    structural)
```

To express a fuzzy relation, the two valued unit behind-front can be used.

```
m8( object      wire-1                                              (5.1.3.2.37)
    relpos      m7( coord-sys          world
                    component
                    m6(     direction  Z
                            measure    m3( value  behind
                                           unit   behind-front)))
    rel-to      chip-4
    modality    structural)
```

5.1.3.3. The Representation of Attributes in Graphical Deep Knowledge

5.1.3.3.1. Types of Attributes

Objects have attributes, pictures have attributes, and users make different judgements about what they consider to be relevant. From this situation a number of different representational possibilities emerges. For our purposes an object can have three different types of attributes: intrinsic-visible attributes, accidental-visible attributes, and invisible attributes. *Intrinsic-visible* attributes are attributes of the form of an object that will automatically be displayed whenever the object is displayed. For example a symmetric 2-d object will appear symmetric whenever displayed. *Accidental-visible* attributes can be varied in displaying an object, without modifying its form. We consider color and orientation as accidental-visible attributes. *Invisible* attributes cannot be displayed at all, but they can be symbolized. The faultiness of a component in a circuit board is an example of an invisible attribute.

Pictures can have four different types of attributes: representative attributes, symbolic attributes, accidental attributes and situational attributes. A *representative* attribute of a picture stands for a corresponding visible attribute of an object. A *symbolic* attribute of a picture stands

for a not corresponding (usually invisible) attribute of an object. An *accidental* attribute carries no information about the object. Accidental attributes should be avoided whenever possible. Finally a *situational attribute* describes a temporary attribute which is characteristic of the whole object viewer system and not just the object itself.

Unfortunately the attribute itself does not tell to what type it belongs, and the attribute of color qualifies for all four classes. If a banana is displayed as yellow, then the color is representative. It represents an object that is in fact yellow. If a faulty circuit is represented as red, then this is a symbolic attribute, standing for a different object attribute namely faultiness. If a banana is drawn white on a black and white graphics terminal, then this is an accidental attribute. It does not express anything. Finally, if a circuit in a circuit board display is usually displayed in blue, but temporarily the display is changed to red to catch the attention of a user, then the red color becomes a situational attribute. A favorite situational attribute on graphics devices is *blinking.*

The third factor in this analysis is the viewer. An object might have invisible attributes which are of interest. Therefore some symbolic mode of representation for such an attribute is required. Unfortunately it is situation dependent what is of interest and what isn't. Because real life is more vivid than computer graphics, the number of choices for object attributes that are visible is already much larger than the number of choices for available pictorial attributes. If one adds the fact that some pictorial attributes have to be used as symbolic attributes one always has to deal with a representational mapping that has a much smaller range than domain. This is the problem of reductionist representation in a different guise.

The only method of dealing with reductionist graphical representations is to relinquish the requirement for stable bindings between object attributes and pictorial attributes. A user can tailor a representation to his requirements by assigning object attributes in order of importance to available pictorial attributes until he runs out of pictorial attributes. It should be pointed out that this free mapping between object and picture attributes constitutes an asset of knowledge

based NLG systems. (Details about this observation will follow shortly, see Section 5.1.3.3.3.)

Another way to analyze attributes is according to the number of argument positions. Although there are very few clear cut cases, it seems reasonable to represent faultiness of a device as a binary attribute. The device is either faulty or it isn't. In other words, this is an attribute that does not have any values. Therefore it does not have any argument positions either. Alternatively one can view faultiness as an attribute with one argument position which can receive one of two values, "faulty" or "working".

Other attributes can better be captured by an attribute value from a small base set. Color is such an attribute. Although one could deal with 10 binary attributes of the form *being-red* versus not being-red, being-blue, versus not being-blue, etc., etc. it seems more natural to talk about the attribute color which can have one of 10 attribute values. To nomologically better differentiate the attribute from the attribute value we will refer to the attribute itself as *attribute class* and to the *attribute value* as such or as *argument position.* The semanticist Lyons [Lyo77] refers to this distinction as "bipartite sense-components consisting of (i) a superordinate marker taken from the set M = { SEX, COLOUR, AGE, SPECIES,....} and (ii) a subordinate marker, μ, specifying which particular location within the domain denoted by the superordinate marker is denoted by the subordinate marker." (p. 325).

Some attributes might have arguments which are subsets of integers or of reals. For instance an object might be rotated relative to its normal position. This object would then have the attribute class "rotated" with a real number as an attribute value. While one can vary this number from 0 to 360, or from -180 to 180 both of these representations are more abstract than the representations people seem to prefer. According to our introspection a representation is more natural that uses numbers from 0 to 180 and left/right instead of +/-. This invites the representation of attributes with two argument positions, namely the sense of rotation, as well as the degree. The first argument position will be taken by a real number between 0 and 180, the second one by one of the values "left" or "right". The analog extension to three or more argument values

presents no difficulty.

If one combines attributes with forms then two ways of interpreting the relation between these two entities become necessary in the presence of an abstract graphics machine. An attribute may be strictly descriptive, i. e. describe a feature that is incorporated in the form of an object. We will refer to such attributes as *unmapped* attributes. Alternatively an attribute might be used "modificatory", to change the form of an object which has been inherited along a class hierarchy. This will result in a modified graphical display. We will refer to such an attribute as a *mapped* attribute. Two separate sections will distinguish between those two.

The last way to discriminate between different attributes that we want to discuss is the distinction between *relative* attributes and *absolute* attributes. An attribute like "faultiness" of a device is absolute. It is possible to decide that the device is faulty, without any reference to another device. Similarly a basic decision about "color" can be established[10] without a reference object. On the other hand, an attribute like "large" requires a reference class to become meaningful.

5.1.3.3.2. Simple Attribute Representations

We will initiate this section with the representation of an absolute attribute with one argument position.

Syntax:

<a-attribute-1> : (5.1.3.3.1)
atrb-cls <attribute-class>
atrb <attribute-value>

Semantics:

(5.1.3.3.1) is a structured individual describing an absolute attribute with one argument position,

---

[10] We make a number of simplifying assumptions here, including day light and a community of speakers that agree on basic color terms.

whereby <attribute-class> denotes any concept of an attribute, and <attribute-value> denotes any concept of an attribute value belonging to this <attribute-class>. Here and in all future uses of <attribute-value> it is permissible that <attribute-value> be itself a structured individual. It is assumed that one object may have only one <attribute-value> for each <attribute-class>. The next structure exemplifies an absolute attribute with two argument positions.

Syntax:

<a-attribute-2> : (5.1.3.3.2)

atrb-cls	<attribute-class>
atrb1	<attribute-value>
atrb2	<attribute-value-2>

Semantics:

(5.1.3.3.2) is a structured individual describing an absolute attribute with two argument positions, whereby <attribute-class> denotes any concept of an attribute, and <attribute-value> as well as <attribute-value-2> denote concepts of attribute values.

The structure for an attribute with three argument positions follows analogously. The structure for a zero-value attribute <a-attribute-0> is created by omitting the <attribute-value> slot from the <attribute-1> definition. With this we can define the structure of an absolute attribute <a-attribute> as a choice.

<a-attribute> : (5.1.3.3.3)

<a-attribute-0> ! <a-attribute-1> ! <a-attribute-2> ! <a-attribute-3>

We will now turn to the representation of relative attributes.

Syntax:

<r-attribute-1> : (5.1.3.3.4)

rel-atrb-cls	<attribute-class>
atrb	<attribute-value>
ref-set	<reference-set>

Semantics:

(5.1.3.3.4) is a structured individual describing a relative attribute with one argument position,

whereby <attribute-class> denotes any concept of an attribute, and <attribute-value> denotes any concept of an attribute value belonging to this <attribute-class>. <reference-set> denotes any concept of a group of objects, being either a <class> of objects, or a structured object with parts. It is assumed that one object may have only one <attribute-value> for each pair of an <attribute-class> and a <reference-set>. If an <r-attribute-1> case frame is given without a <reference-set>, then it is assumed that the immediate super-class of the object provides the reference. (This relies on a non-tangled class hierarchy).

Above interpretation of a default is in complete analogy with relative positions where a missing reference object is derived from the part hierarchy. In fact this choice seems quite natural, considering the linearity principle again. If somebody says that "Joe is tall", we will assume that Joe is a human, and that he is probably in the area of 6 feet and above. Therefore we will not represent the reference class explicitly in the system. If somebody wants to preempt the default interpretation he has to supply a reference set. Such would be the case in a sentence like "Joe is tall for a kid of four years", or "Joe is small for a basket ball player".

The extension of this representation to two, three or zero attribute values raises no problems, and we can summarize being an attribute as being either a relative attribute or an absolute attribute.

<r-attribute> : (5.1.3.3.5)
<r-attribute-0> ! <r-attribute-1> ! <r-attribute-2> ! <r-attribute-3>

<attribute> : (5.1.3.3.6)
<r-attribute> ! <a-attribute>

After clarifying what an <attribute> is, we now have to assign it to an an entity in our knowledge base.

Syntax:

<attribute-assignment> : (5.1.3.3.7)
patient <patient>
attr <attribute>
modality <modality>

Semantics:

The <attribute-assignment> case frame asserts that a patient <patient> has an attribute <attribute> in the modality <modality>. <patient> represents the syntactic class of all units that may receive an attribute. It may be an <object>, a <form>, a <class>, or a <feature>. The last two syntactic classes will be introduced below.

This structure makes it possible to query what attributes <patient> has. Because the definition of <attribute> guarantees that if an object is "faulty" it is not "working", and if it is "red", it is not "green" etc. it is possible to give correct *no* replies to questions like: "Is the multiplier faulty?" even if there is no complete list of all possible values of an attribute in the system.

Example:

```
m2( patient     gate-1                                    (5.1.3.3.8)
    modality    logical
    attr        m1( atrb        faulty
                    atrb-cls    state))
```

The above structure associates the object gate-1 with an absolute attribute value *faulty*, belonging to the attribute class *state*. A second example will show a relative attribute with two argument positions.

Example:

```
m5( patient     gate-1                                    (5.1.3.3.9)
    modality    logical
    attr        m4( rel-atrb-cls    rotation
                    atrb1           left
                    atrb2           90))
```

The attribute class here is rotation, while the two arguments describe the sense of rotation and the amount of rotation. It is assumed that gate-1 belongs to a class of gates which is bound to a form. This form has a natural orientation, because our form representation is not rotation invariant. The attribute of the class member gate-1 expresses its rotation relative to this class associated orientation.

The use of attributes with two or three argument-positions amounts to a compromise between structure complexity and expressive power. The attribute class of an attribute *describes* the attribute value. If there are two attribute values, then the attribute class gives a "global" description of both of them together, but it does not say anything about each one of them separately. If somebody wants to capture such distinctions, attribute sub-classes become necessary. The attribute-class rotation is then characterized by a sub-class "direction" and a sub-class "amount".

It is clear that this restructuring adds considerable knowledge to the attribute representation. The system gains the knowledge that "left" is a "direction", etc. However, we consider the previous representation sufficient for all practical purposes. For completeness we will show an example of the use of a three-valued attribute and a zero-valued absolute attribute.

Example:

```
m7( patient     adder-6                                    (5.1.3.3.10)
    modality    logical
    attr        m6( atrb-cls    addtext
                    atrb1       "a text"
                    atrb2       bold
                    atrb3       12))
```

The object adder-6 receives the text "a text" with a font size 12 and a bold face as attribute. The structure used makes it clear that one cannot create attributes with arbitrarily many argument positions, or with a varying number of argument positions. The first problem is due to the fact that arcs have to be defined in SNePS before they can be used, and there is always a fixed number of them. The second problem is due to the association of attribute classes with modifier functions which will be discussed in the next section.

Example:

```
m2( patient     icon-1                                     (5.1.3.3.11)
    modality    function
    attr        m1(     atrb-cls faulty))
```

This is an alternative representation of the example (5.1.3.3.8), assuming that one wants to conceptualize faultiness as a zero-value absolute attribute.

5.1.3.3.3. Attribute Mappings

One of the attractive features of GDK based systems is that mappings between different representational formats can be done declaratively and therefore changed easily, but still show a procedural effect. This has a number of practical applications, besides being sometimes a bare necessity to deal with the reductionist problem. Color is usually a strong medium of communication and can be used to symbolically represent other attributes which are of a non-graphical nature. However if a specific user happens to be color-blind, then he will loose important features of the system. In a knowledge based representation system one can specify the desired mapping of object attributes to picture attributes explicitly. Because this information is accessible to the abstract graphics machine, it can be used to change the display format.

We first need to introduce a structured individual that will be used in attribute mappings.

Syntax:

<value-mapping> : (5.1.3.3.12)

expressed	<attribute-value>
expressed-by	<argument>

Semantics:

(5.1.3.3.12) expresses a structured individual describing a mapping between one <attribute-value> and a corresponding argument to a modifier function <argument>. The <attribute-value> usually describes an invisible attribute value, while the <argument> is its corresponding symbolic attribute value. <argument> must be represented by a base-node.

The following structure completely describes how to represent the necessary mapping.

Syntax:

<attribute-mapping> : (5.1.3.3.13)

attr	<attribute-class>
mod-func	<modifier-function>
val1	<value-mapping>
val1	<value-mapping-2>
:	
:	
modality	<modality>

Semantics:

The structure (5.1.3.3.13) expresses the proposition that attributes of the class <attribute-class> can be expressed graphically by the functional <modifier-function>. In addition it expresses the correct mapping between argument positions of the attribute case frame and arguments of the <modifier-function>. Each syntactic variable named <value-mapping>, possibly with a number after the word "mapping", expresses one mapping between an argument position and a corresponding function argument. The "val1" arc makes it possible to differentiate between attributes with one or more argument positions. If an attribute happens to have two argument positions then the <value-mapping> case frames are qualified by "val1" and "val2" arcs. The "val2" arc marks mappings for the second additional argument. For attributes with three values a "val3" arc is necessary.

This structure incorporates the knowledge necessary to answer questions like "How would you display something <attribute-value>?". The procedural effect of a mapped attribute comes about the following way. If the user's request is to display an object which has the attribute <attribute-value> belonging to the attribute class <attribute-class> and if <attribute-class> is linked to the LISP function <modifier-function> by a structure like (5.1.3.3.13) and if the <attribute-value> is linked to <first-argument>, then call the function <modifier-function> with the form of the object as first argument and with <first-argument> as second (first additional) argument.

So the <modifier-function> takes a <form> as main argument, and it returns a form-function that has been changed such that if it is executed it will incorporate the attribute

expressed by <attribute-value>. Now the position is retrieved as usual and plugged into the modified form function.

Example:

```
m14( attr       newstate                                    (5.1.3.3.14)
     modality   logical
     mod-func   tint-jg
     val1       m12( expressed      faulty
                     expressed-by   green)
     val1       m13( expressed      working
                     expressed-by   magenta))
```

Above example expresses that faultiness of the attribute-class "newstate" is expressed by the "tint-jg" function with the argument "green" while a "newstate" of "working" is expressed by the color "magenta".

For certain attributes numerical values are used, and it would be difficult to impossible (and often unnecessary), to create a table of corresponding attribute values for object and picture. This problem can be solved by a simple convention. If any attribute-value is not covered by an expressed/expressed-by case frame the attribute-value itself is meant to be used as an argument to the modifier function. Alternatively, the same strategy has to be used, if the mapping involves an elaborate computation, not to be expressed in the knowledge base. The mapping case frame for this situation degenerates to the following format:

```
attr       <attribute-class>                                (5.1.3.3.15)
mod-func   <mod-function>
modality   <modality>
```

The description of the procedural effect of this structure can be expressed in the following way. In order to draw an object with an attribute <attribute-class>, send the <form> of the object as first argument to the function <mod-func>. It is assumed that the <form> has been inherited along the class hierarchy. Send the attribute values unchanged as arguments to <mod-func>, such that the value at the first argument position becomes the second argument, the value at the

second argument position becomes the third argument and the value at the third argument position becomes the fourth argument (if they exist).

By now the reader should be sufficiently familiar with our approach to knowledge representation to realize that this convention is a short-cut in the interest of efficiency. No knowledge *about* this convention is stored in the system, so the system could *not* answer questions about it. The convention exists only in the eye of the beholder and in the abstract graphics machine.

The use of the modifier function requires some more clarification. In the theory developed here, an attribute-class is considered as an abstract functional which takes the object it is applied to as argument. If there are any attribute-values asserted, then these correspond to additional arguments to be supplied to the functional. The node at the end of the "mod-func" arc is at the same time the concept that represents the abstract functional and a reference to the code that performs the operations of the concrete functional. This is done in complete analogy to form concepts which simultaneously refer to procedures embodying graphics code.

As trivial as this idea might seem, it creates the only possible attribute theory that is true to the following two principles:

(1) A form which is modified by an attribute is itself a form and should therefore be represented consistently with all other form representations.

(2) All attributes in the system should be treated consistently. The other way one could incorporate attributes in a form function is to make the attribute-values arguments to the form functions themselves. However, this would require the user to predict ALL attributes he would ever want to use, or to recode all forms for every new attribute, or to use the method of modifier functions for all the attributes he did not think of in the first place. This would create form functions with *many* arguments in the first place and would still result in ad hoc extensions for newly discovered attributes. Obviously none of these alternatives is very satisfying.

To change the representation of a certain attribute by color to a representation by different line styles, a new attribute mapping becomes necessary. To keep track of the currently valid mapping a CURRENT-MAPPING pointer will be used. This is in complete analogy to the CURRENT-REFERENCE-FRAME pointer used earlier (section 5.1.3.2.6).

An attribute that expresses a form modification of an object which has its form assigned as an individual is not supposed to be used as a mapped attribute. The reason for that is as follows. One can say that somebody is a "tall man", thereby modifying the size expectation for a typical man. However one cannot say that "Joe is tall", expecting this to modify our image of Joe. Either Joe is tall, and then the attribute incorporates useful redundant information, which however should not be used to influence our image of Joe, or Joe is small, in which case the attribute contradicts our knowledge of his form. This limitation does not apply if one wants to model changes over time, something which is not our concern.

5.1.3.3.4. Unmapped Attributes

If an attribute does not have an associated mapping function then it is not used to modify the pictorial representation of an object. This is referred to as an unmapped attribute. We will present a few interesting unmapped attributes in this section.

As noted before, the fact that the system has complete procedural knowledge of its form primitives does not make declarative knowledge about these forms unnecessary. So one might want to assert that an <object> or a <form> is round.

```
m1( patient     circle-1                                    (5.1.3.3.16)
    attr        m2( atrb-cls    form
                    atrb        round)
    modality    physical)
```

Similarly, symmetry is an important attribute that has initially (section 5.1.3.1) helped to motivate the representation of "redundant" information in the system. Like the "form" attribute symmetry can be asserted about objects as well as forms.

```
m1( patient     circle-1                                        (5.1.3.3.17)
    attr        m2( atrb-cls    symmetrical
                    atrb        x-axis)
    modality    physical)
```

The atrb arc points to one of the three base concepts "x-axis", "y-axis" or "center". The representation of other symmetries is not intended. This exclusion of general symmetry is in line with cognitive data that shows the privileged position of vertical symmetry which is easiest detected by the human visual system [EaL86].

The *area* of a <form> or an <object> will also be considered an attribute, however we will treat only concrete areas as unmapped attributes, while fuzzy area representations will be mapped attributes. The attribute value of a concrete area representation may be any value from the earlier introduced class <area-measure> (structure 5.1.3.2.9), i. e. it will be a structured value.

In the chapter on contributions of philosophy to NLG we have presented a feature analysis of symbolic graphical representations. We will now demonstrate with an example the representation of feature types as attributes.

Example:

```
m1( patient     line-intersection                               (5.1.3.3.18)
    attr        m2( atrb-cls    feature-type
                    atrb        accidental)
    modality    logical)
```

The "atrb" arc may point to one of the base nodes {semantic, accidental, conventional}. The modality arc may point to any of the representation systems that have been discussed in chapters 3.3.1-3.3.11. Finally, the patient arc may point to any feature from the list of common features of symbolic representations presented in chapter 3.3.

Although attributes of features are not supplied with mapping functions we will indicate how a structure like example (5.1.3.3.18) might be used by a graphical generator function. In drawing a wire plan, it is considered good style not to have any unnecessary intersections. We

have mentioned psychophysical evidence [Wal87] that there are good reasons for this "stylistic" decision. If a graphical generator function would create layouts on its own (an idea to which we will return), it should prefer a solution with fewer intersections over a solution with more intersections. This "advice" is embodied in above case frame, if one assumes that the display function tries to minimize the number of accidental features it is producing.

The actual use of above vague description in a display algorithm would imply an abstract graphics machine that is able to create alternative designs, do a "readback" of their graphical features and evaluate them comparatively. This requires at least an additional vision component. The field of AI is just about approaching problem settings of this complexity, but has not reached them yet. Nevertheless, it has hopefully become clear that the previously presented feature analysis is a valuable tool for future research in AI.

The last attribute to be mentioned is "importance". We will make use of this attribute in the section on superlatives below. Importance is clearly a relative attribute. It will be expressed most likely about a component in a structured object.

Example:

```
m1( patient     pcm-chip                                        (5.1.3.3.19)
    attr        m2( rel-atrb-cls   importance
                    atrb           important
                    ref-set        board-1)
    modality    logical)
```

This expresses the fact that the "pcm-chip" is an important component of the structured object "board-1".

5.1.3.3.5. Inheritability of Attributes

Some observations on inheritance and inheritability will be offered in the sections on hierarchies (5.1.3.4, 5.1.3.5). In this section it will simply be pointed out that attributes might or might not be inheritable, and that this item of information is dependent on the attribute itself and can be communicated in a simple sentence. Therefore the linearity principle requires that it should be

representable in the system in a simple declarative structure.

Syntax:

inheritable <attribute-class> (5.1.3.3.20)

Semantics:

Above structure expresses the assertion that <attribute-class> is an inheritable attribute.

It will be explained later that if an object has attributes such that their <attribute-class>es are *not* marked by the inheritable case frame then they are *not* considered for being propagated down a (part) hierarchy. If an attribute class is marked by structure (5.1.3.3.20) then all attributes will be applied not only to objects for which they are asserted, but also to all their parts.

Example:

m1(inheritable size) (5.1.3.3.21)

If a user asserts about an object with parts that it is of a large size, then he will expect all parts to be scaled correspondingly. We will discuss this important observation in detail in the section on parts (5.1.3.4).

5.1.3.3.6. The Representation of Superlatives and Comparatives

The representation of superlatives falls out naturally from our representation of attributes. The attribute value has to be one of the two atomic concepts {minimum, maximum}. The <reference-set> will usually have to be specified. The representation of comparatives is a variation of the representation of superlatives however it is syntactically *not* an attribute representation. This is the case, because we conceive of attributes as referring to one single object, while a comparative connects two object. Therefore the <reference-set> has to be replaced by a <reference-patient>, and the equal importance of <patient> and <reference-patient> is

expressed by integrating them at the same structural level. Finally the attribute values used will be different, and we express this by using a different case frame and a different syntactic variable.

Syntax:

<comparative> : (5.1.3.3.22)

patient	<patient>
rel-atrb-cls	<attribute-class>
comp-atrb	<comparative-attribute-value>
ref-patient	<reference-patient>
modality	<modality>

Semantics:

The <comparative> case frame expresses the fact that <patient> has the comparative attribute value <comparative-attribute-value> belonging to the attribute class <attribute-class> with reference to a <reference-patient>. <reference-patient> must be replaced by a member of the same epistemic class as <patient>, i. e. both must be <object>s or <form>s or <class>es or <feature>s. <comparative-attribute-value> describes attribute values like "larger", "hotter" etc.

The representation of the attribute class "importance" together with the comparative and superlative representation can be used by the abstract graphics machine to decide about the order of drawing components of an object. This "orderly" drawing was something that was requested by one of Grice's pragmatic maxims.

5.1.3.4. Part Hierarchies

Part hierarchies have been of fundamental importance in a number of different areas of artificial intelligence. Knowledge representation [PaS81] has dealt with them as well as hardware modeling in maintenance [Tai87] and research in computer vision [Bie87, Ley86].

Our interest in part hierarchies is of course oriented towards the needs of NLG and knowledge based graphics. We want to know how part hierarchies can help to decide *what content* to put on the screen of an NLG system and *how to organize it,* to be optimally useful to a viewer.

In KBUIMS (knowledge based user interface management system) design this complex of problems has been referred to as "presentation planning" [AMS88].

In the chapter on pragmatics of graphical representations (3.1) it has been pointed out that part hierarchies permit the pragmatic control of graphical displays, according to the Gricean maxim of quantity [Gri75].

Part hierarchies permit a first strategy to decide what to show and towards avoiding information overload: don't show all the parts of a requested object. If a simple display is expected, just show an integral object. If a more informative display is expected, then show the integral object with its first level parts. Only if a complete display is expected, then show the integral object with all levels of its parts. In other words, decide what to display and control the complexity of a display by selecting the number of levels of the part hierarchy that are shown on the screen. This assumption has lead us to distinguish three different types of part hierarchies [GeS87] which we will discuss in detail in this report.

In our view of parts, we distinguish between "real" part hierarchies as opposed to assemblies and clusters. An analysis of different part whole relations based on sentence patterns has been described in [WCH87]. Differences between these two analyses will be pointed out (Section 5.1.3.4.6).

5.1.3.4.1. The Definition of Parts

Before dealing with the different types of part hierarchies it is helpful to define what a part is. The components of a circuit board are definitely NOT typical parts. They have the pleasant property of a history where the parts (the components) have existed separately and were put together by a planned and complicated process.

No definition can be based on this property because it is the exception rather than the rule. A child that gets born has arms and legs, and most people would agree that these are parts of the body, but at no time was a formal assembly done. Another example would be the cutting of a

screw from a cylinder of steel. After the end of the process the head of the screw will undoubtedly be recognized as a part, but at no time was there any step of assembly involved. Switching to other domains, parts become even harder to characterize. A country can have different parts, but no clear boarder line can be drawn between them in many cases. The same thing applies to bodies of water like lakes and oceans.

In some areas of language, parts are not spatial subunits at all, as is the case with the parts of an hour. (The use of the word "part" might be due to the visualization of an hour as the full disc of a clock). Another interesting example is a book. Books have two different part hierarchies which more often than not do not coincide. Pages are parts of a book, and so are chapters or paragraphs. But often a paragraph stretches over page boundaries.

With all these different types of parts it becomes difficult to find a necessary and sufficient condition for the part relation. Winston et al. [WCH87]. characterize being a part by "the elements of inclusion and connection" (p. 438). We want to add a criterion to this characterization, namely we think of parts as smaller than their corresponding wholes.

For abstract objects it is necessary to talk in terms of a mapping function that transforms them such that a spatial measuring function or a counting function can be applied to the image. Such a mapping function always selects a salient feature that is common to whole and parts. We will formulate this in the following necessary condition for being a part.

If an object P is part of another object W, then a mapping m and a function f exist such that m and f fulfill the following four conditions. (1) m is a mapping function that can be applied to both P and W and which transforms the same salient feature of both of them into a spatial or into a countable representation. (2) f is a measuring function which assigns a measure of size or a count to the mapping of W as well as to the mapping of P. (3) The size assigned by f to the mapping of W must be larger than the size assigned by f to the mapping of P.

$$f(\, m(\, P)) < f(\, m(\, W)) \qquad (5.1.3.4.1)$$

(4) The sum of the measures or counts of the mappings of all *immediate* parts of an integral object is smaller or equal to the corresponding measure or count of the mapping of this object. If there are k immediate parts this can be formulated as follows.

$$\sum_{i=1}^{i=k} f(\, m(\, P_i)) \leq f(\, m(\, W)) \qquad (5.1.3.4.2)$$

The reason for the use of the sign $\leq$ will be explained with an example. If we consider a forest and apply a count function f, then the sum of trees will be exactly equal to the number characterizing the forest. If we use an area function instead, the sum of the areas of the trees will be smaller than the area taken by the whole forest.

5.1.3.4.2. A General Purpose Part Representation

We will now introduce an overarching representational structure that covers the three different types of part hierarchies that we have found necessary to distinguish.

Syntax:

object	<object>	(5.1.3.4.3)
modality	<modality>	
part-relation	<real-assem-clu>	
sub-object	<object-2>	

Semantics:

Above structure asserts that <object-2> stands in the part-relation to <object> which is specified by <real-assem-clu> and is valid in the modality <modality>. <real-assem-clu> may be one of the base nodes {real-part, sub-assembly, or sub-cluster}.

5.1.3.4.3. Real Parts

In the realm of graphical deep knowledge an object P is defined to be a "real" part of another object W if the following conditions are fulfilled:

(1) The object P is considered part of the object W in the real world.

(2) Both the object W as well as the object P have a form, i.e. they are both by themselves displayable and can be displayed simultaneously.

(3) The display of W without the display of P creates a "useful" image.

The last criterion is of course up to the eye of the beholder and cannot be formalized any further. The purpose of this definition is obvious: for real parts the complexity of a display can be limited by showing only the integral object without its parts, and without in this way creating an "amputee". For this purpose we assume that the form of W shows some sort of sketch that is suggestive of the object and all its parts without showing the parts in too many details. For a structural circuit board display this sketch will be identical to the outline of the dominating part, the printed wiring board.

We will now present the correct instantiation of the general purpose part case frame.

object	<object>	(5.1.3.4.4)
modality	<modality>	
part-relation	real-part	
sub-object	<object-2>	

Structure (5.1.3.4.4) expresses the fact that <object> has <object-2> as its real part, under the modality <modality>. Objects with many parts will be represented by many assertions of the same type.

If requested to display the object <object> at a low level of graphical complexity, then this is interpreted as a requirement to display <object> alone. If required to display <object> with complete information, then <object> and also <object-2> are displayed, even if <object-2> *was not explicitly requested.*

Example:

m1(	object	board-1	(5.1.3.4.5)
	modality	function	
	part-relation	real-part	

```
        sub-object          multiplier-1)

m2(     object              board-1                          (5.1.3.4.6)
        modality            function
        part-relation       real-part
        sub-object          adder-1)
```

Above structures represent a board called board-1 which has two parts, an adder called adder-1 and a multiplier called multiplier-1.

5.1.3.4.4. Assemblies

In the realm of GDK an object P is defined to be a "sub-assembly" of another object W if the following conditions are fulfilled:

(1) The object P is considered part of the object W in the real world.

(2) Both the object W as well as the object P have a form, i.e. they are both by themselves displayable and can be displayed simultaneously.

(3) The display of W without the display of P creates an image never desirable for the user.

An object, like W in the above definition, that has sub-assemblies is called an "assembly". An assembly may also have real parts. An example from the domain of circuit board maintenance will clarify the changed condition. In a purely functional representation the pins of an integrated circuit will not be displayed at all, they will just be implied by the wire that connects the pin to another circuit. In maintenance such a display is not sufficient, because the pin as well as the connecting wire can be defective. Therefore one would want to display both of them separately, and this is usually done by a little rectangle, representing the pin and called the *port* of the component. In fact it has even been argued [Tai87] that the contact point itself should be seen as an abstract object.

Now it would make no sense to display a component and the wire leading to it, but omit the port that creates the connection between these two objects. In other words, whenever a display of the component is required, one automatically also wants the ports shown. A representation of

ports as real parts of the component is therefore not advisable, because a user could specify a complexity limited display which would show the component but omit its parts. Therefore the ports have to be made sub-assemblies of the component.

The Fig. 5.1.3.4.1 shows a logical representation of a multiplier with its ports. The following structure shows the representation of assemblies which differs only slightly from the representation of parts.

object	<object>	(5.1.3.4.7)
modality	<modality>	
part-relation	sub-assembly	
sub-object	<object-2>	

A proposition node dominating above structure expresses the fact that <object-2> is a sub-assembly of <object> in the modality <modality>. In other words it is a graphically inseparable part.

For display purposes the sub-assembly structure in a sense overrides user requests for low complexity displays. If the display of the object <object> is requested, then display of <object-

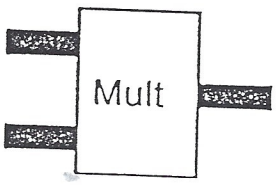


Fig. 5.1.3.4.1: A multiplier and its ports (logical representation).

2> is automatically effected, even if the user has asked for a display of low complexity.

Example:

```
m1( object          gate-1                                   (5.1.3.4.8)
    modality        function
    part-relation   sub-assembly
    sub-object      port-2)

m4( object          gate-1
    modality        function
    part-relation   sub-assembly
    sub-object      port-1)
```

5.1.3.4.5. Clusters

In the realm of GDK an object P is defined to be a "sub-cluster" of another object W if the following conditions are fulfilled:

(1) The object P is considered part of the object W in the real world.

(2) The object P has a form, while the object W usually has no form at all in the real world but has an assigned *symbolic* form.

(3) It is never desirable to display P and W together.

An object, like W in the above definition, that has sub-clusters is called a "cluster". A cluster may not have real parts or sub-assemblies, although some of the sub-clusters may be real parts of another object. Several sub-clusters are said to "form a cluster" if they are all and the only sub-clusters of a cluster. Clusters are standing nearer to what one could call an abstraction hierarchy, than the other two types of part hierarchies. The definition given here corresponds to an improvement of the definition of clusters presented in [GeS87]. In our original definition symbolic forms were always and only boxes, and this fact was not represented in the knowledge base. This is quite satisfying for technical applications, however, if one wants to replace a cluster of trees in a map by a green blob, then this blob will not usually be a rectangle. This improved definition permits one to associate a symbolic form with every super-cluster, as a "real" form is associated with

every real super-part, and store it in the knowledge base.

The case frame used is a specialization of the general purpose part representation.

object	<object>	(5.1.3.4.9)
modality	<modality>	
part-relation	sub-cluster	
sub-object	<object-2>	

A structure as the one above asserts that <object> has <object-2> as a sub-cluster, under the modality <modality>.

If the drawing of <object> is requested in complete detail and <object> is a cluster, then only its sub-clusters will be drawn. If the drawing of <object> is requested at an appropriately reduced complexity level, then such a request is interpreted as a command to display the symbolic form of the cluster itself, but not its sub-clusters.

It is permissible to have an object with real parts, such that some of those parts form one or more clusters. However it is required that all the parts forming a cluster have to be at the same level in the real-part hierarchy, and clusters themselves may not be hierarchical. With this mechanism the user gets one additional level of control over the complexity of the display. He can either request the display of an object, or the display of the object and its parts, or the object and its parts, however, with all the objects forming clusters replaced by these cluster. This is considered a display of intermediate complexity.

A complete example from the domain of circuit boards will now be represented. Fig. 5.1.3.4.2 shows a board with a number of components. It becomes immediately clear that there is more regularity on this board than just a one level part hierarchy. It is quite obvious that a number of components stands in a relation to each other which is replicated three times on the board.

Each one of the three groups of components represents a functional unit dealing with one channel of a signal processing unit. However there is no object "channel processor" with its own form. The group of components that is functioning in a certain way is an object only in an

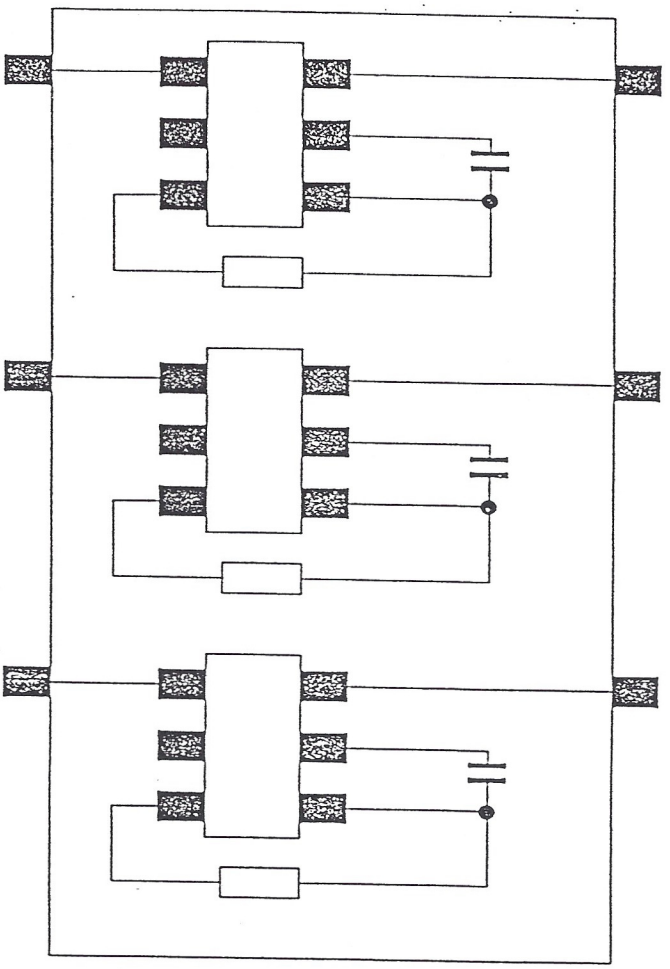


Fig. 5.1.3.4.2: An example board.

abstract sense, and no specific symbol can be introduced for it. But how does this relate to part relations and display complexity?

It has been a long standing tradition in electrical engineering to describe complicated circuits by means of block diagrams which display each functional group of components as one so called black box. This method of display eliminates details and complexity. Sometimes these boxes are drawn around the components in the wire plan, but usually only empty boxes and their interconnections are drawn.

The GDK approach to this display problem is to represent all components as real parts of the whole board, but to summarize corresponding groups of objects as clusters. If a complete

display of a device is required, then the clusters are not displayed at all. However, if a simplified display is requested, then components that are sub-clusters of a cluster are collectively replaced by the representation of the cluster. Fig. 5.1.3.4.3 shows the block diagram for Fig. 5.1.3.4.2. One of the structures that express the sub-cluster relation for the given example might look as follows.

```
m1( object          cluster-1                                   (5.1.3.4.10)
    modality        function
    part-relation   sub-cluster
    sub-object      pcm-chip-1)
```

The display of clusters of objects that consist of icons with connections, like circuit boards or SNePS networks, requires that all connections of cluster elements to non-cluster elements are

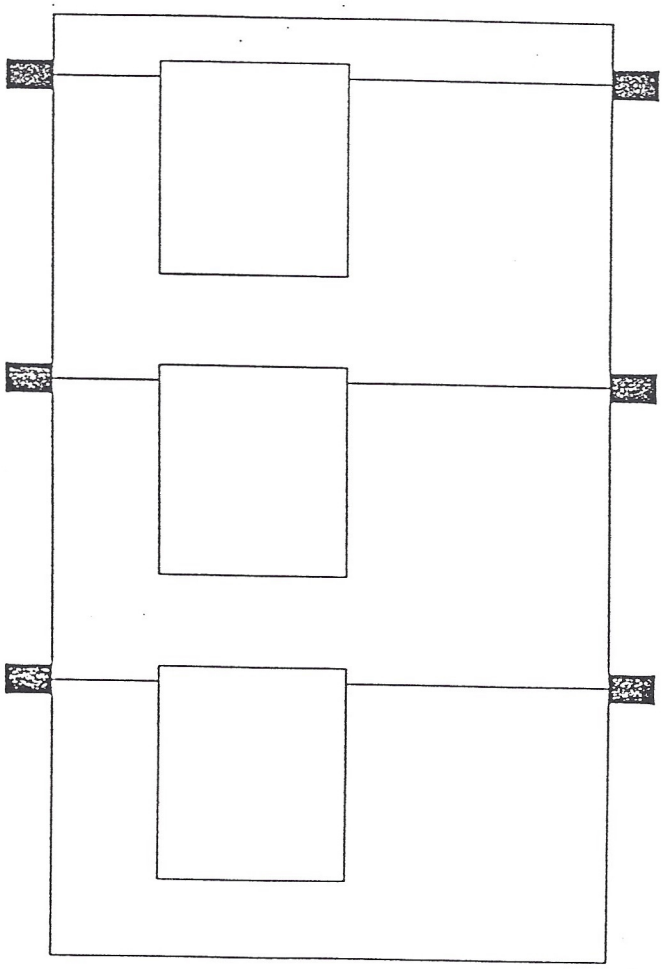


Fig. 5.1.3.4.3: The block diagram for Fig. 5.1.3.4.2

replaced by connections that impinge on the non-cluster element and the symbolic representation of the cluster.

5.1.3.4.6. A Comparison of Taxonomies

In this section we will analyze the relations between our analysis of part hierarchies and the one in [WCH87] which stands in spirit quite near to our ideas. Our distinction between different part hierarchies has been motivated by the necessities of creating a graphical representation of an object with parts. In summary the super object of a real-part relation may be displayed alone or with its parts. The super object of a sub-assembly relation may be displayed only with its parts. Finally the super object of a sub-cluster relation may be displayed only symbolically when its parts are not displayed. The taxonomy of Winston et al. has been motivated by the use of the word "part" in language. They distinguish between (1) component / integral part, (2) member / collection, (3) portion / mass, (4) stuff / object, (5) feature / activity, and (6) place / area. (1) corresponds in most cases to our real parts, and in some cases to sub-assemblies, depending whether one wants to permit the object displayed without details of its parts. (2) corresponds quite well to sub-clusters. (3), (4), and (5) are not covered by our analysis for the following reasons. (3) talks about parts that are created by a separation process, for instance cutting a pie, while we are thinking in terms of an assembly process. The picture of a pie is *not* created by drawing all its slices! (4) deals with sub-parts on the molecular level ("stuff"), which we do not intend to display, unless material boundaries coincide with real sub-part boundaries. (5) deals with a metaphorical temporal use of the term "part" while we are talking about material parts. (6) seems well covered by our notion of sub-assemblies. Winston et al. write "places cannot be separated from the areas of which they are a part" (p. 426).

5.1.3.4.7. Are there more Graphically Interesting Part Relations?

The three described part relations for graphical deep knowledge leave one with the question whether there might be other part relations of interest. It turns out that one can organize these three relations in a table and then systematically extend the table to check for other candidates. The first line in the table should be read as follows. A real part is characterized by the fact that the part may be drawn by itself (a Y in the P column), the whole may be drawn by itself (a Y in the W column), and both may be drawn together (a Y in the T column).

	W(hole)	P(art)	T(ogether)
Real Part	Y	Y	Y
Assembly	N	Y	Y
Cluster	Y	Y	N
4	N	N	N
5	N	N	Y
6	N	Y	N
7	Y	N	N
8	Y	N	Y

Line 4 is obviously a useless combination. Nothing gets ever displayed. Line 6 is uninteresting because the whole is never displayed, and line 7 is uninteresting because the parts are never displayed. This leaves us with lines 5 and 8 as serious candidates. Line 5 implies that if one object, be it whole or part, is ever displayed the other one is also displayed. This expresses a symmetrical relation, while part relations are hierarchical relations.

Line 8 becomes interesting if one wants to refer to holes (sic!) as parts. It is obviously difficult to draw a hole without the surrounding object. Therefore it is permissible to draw the outline of an object and the object with its hole, but not the hole by itself. Similarly, the peak of a mountain cannot be drawn without a mountain. One reason why we did not include this type of part hierarchy in our analysis was the awkwardness of expressing this relation in natural language. Consider for example a sentence like "The piece of Swiss cheese has eight holes as its parts". Nevertheless, in areas like solid modeling [Req80] holes *are* considered as parts, and if we would shift our focus from NLG to solid modeling representations we would need to deal with line 8 as a

part relation.

5.1.3.4.8. Modalities and Parts

It has been pointed out before that modalities are used universally to discriminate between different modes of representation, most notably logical and physical representations. One might doubt that this is really necessary for part hierarchies, but there are at least two interesting cases where differences between logical and physical displays occur. The first case has been alluded to before: integrated circuits very often contain a number of components, in a single (physical) box of a device. In the logical representation all the components are separate objects and might even be parts of different functional units.

The other case occurs when having to refer to drawn parts of a wire. Imagine a functional circuit board diagram, as shown in Fig. 5.1.3.4.4. A user talking with a program about this circuit board could refer to the "upper part" of wire-7, because wire-7 in the Fig. is cut by wire-6 into two different parts. However this diagram is a logical diagram. The intersection of the two wires wire-7 and wire-6 is an artifact of the functional representation. In the physical representation no intersection can exist!

In other words the logical representation of the circuit board has a part structure which is not contained in the physical structure and not even necessary in the logical structure. It is an accident of a specific layout of the logical representation and as such created only by the process of routing. If a system wants to understand the above reference to an "upper part", it has to maintain a part representation under two different modalities. In addition it will have to create the correct part assertion at drawing time, by reading back the picture that it drew. We have previously pointed out that readback of diagrams drawn by the system is an exciting perspective, however not within reach of this current investigation.

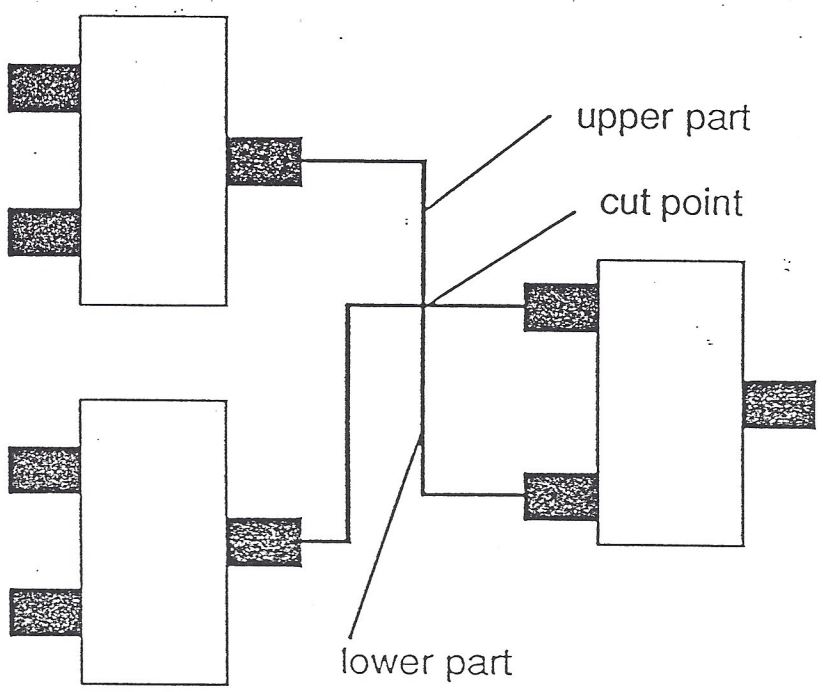


Fig. 5.1.3.4.4: An example board with wire parts.

5.1.3.4.9. The Inheritability of Attributes in a Part Hierarchy

We will now turn to the problem of the inheritability of attributes along a *part* hierarchy. Consider the attribute of "faultiness", which is very popular in the domain of circuit board maintenance. At the beginning of a maintenance session it is known that a whole board is faulty. The purpose of the maintenance session is to narrow down the fault to a single or possibly a few faulty components and acquit all the others. To inherit the attribute of faultiness would completely defeat the purpose of the maintenance system!

However, in creating a picture from predefined components, another view emerges. It is the nature of (our) graphics primitives that they do not represent an object in a size invariant format.

In other words, the form-function of an object always implies a size. To assert in the network that one wants a scaled multiplier, this operation has to be represented as an *attribute* of the picture of the multiplier.

If a whole board is to be scaled, then all of its parts also have to be scaled. In that case it is obvious that one wants to inherit an attribute from an object to its parts. So two attributes have been shown, an inheritable one, and a non-inheritable one. This is the reason why inheritability of an attribute has to be asserted explicitly. Also note again that this is inheritance along a *part* hierarchy, not along a class hierarchy, a technique that has not been used in the KR literature. (We know of one approach that permits inheritance to any "grouping" [Smi83] including parts.) The case frame for asserting inheritability has been presented and completely described in the section on attributes (5.1.3.3).

The necessity of inheritance along part hierarchies is a very important and interesting finding, because it forces us to raise some criticism about the KL-ONE family of knowledge representation [BrS85]. KL-ONE and its descendents comprise the currently most popular family of network based knowledge representation systems in the field. The basic assumption made by these systems is that knowledge representation environments should include a taxonomic reasoner that is operating on a class hierarchy (IS-A hierarchy). The KL-ONE interpreter automatically takes care of inheritance (better: role inheritance) along this class hierarchy. However it does not supply a general purpose inheritance mechanism for other hierarchies, like part or containment hierarchies. As we have shown, it is a reasonable request to ask for inheritance along part hierarchies, and it seems that a knowledge representation system should treat different inheritance hierarchies consistently.

SNePS, a propositional network-based knowledge representation system, does not supply any automatic inheritance, but supplies the ability to write path-based inference rules. An interpreter for such rules is implemented. It is the responsibility of the user to incorporate any required inheritance in his specific network interpreter. This is consistent with SNePS' status as a network

at what Brachman has called the "logical level" [Bra79]. However, knowledge representation systems at the "epistemic level" (like KL-ONE) should give due consideration to uniform treatment of other major ontological hierarchies.

5.1.3.5. The Class Hierarchy

5.1.3.5.1. The Class Inheritance Mechanism

The oldest known principle of knowledge organization, especially for network based systems is the class hierarchy [CoQ69, CoL75, Qui68]. Class hierarchies permit the elimination of redundant storage of attributes with members of a class, because if every member has a certain (the same !) attribute associated with it, it is possible to store the attribute with the class instead of storing it with every single member. This elimination of redundancy has been based on the principle of *cognitive economy*, which, however, has gotten under some attack long time ago [Con72]. Considering the number of active elements in the human brain and the speed of many knowledge based AI programs, one also tends to advise AI researchers to make any effort to waste space and gain time, if human-like intelligence is to be achieved.

Nevertheless class hierarchies have not lost their appeal to AI researchers. By concluding that a certain object belongs to a class, it becomes possible to assert a large number of (default) attributes about this object, without having to derive or perceive them. This ability is advantageous in dealing with situations that require fast decisions (like tigers). If one has to verify that this specific member of the class of tigers is dangerous, it will be too late.

The use of a class hierarchy has mostly been discussed in the sections on forms and positions. It has been shown that objects can inherit their forms and their positions along the class hierarchy. The latter is the case if the position is a relative position and all members of a class are parts at the same relative position relative to their reference objects. For the exact structures used refer to the chapter on positions (5.1.3.2).

The class hierarchy used is NOT a tangled hierarchy and thereby avoids a number of well known problems with multiple inheritance [ReC81]. For instance a conflict would arise if an object could be a member of two different classes each of which might have a different form. The two main case frames for class hierarchies will be repeated here:

Syntax:

<subclass> : (5.1.3.5.1)

sub-class	<class-1>
class	<class>
modality	<modality>

<membership> : (5.1.3.5.2)

object	<object>
type	<class>
modality	<modality>

Semantics:

(5.1.3.5.1) is a case frame, such that any node dominating it asserts that <class-1> is a sub-class of <class>. (5.1.3.5.2) is a case frame, such that any node dominating it asserts that <object> is a member of the class <class>. The arc labels "type" and "class" are used as synonyms.

Reasoning based on the transitivity of the <subclass> case frame permits to deduce that <class-1> is a sub-class of any super-class of <class>, etc. If drawing of an <object> is required and its form and/or relative position are not directly associated with it, above structures can be used to localize a form or relative position.

Example:

m1((5.1.3.5.3)

sub-class	logical-component
class	component
modality	function)

Modalities are of interest in class hierarchies if one wants to talk e. g. about the class of all round components which might have the class of all circular components as sub-class. An object that has a round symbol in a functional display might very well be a square in its structural representation, and therefore be a member of a different class.

5.1.3.5.2. Class Constructs and Categorization Theory

One of the most interesting lines of work in the field of cognitive psychology is the development of categorization theory. The literature reports three different approaches to categorization [SmM81] the classical approach, the probabilistic approach (prototype theory) and the exemplar approach. The classical approach has been but totally rejected from a cognitive point of view. It requires that every member of a class is described by necessary and sufficient conditions.

The prototype view has been developed by Eleanor Rosch [Ros78]. A "prototype" is a summary description of all the members of a class, and members are considered typical if they are similar to the summary description, and untypical if they are different from the summary description. However, even untypical members will be more similar to their prototype than to the prototype of any other category.

An early knowledge representation system that incorporated the notion of prototype has been KRL-0 [BoW77a,BoW77b]. Recently some SNePS modeling of Rosch's work has been reported in the literature [PeS87].

The third theory of categorization is the exemplar view. The exemplar view differs from prototype theory in the following way. Prototype theory describes categories by one summary description which is not necessarily identical to any existing member of the category. Exemplar theory on the other hand postulates the use of one or more stored real exemplars of the category, in other words **no** summary description exists.

The exemplar view of categorization permits an interesting twist to the problem of inheritance. Consider an object with no specified form that belongs to a class hierarchy. Classical inheritance (and our algorithms represented so far) would search up in the hierarchy until at some higher level a form is encountered. However it might happen that no form is found anywhere in the hierarchy. It is our interpretation of exemplar theory that it would be valid to do a *down* search in the hierarchy for an object that belongs to the same class as the current focus object, and to inherit an existing form with *up-and-down inheritance*.

The idea of up-inheritance is not popular in AI. It is either ignored or explicitly prohibited. For instance, knowledge representation of the NETL style [Fah79], which is based on marker passing, prohibits the idea of inheritance according to an up-and-down-movement because if one would permit markers to move up and down in the network the whole network would eventually be marked.

One is tempted to interpret up-and-down inheritance by considering the first step (the up-inheritance) as a form of generalization or inductive learning. However, this is not what we have in mind, because the representation of the class itself is *not* changed by a step of up-and-down inheritance. If a class should have many members only one of which has a form, and if this form should be changed after one application of up-and-down inheritance, then the second application of this inheritance rule will supply the new form, not the old form. Were we talking about a step of generalization, then the class would preserve the form after the first use of up-inheritance.[11]

Are we then making a decision for the universal use of exemplar inheritance and against prototype theory? Clearly this is not our intention, because up-and-down inheritance is *only* used when no sufficient form is associated with the class used for inheritance i. e. when our version of a summary description fails.

In addition we have limited the use of up-and-down inheritance in two other ways. Up-search is done only from the lowest level, the level of the individuals, to the level immediately above it, i. e. to the lowest level of classes. If there is no other member in this class, or if the other members do not carry the desired information, then up-and-down inheritance fails. One can argue that this does not make complete use of the class hierarchy, but it seems like a reasonable compromise, because humans use hierarchies that are flat and bushy. Rosenfeld has even argued that it is not necessary to view operations on hierarchies as recursive to an arbitrary depth, because this constitutes an unnecessary effort if one has only a flat hierarchy. [12]

---

[11] This is not necessarily true if one wants to deal with generalization combined with truth maintenance.

[12] Azriel Rosenfeld, talk 4/28/87 SUNY at Buffalo, on Recognizing Unexpected Objects.

Secondly, up-and-down inheritance is used only for information that is urgently needed, and not as the default case. In a graphics system the one item of information that is obviously needed most is the form of an object, for which no "reasonable defaults" can be supplied. We will now formally define up-and-down inheritance which we also refer to as exemplar inheritance.

Definition 5.1.3.5.1: Exemplar Inheritance.

> If an individual is missing information about an important property, and the class it is immediately a member of is missing the same property, then the property may be inherited from any of the other members of this class.

In our domain only "forms" are considered important, and we have therefore decided not to represent the fact that a property is important by an explicit assertion.

It is not yet clear what happens when several members of a class have different forms. We will argue now that this is unlikely for two different reasons. (1) Form is a salient feature of class members. In fact Rosch [Ros78] has shown that for base level and sub-ordinate[13] categories a pictorial representation is maintained such that there is a strong correlation between prototypicality of a new member and the area overlap between the prototype and the new member. In other words, if two things look very different, it is quite unlikely that they belong to the same category. (2) It is also quite unlikely that a user would tell an NLG system about many different individuals that they have a certain (the same!) form, and then, as an afterthought, tell that all these individuals belong to the same class. Classes are (among other things) used to minimize communication, so such a behavior would completely defeat this purposes.

It *is* likely that a user initially thinks of one object only and assigns it a form. When the second object of the same class occurs in the interaction he will notice that he can save himself a lot of typing if he tells the system of the class. At this point he should not be forced to tell the system that the class has the same form as the one member that he already introduced. But if the

---

[13] "Chair" would be a base level category, "kitchen chair" a sub-ordinate category, and "furniture" a super-ordinate category.

form is actually different, then he will have to tell this fact. So the system is quite safe to rely that it receives complete but not redundant information, i. e. the system relies on pragmatic maxims!

Should the unexpected happen, and different members of the class have different forms, then several strategies are possible. Random selection (which is what is currently implemented), selection of the most common form, or selection of the most recently mentioned form are reasonable choices.

5.1.3.6. Periodicity

Periodic structures built up from simple constituents can be described in simple words and therefore (linearity!) deserve to be represented by a simple knowledge structure. Fig. 5.1.3.6.1 shows a structure that is periodic in one dimension. The case frame for this structure is:

Syntax:

<periodicity> : (5.1.3.6.1)

object	<object>
constituent	<object-form>
inter-pos	<vector>
const-number	<number>
modality	<modality>

Semantics:

Above case frame asserts that <object> consists of <number> occurrences of <object-form> that have relative positions between each other that are defined by the vector <vector> in the modality <modality>. <object-form> must be either an <object> or a <form>.

This structure is similar in flavor to a cluster, except that all sub-clusters have the same form and relative position and no separate concept node for the sub-objects exists. The usefulness of this structure for drawing purposes is obvious. If the display of <object> is requested, then it is necessary to display <number> occurrences of <object-form>. The first <object-form> is

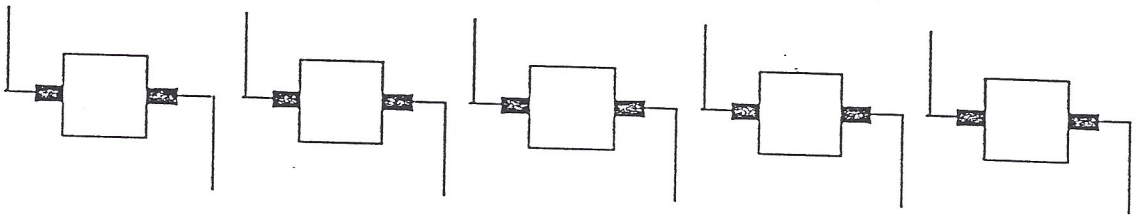


Fig. 5.1.3.6.1: A one dimensional periodic structure.

placed according to the position of <object>. All other <object-form>s are placed by adding the vector <vector> to the position of the <object-form> placed immediately before.

The <periodicity> case frame introduces the problem of uninstantiated individuals [Fah79]. It refers to a number of objects for which no individual concept nodes exist. Should the user want to refer to any single member of a periodical structure, the following new structure has to be generated by the NL parser.

Syntax:

<mentioned-indiv> : (5.1.3.6.2)

object	<object>
one-of	<object-2>

modality <modality>

Semantics:

The object <object> is one of the objects that are described by <object-2> as a summary representation. This relations holds only in the modality <modality>.

As a side note we would like to mention that the problem of uninstantiated individuals occurs in a different guise in the representation of plurals. Upon hearing somebody mention 500 horses, not all of them will be represented individually in the listener.

Example (periodical structure):

```
m2( object          row-of-pcm-channels                                    (5.1.3.6.3)
    constituent     pcm-channel
    inter-pos       m5( coord-sys    world-1
                        component
                        m4(          direction   X
                                     measure     m1( value   1
                                                     unit    inch))
                        component
                        m3(          direction   Y
                                     measure     m2( value   0
                                                     unit    inch)))
    const-number    5
    modality        function)
```

This case frame describes an object called "row-of-pcm-channels" which is made-up of 5 pcm-channel forms whereby the individual channels have reference points in relative positions to each other which are described by the vector (1 0). Two dimensional patterns can be constructed by requiring <object-form> to be instantiated as <object-2> only and by making <object> a constituent of another periodical structure. Fig. 5.1.3.6.2 shows the pattern that is created after adding the following structure:

```
m4( object          board-of-channels                                      (5.1.3.6.4)
    constituent     row-of-pcm-channels
    inter-pos       m5( coord-sys    world-1
                        component
                        m4(          direction   X
```

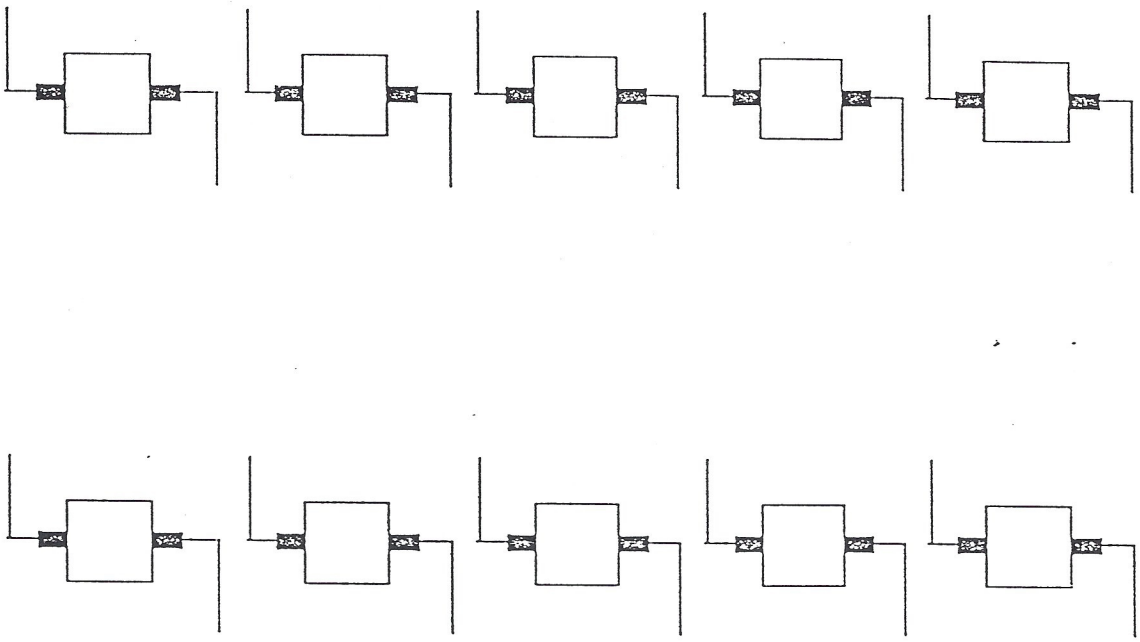


Fig. 5.1.3.6.2: A two dimensional periodic structure

```
                                measure     m1( value   0
                                                unit    inch))
                    component
                    m3(         direction   Y
                                measure     m2( value   2
                                                unit    inch)))
const-number   2
modality       function)
```

5.1.3.7. Angle

Our analysis has assumed the existence of primitive objects which are spatially combined. This does not leave a lot of opportunity for using angles, because angles apply normally to sub-primitive units like lines. For completeness a case frame will be specified that expresses an angle between two linear objects. This case frame specifies the angle between two lines or line segments.

Syntax:

object	<line-1>	(5.1.3.7.1)
angle	<angle-measure>	
ref-obj	<line-2>	
modality	<modality>	

Semantics:

The angle between the objects <line-1> and <line-2> under the modality <modality> is <angle-measure>. <angle-measure> was defined previously (Structure 5.1.3.2.7). <line> describes a <form> or <object> consisting of one line segment only.

5.1.3.8. Pragmatic Hierarchies

We have so far mentioned part hierarchies and class hierarchies which are widely used in AI. Assemblies which are operationally different from parts, but strongly related have been introduced. Later on clusters were presented which relate to part as well as abstraction hierarchies. Some researchers in AI have used topic hierarchies [HaS86] and containment hierarchies [Fah79]. Goal hierarchies are ubiquitous in the planning literature, starting with PAM [ScA77]. Using the representation of comparatives and the attribute of importance a partial order can be introduced for the importance of all elements of a class, or all elements of a cluster. It seems natural to raise the question if there are any relations between these different sorts of hierarchies.

Miller and Johnson-Laird review literature concerning unifying factors between different hierarchies [MiJ76]. It is claimed that part hierarchies (they call the part relation "partonymy") are a more abstract version of containment hierarchies. Miller and Johnson-Laird say that

> "Drawings of the parts of an object will be parts of the drawing of the object, that is they will be located inside it; to the extent that part-whole relations can be represented graphically, the part-whole hierarchy must be isomorphic with a locative-inclusion hierarchy."

Winston et al. [WCH87] refer to parts as meronyms and to the associated wholes as holonyms. Part relations are *meronymic* relations. They interpret part relations as semantic inclusion

relations in a taxonomy that has class relations and spatial relations at the same level.

Class hierarchies have been introduced to AI for reasons of cognitive economy. However, their real benefit seems to be the speed up of communication that they permit. One can say useful things about all members of a class, without having to pay the price of mentioning all of them. If no intensional description of the class exists and one wants to say many things about this class, then one can still save time, even if one complete transfer of an extensional class description is necessary. Similar observations are true for part hierarchies, containment hierarchies, and importance hierarchies. One might want to say something about all the parts of an object, about all the cities in a certain area, or about all the important components on a circuit board. Because of these commonalities we refer to all such hierarchies as *pragmatic hierarchies.*

Although all these hierarchies relate a super-unit to a number of sub-units, it has not been found useful to combine them into a single case frame. Doing that would remove a number of important distinctions. Classes are never drawable, while objects are. If the "class" arc and the "object" arc would both become "super-unit" arcs, then this distinction would only be expressed by the relation itself and would eliminate declarative knowledge from the network and move it into the interpreter. Similarly, a "sub-unit" arc would not express the distinction between part relation and class membership, which would lower its value to a pre-Brachmanic is-a arc. In short, there seems to be no value in expressing this very general commonality of all pragmatic hierarchies in a knowledge base. *Relation element theory* [WCH87] that analyses semantic relations according to their features would permit us to deepen the level of our representation so that commonalities between different pragmatic hierarchies are expressed, something we have not found necessary for GDK.

5.1.4. Knowledge Compilation

Our presentation of KR for NLG will be concluded with a short look at knowledge compilation. Anderson [And83] describes applications of knowledge compilation to production systems.

He distinguishes two different steps of compilation. In the first step of his production rule based system, a number of production rules that are executed in close sequence is combined into a single production rule. This process is called *composition.*

In the second stage this new production is transformed into executable code by a process of proceduralization. Knowledge compilation achieves the speed and reliability of a procedural system while still maintaining the flexibility and generality of a declarative system. Knowledge compilation models the process of extended human training that finally leads to the execution of routines without any thinking.

The theory of graphical deep knowledge as presented in this investigation invites the introduction of a comparable method of knowledge compilation. As has been made clear in previous sections, the display of a complicated object with many attributes and many parts is achieved by a time intensive search which retrieves the specified parts and the specified attributes, does inheritance of attributes to parts (if necessary) and of forms and sometimes positions along class hierarchies, and finally applies the attributes to the resulting forms and displays them one by one.

As opposed to Anderson's system there are no productions used anywhere, nevertheless it is possible to introduce a two step methodology of knowledge compilation. The first step consists of creating assertions about forms and absolute positions of objects on the screen. These objects will have their attributes already "frozen in", and no relative position evaluation, no part expansion, and no inheritance has to be done. Finally all these objects on the screen are made sub-structures of a new object representing the complete screen content. This structure represents a first step of knowledge compilation, because in the case of an identical display request as the one that resulted in creating these structures (and under the additional assumption that no new attributes have been asserted) the pictorial structure of the repeated request will be identical to the pictorial structure of the initial call.

The second step of knowledge compilation, the proceduralization, can also be modeled easily. One should remember that all the form functions are constructed from a small set of primitives

(like lines, arcs, text, etc.). Sequences of these primitives create the structure of all the form functions, for objects as well as for icons. The representation of a screen situation by assertions about the icons of the configuration at hand can be conflated into a single large graphics procedure by chaining the primitives of all icons together.

This operation creates a structure with all the properties of a proceduralization. The simple execution of this procedure will create a display identical to the previous one, but the actual drawing will happen many times faster than the drawing from the declarative representation. On the other hand, if all declarative knowledge is wiped out (by whatever cause), the procedural representation will not maintain anything like object identity of parts, part relations between them, their attributes, or their relative positions. In other words it becomes impossible to query the representation about these items of information. This is a typical characteristic of procedural knowledge. Cognitive agents have problems to explain how they perform activities maintained only procedurally, like riding a bicycle.

5.2. Reasoning

For my thoughts are not your thoughts, neither are your ways my ways, saith the Lord. For as the heavens are higher than the earth, so are my ways higher than your ways, and my thoughts than your thoughts.

Isaiah 55, 8-9

The major reason for introducing the notion of graphical deep knowledge as separate from graphical knowledge has been the interest in doing reasoning about graphical structures. There are two types of reasoning that have been established in the literature. The more common notion of reasoning is based on a declarative representation (knowledge representation), often founded on a logic based theory. The other type is analogical reasoning. We will concentrate on logical reasoning.

5.2.1. Logical Reasoning about GDK Structures

The first step of making a corpus of representations accessible to logic based reasoning is to transform it into a well formed declarative format with a defined syntax and semantics. It has been the approach of this investigation to limit the procedural representations which at some point are not avoidable in graphics to a small area, namely to iconic primitives. All conceptual relations between these iconic primitives are represented declaratively.

The second step is to define rules of reasoning that can operate on the given structures. One of the strong points of SNePS is its ability to represent more sophisticated rules than other AI systems. The following piece of SNePSUL code shows a rule that says that anything that is left of something has this object to its right.

```
(build  avb      ($x $y $z)                                                    (5.2.1)
        thresh   1
        arg      (build object    *x
                        relpos    (build  coord-sys  world-1
                                          component
                                          (build     direction X
                                                     measure (build  value left
```

```
                                                                    unit   left-right)))
                rel-to     *y
                modality   *z)
arg     (build object      *y
                relpos     (build   coord-sys   world-1
                                    component
                                    (build      direction X
                                                measure  (build   value  right
                                                                  unit   left-right)))
                rel-to     *x
                modality   *z))
```

This can be used in a situation where the position of one object A is known, and in addition it is known that A is left of an object B, however nothing (else) is known about the position of B. A person would immediately conclude that B is to the right of A, and a system claiming any form of intelligence should do the same thing. The above rule will enable it to do so.

The reading of the rule is, that for all x,y,z if one of the two following arguments is true, then the other one must be true too, or otherwise both of them are false. The two arguments express leftness of x relative to y and rightness of y relative to x. In other words, this representation is an equivalence between the two arguments, but it is achieved as a specialization of a conceptual generalization of equivalence which holds for any number of arguments [Sha79a].

The three terms $x, $y, and $z introduce three new variable nodes. These are nodes that can match any other node in the network. All occurrences of *x, *y, *z refer back to these same variable nodes. The "avb" arc expresses universal quantification of the variable nodes it is pointing to. This is a predefined system arc. The "thresh" arc points to the number indicating how many of the following arguments have to be true in order to require all of them to be true. The "arg" arcs finally point to the structures that have to be true (or false). Both "thresh" and "arg" are also predefined SNePS arcs.

Unfortunately, node-based reasoning is not sufficient for the arithmetic operations that are used in position computations. Unless one wants to implement all of arithmetic in SNePS, there is no other way to do it than to escape to the implementation language, i.e. LISP. Concerning the slow speed of rule based reasoning, SNePS offers an alternative reasoning mechanism, called "path

based inference", that can be used for many of the reasoning tasks indicated in the previous chapter on GDK structures.

Path based inference in SNePS assumes that one has a node of a well specified category available (typically an "object") and follows the arcs that are pointing to this node backwards until one hits a node describing unknown and interesting information (for instance a "form" or one coordinate of a position). The well specified case frames of GDK assure that if the required information exists at all in the network, then it will be reachable by a well defined path. Paths can be described by a sublanguage of SNePSUL, as explained in [ShS83], but we will limit ourselves to an example.

```
(find                                                          (5.2.2)
        (compose
                form- !
                (domain-restrict
                        (modality function)
                        class)
                (kstar
                        (compose
                                class- !
                                (domain-restrict
                                        (modality function)
                                        sub-class)))
                type- !
                (domain-restrict
                        (modality function)
                        object))
                pcm-1)
```

The "find" function denotes a retrieval operation from a SNePS network. Two arguments are supplied to this operation, a modality (namely "function") which is used at three positions in the find call, and an object, namely "pcm-1". All other symbols in (5.2.2) correspond either to arcs of GDK case frames, or to keywords of the path sublanguage of SNePSUL. The goal of this operation is to retrieve a form for pcm-1. Figure 5.2.1 will be helpful in understanding this example.

The "compose" keyword introduces a path. A path is constructed from its tail to its head, therefore we start with a "form-" arc that would emanate from our requested result, should it exist. (Remember that an arc with a minus after its name is an ascending arc). The exclamation

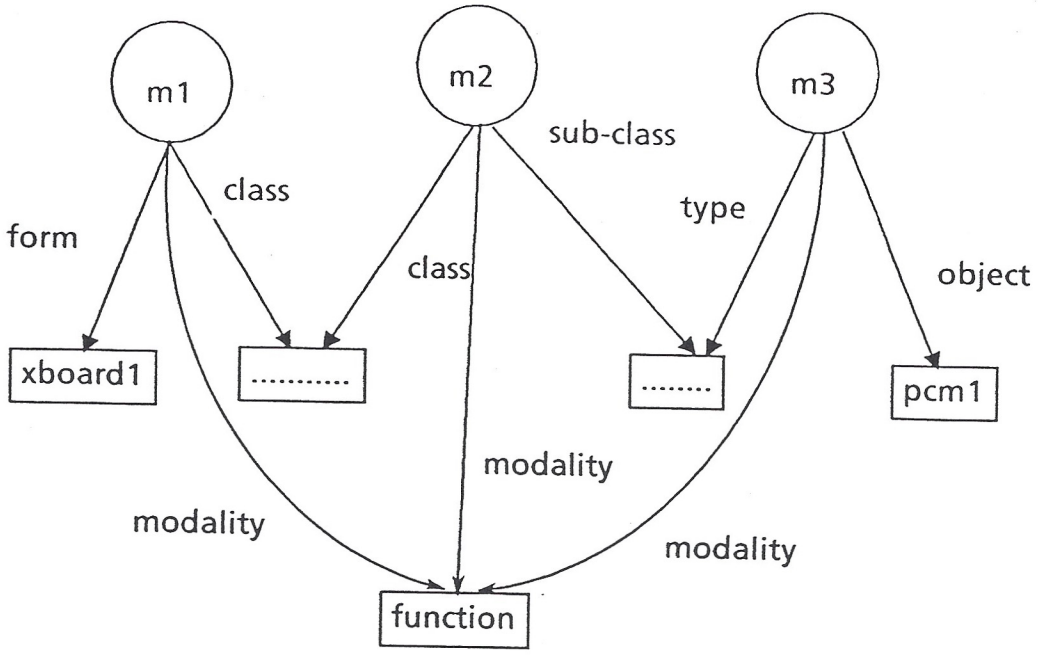


Fig. 5.2.1: An example structure for pathbased inference.

mark indicates that we are looking for an assertion, i. e. it would not be sufficient to find a correct structure if it would only describe a hypothesized proposition. The "(domain-restrict (modality function) ..." piece expresses an additional constraint on this assertion: it must dominate a node "function" by way of a "modality" arc. Finally our path leaves the assertion node along a class arc. This time no negation is specified, because the path actually runs parallel to the arc.

The "kstar" keyword indicates that the structure that is parenthesized together with it can be repeated 0, 1 or arbitrary many times, i.e. it is a Kleene * operator. The potentially omitted or repeated structure consists of a reverse "class" arc "class-", followed by a "sub-class" arc. As before, this piece of path can be passed only if the node dominating it is asserted ("!"), and if it is also dominating a modality arc pointing to a "function" node. Finally the structure must terminate with a "type" arc which is traversed in the reverse direction, and an "object" arc that coincides with the head of the path which is pointing to "pcm-1". Like before, modality and assertional status must be correct, otherwise the path traversal fails.

For use in a graphical generator function the described path is encapsulated in a LISP function, with "pcm-1" and "function" consistently replaced by LISP variable names. Because of the SNePS unquote convention[14] it is necessary to enforce an additional level of LISP evaluation. For the computation of a typical position several such paths will be necessary: one for a relative x coordinate, one for a relative y coordinate, and two for x and y coordinates of a reference object (which could themselves be relative to another object...). The final operation is then arithmetic, as opposed to logical: relative coordinates have to be added.

---

[14]Atoms are not evaluated unless they are specifically unquoted.

CHAPTER 6

IMPLEMENTATION

As is the custom for an AI dissertation, large parts of the theory developed in the previous chapters have been implemented. Natural language parsing is done by using a semantic grammar [BuB79] based on the ATN formalism [Woo70] [Bat78] [Pal85]. The used ATN is part of the standard SNePS distribution [Sha82].

The grammar written for this investigation categorizes input into assertions, questions, and commands. Assertions result in building GDK structures that represent the language utterances. Questions activate retrieval operations and return the results of these operations, sometimes combined with a canned phrase. Whenever possible, questions also activate calls to the display function, such that replies are given graphically as well as with language. Commands always activate calls to the display or the screen erase function.

During natural language input a user might refer to a "form" which is unknown to the system. In this case, the grammar invokes a graphics editor (called Readform) that permits the user to design the referenced form icon. After exiting the editor, the user will find himself again in the language interaction environment. Alternatively the graphics editor may be invoked directly from LISP to create necessary form icons before initiating a natural language dialogue.

Diagram display is performed by a program called "TINA" (which stands for "TINA Is No Acronym"). Inside of LISP a user can invoke TINA as a function and pass the objects he wants to see as arguments. A number of options are provided, for instance a user can select how many levels of the part hierarchy of the object should be displayed.

There exist two versions of the combined parser/TINA program. The version that implements the theory shown in the previous chapters is running at USC/ISI on an HP 320 workstation with color graphics. SNePS as well as TINA are coded in Common LISP. The necessary graphics

primitives are implemented as foreign function calls referring to C procedures which are part of the X window environment. Earlier work which is based on a weaker set of GDK structures, and which was reported in [GeS87] has been implemented in Franz LISP on a VAX 11/780 at SUNY at Buffalo, using a "GIGI" graphics terminal and a set of LISP graphics primitives originally written by Z. Xiang. This earlier version also contains the "Intelligent Machine Drafting module" which will be described later on. In this chapter we will first show a few examples of calls to the "TINA" program, to demonstrate some of the possible options. Then we will show a number of annotated demonstrations of the combined parser/TINA environment. The graphics output will be demonstrated by screen dumps. After that we will briefly discuss "Readform" the graphics editor. In the final section of this chapter we will explain the use of TINA for maintenance purposes.

6.1. The TINA Display Program

The TINA display program expects a calling argument that specifies one or more objects to be displayed, and a knowledge base containing graphical deep knowledge about these objects. The basic calling format is as follows:

(tina [global-options] { [local-options] { object}})

Hereby the following conventions are used: "[]" describes optional elements; "{}" describes one or more repetitions. "object" stands for a node representing an object in the system, i. e. an <object> of our theory.

Global options may be :fill :environ, and :world and only one of these may occur in one command. :fill and :environ specify two modes of display that will be discussed in the examples below. :environ also takes a sub-option called :upto which will also be explained with an example. The ":world" option specifies that all given objects can be expected to have concrete positions asserted in the network. This option permits skipping tests that would lead to fuzzy positioning routines.

Local options have to be given in a specified order and the following are permissible: :window, :modality, :mark, :level. A :modality always has to appear after a :window but before a :mark or :level option. :modality permits to select a modality different from the default modality which is ":function". :window takes four numeric parameters that define a sub-window for display. :mark graphically marks the object following it. Finally, :level tells the system the number of levels in the part hierarchy that should be displayed.

(tina D1M1) (6.1.1)

This is the simplest possible call. An object named D1M1 will be retrieved. TINA will search through the knowledge base to find form information about D1M1 (section 5.1.3.1). After that, attribute information will be retrieved. As has been explained in the chapter on attributes (5.1.3.3), this knowledge will be used as a set of functionals which modify the form of the object. The next step is position retrieval. The chapter on positions (5.1.3.2) contains a large number of possibilities how the position case frame can be instantiated, but in most cases one can assume that a position will be stored as a numerical relative position such that a chain of relative positions can be constructed which finally refers to the screen coordinate system. By applying the modified form function to the x and the y value of this position, D1M1 is made to appear on the screen.

(tina D1M1 D1M2) (6.1.2)

This second example is given only to illustrate the fact that variable numbers of parameters may be used.

(tina :level 2 D1M1) (6.1.3)

Part hierarchies have been explained in detail in the chapter on parts (5.1.3.4). The :level option describes how many levels of objects in the part hierarchy below the given object are requested. In this case it is specified that D1M1 and one level of its parts are requested.

(tina :level 0 D1M1) (6.1.4)

(6.1.4) has been implemented for completeness purposes and describes an empty call, because 0 levels i.e. not even the object itself, are displayed.

(tina :level 1 D1M1 D1M2) (6.1.5)

(tina :level 1 D1M1 D1M2 :level 2 D1A1) (6.1.6)

The above two examples show the range of validity of a local option. It extends to the beginning of the next option. Global options apply to the whole command. In designing this calling syntax, an attempt was made to model UNIX[1] options, rather than having a proliferation of parentheses.

(tina :fill D1M1) (6.1.7)

The :fill option activates an image transformation. It permits the display of an object optimally, by using all the space available. For this purpose a scaling/shifting step is interpolated into the display operation. This step blows up or reduces and relocates a requested object such that it optimally fills a given default window and that no distortion of the given shape is effected. The default window used is the right half of the screen. The :fill option is a global option in that it applies to the whole call. The fill option is a special case of the :window option, which permits to select a sub window of the graphics window at calling time:

(tina :window 100 100 200 200 :level 1 D1M1) (6.1.8)

However there may be several :window options in one call, but only one :fill option.

(tina :mark D1M1) (6.1.9)

The user can specifically request the emphasized display of an object. The :mark option is implemented (depending on the type of graphics hardware available) by turning blinking mode on for

[1] UNIX is (of course) a trade mark of AT&T Bell Labs.

the time the specified object is displayed. If the given graphics hardware does not permit any blinking the mark operation is executed by repeatedly overdrawing of the object while slightly (one pixel) displacing the object in the process of overdrawing. This creates a triple line width for all horizontal and vertical lines and makes the component more conspicuous. The mark implementation is also used in the :environ implementation as specified below.

(tina :environ :level 1 D1M1) (6.1.10)

The environment option has been especially useful for circuit board display. Showing a whole circuit board on a single small screen will not result in too much detail for each object. On the other hand, showing a single component of a circuit board is next to useless, because connectivity conditions have to be preserved. The environment option solves this dilemma by splitting the screen into two parts. The left half of the screen is :filled with the requested object. The right half of the screen is :filled with the integral object of which the requested object is a part.

The user specifies only the component for the left window. The environment is found automatically by doing an up-search along the part hierarchy. Note that the same part hierarchy is now used in the opposite direction, and therefore proves already as more powerful than any procedural object representation.

Also the display does two more things to help the user orient and to eliminate irrelevant information. First of all it :mark's the requested component inside of the reference object. Secondly, when searching the part hierarchy up the program uses a sibling strategy of display which moves one level back down from every superior level found in the search. This means that an object is displayed with its siblings, and so is its immediate super-object, but the siblings of the super object are not farther expanded. This creates a representational structure as shown in Fig. 6.1.1. The :environ mechanism has been developed in the chapter on pragmatic maxims of quantity (3.1).

(tina :environ :upto 2 :level 1 D1M1) (6.1.11)

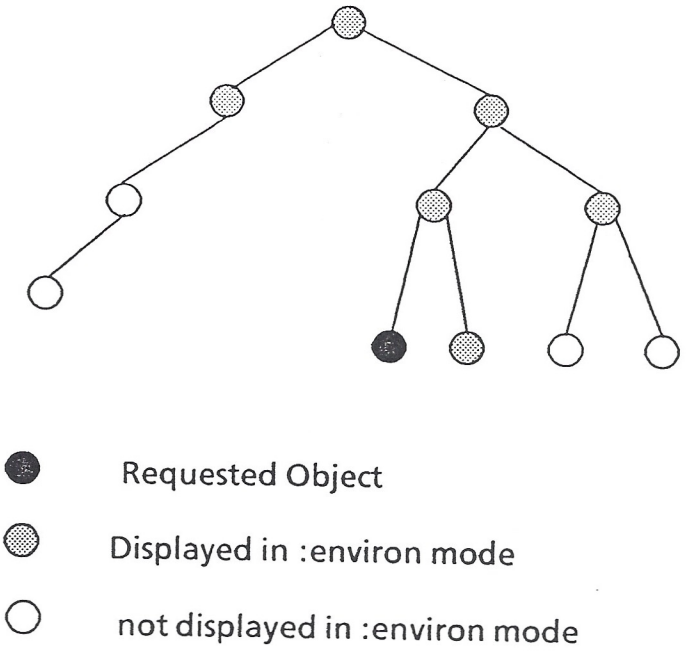


Fig. 6.1.1: The environment display.

It has not been made clear so far, but the upsearch through the part hierarchy is unlimited. If the user knows that there are too many levels of hierarchy between the object requested and the main object of the system, he can limit the upsearch length. This is done by the sub-option :upto to the :environ option.

(tina :environ :upto 0 :level 1 D1M1) (6.1.12)

This is a special case that suppresses the display of the environment altogether. It is not usefully different from the :fill parameter, but it is useful if display is itself called from another program with a LISP variable as parameter, and this variable is possibly set to zero. :environ is a global option, however it permits only one focus object, so the distinction is of little interest.

(tina (^ (car object-list))) (6.1.13)

(6.1.13) shows the interface that SNePS supplies to get at the LISP evaluator. Objects displayed so far were concepts of the SNePS representational system. Here a node is accessed as a LISP atom. (Of course LISP atoms are the implementational basis for SNePS nodes too).

(tina (find object- m50)) (6.1.14)

This shows another version of SNePS access, TINA is called on the result of a knowledge base search. "find" is the SNePS function that retrieves a node according to a pattern, not the Common LISP "find" function.

6.2. Example Runs

In this section we will present a number of annotated demonstrations of the combined parser/TINA program. We will show the interaction in the text section and interpolate figures of screen dumps wherever necessary. User input is always preceded by a prompt consisting of a colon and two greater signs (: >>). Annotations are given in square brackets.

Demonstration 1

[*This demonstration introduces a few of the basic capabilities of the system. It solves a challenge problem that was presented by Norm Sondheimer at IJCAI 1987, at the Panel on the use of "Pointing" in user interfaces. The problem solved is, that we have a system with access to a database of ships and information about their state. All ships that are in a non-operational state should be displayed upside down. This change in presentation policy is to be achieved by a natural form of interaction, preferably by "telling the system".*]

: >>screen-coord-sys is a screen coordinate system with s-x s-y in screen

[*The program assumes the existence of a coordinate system called screen-coord-sys. However, the user is free to assign names to the coordinate axes.*]

PARSED

: >>The assumed modality is function

[*Instead of forcing a modality into all sentences we declare one in the beginning.*]

PARSED

: >>one inch corresponds to 73 pixels

[*Different display devices differ in their resolution, and we can inform the system about the current resolution. Isotropic hardware is assumed, however.*]

PARSED

: >>the view of screen-coord-sys is front

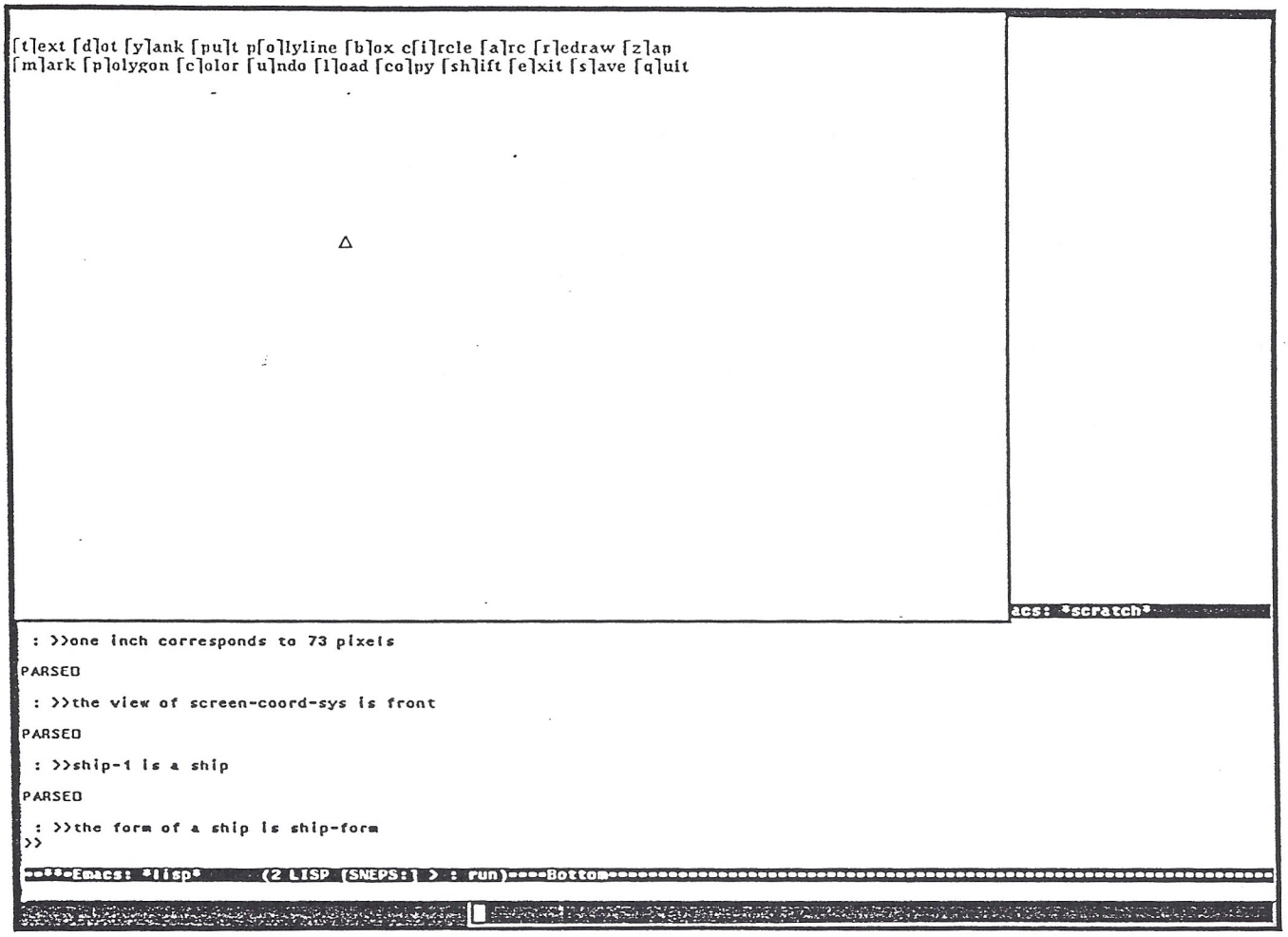


Fig. 6.2.1

[*This is necessary in case a term like "behind" is used, even if a plane coordinate system has been defined. Above four sentences define a sort of preamble that occurs in most demos and that will not be repeated any more.*]

PARSED

: >>ship-1 is a ship

[*This creates an assertion that "ship-1" is a member of a class called "ship". The ATN does not use a lexicon, and both words ship-1 and ship are completely unknown to the system.*]

PARSED

: >>the form of a ship is ship-form

[*This sentence asserts that the form of every member of the class "ship" is "ship-form". The class is known from the previous interactions, but "ship-form" is again new to the system. Because this form is not yet known, the user is put into the "Readform" editor that permits the creation of a form. This editor is menu based and uses mostly single letter commands. The first screen dump (Fig. 6.2.1) was done after entering the main menu of Readform. The upper part of the screen is the graphics window, the lower part is the interaction window which runs under GNU Emacs. The next three screen dumps (Fig. 6.2.2 to Fig. 6.2.4) show intermediate states in icon creation. After exiting Readform, the user returns automatically to natural language interaction.*]

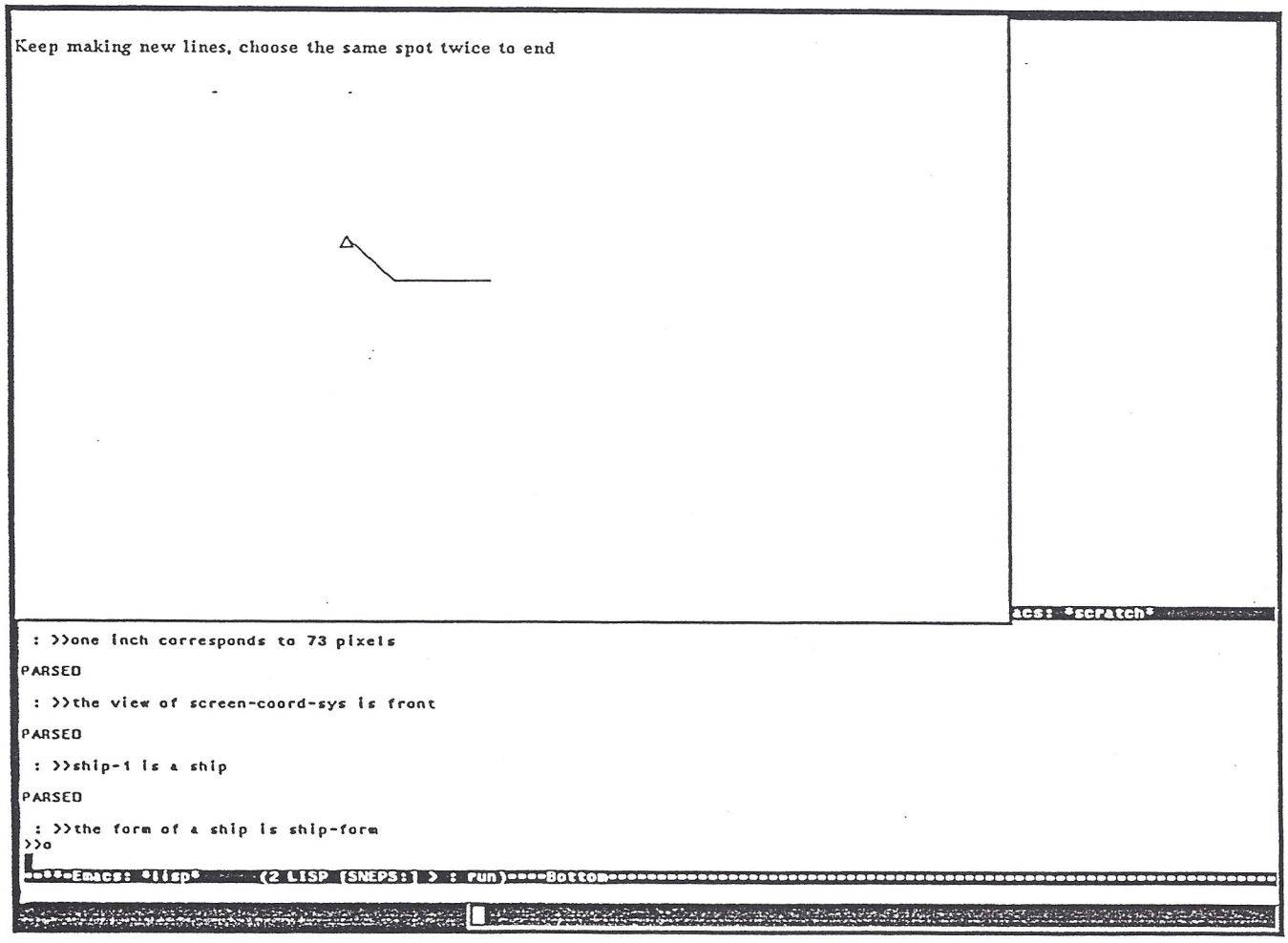


Fig. 6.2.2

PARSED

: >>ship-1 is at 200 300 on the screen

[*This sentence asserts a position for ship-1.*]

PARSED

: >>clear the screen

[*Now the graphics window is erased.*]

DONE

: >>please show ship-1

[*Ship-1 is displayed at its correct position, not the one where the icon was created. This is the situation in Fig. 6.2.5*]

DONE

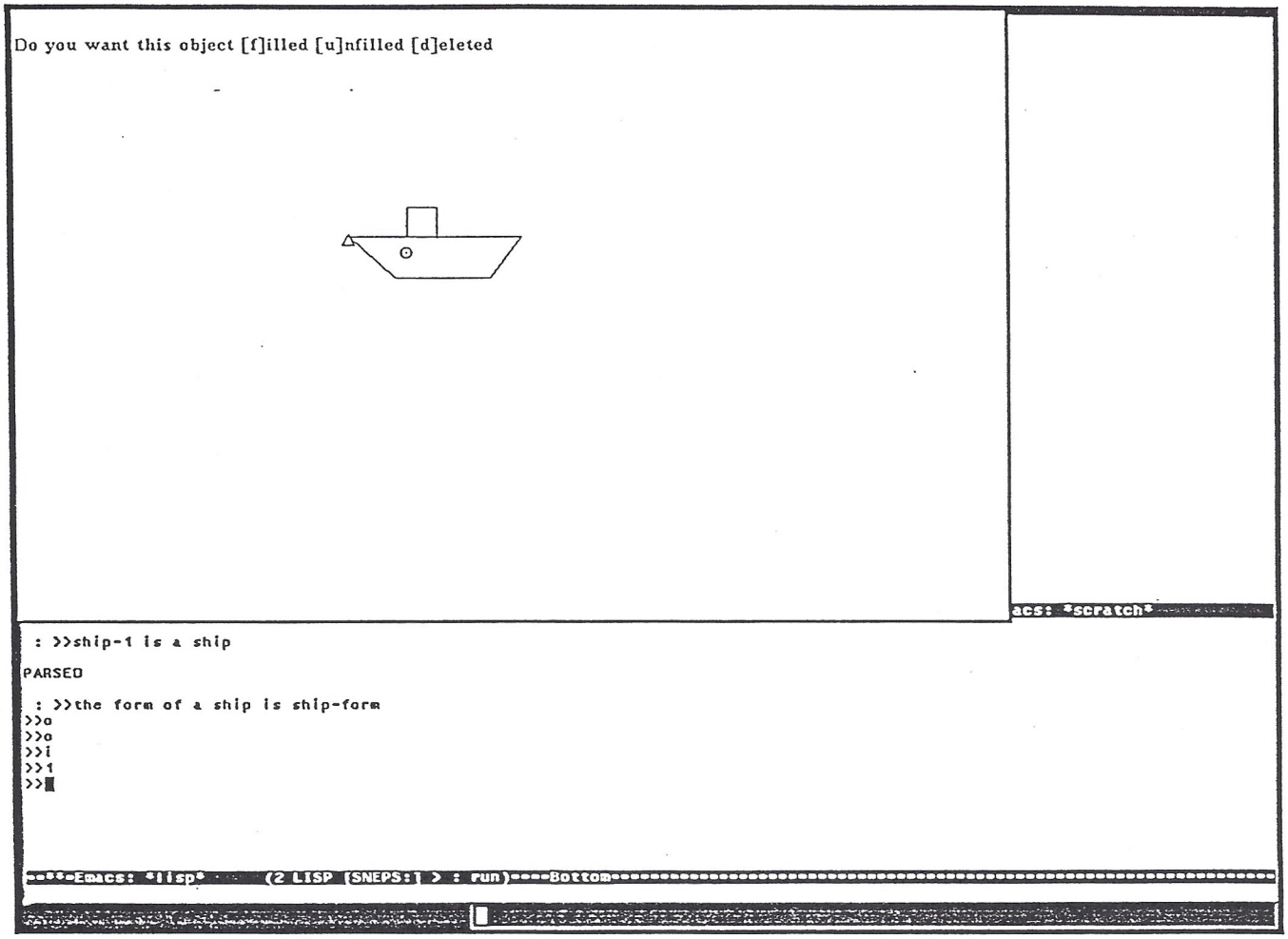

Do you want this object [f]illed [u]nfilled [d]eleted
acs: *scratch*
: >>ship-1 is a ship
PARSED
: >>the form of a ship is ship-form
>>o
>>o
>>i
>>1
>>
--**-Emacs: *lisp* (2 LISP [SNEPS:1 > : run)----Bottom

Fig. 6.2.3

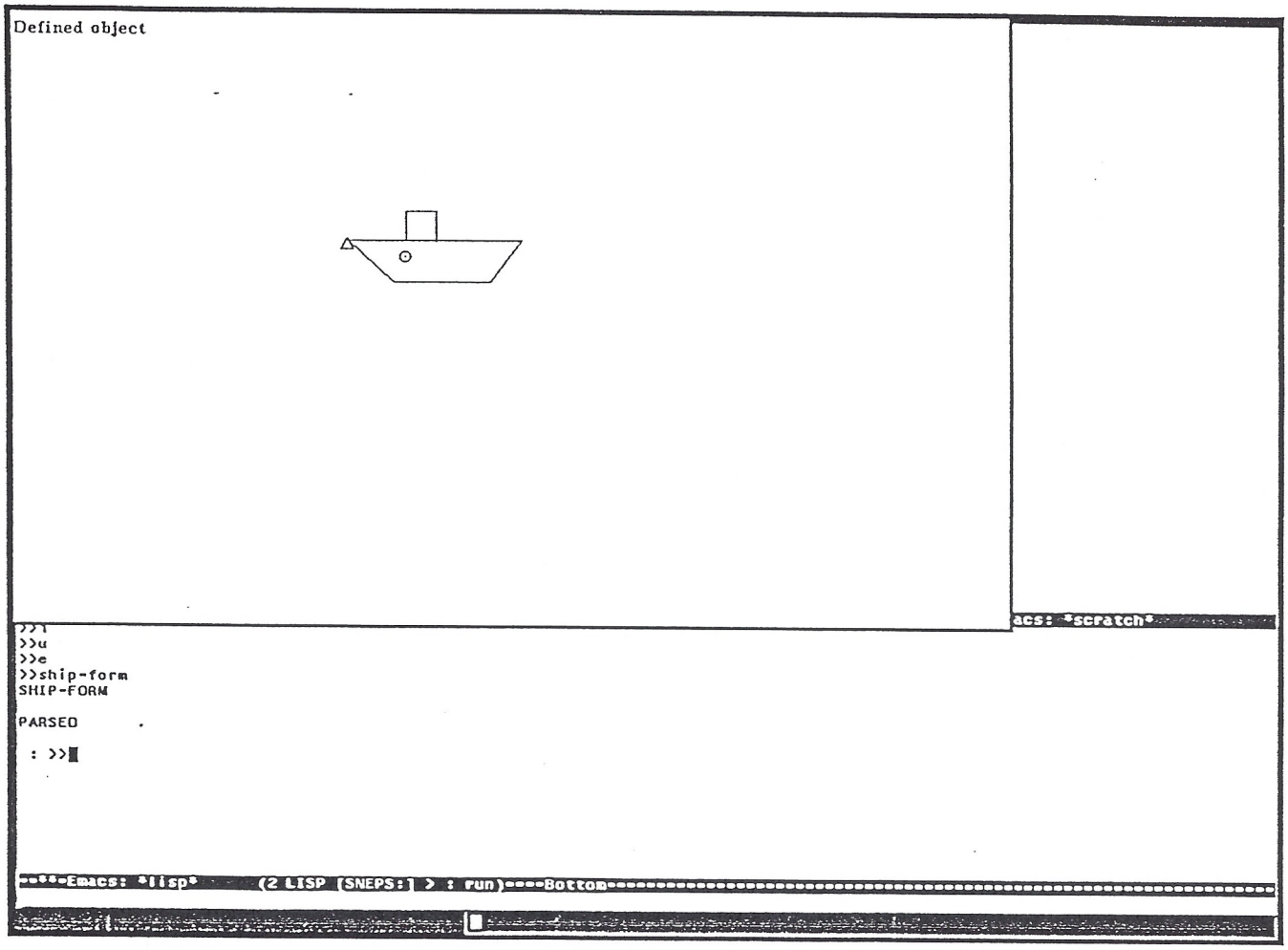

Defined object
acs: *scratch*
>>l
>>u
>>e
>>ship-form
SHIP-FORM
PARSED
: >>
--**-Emacs: *lisp* (2 LISP [SNEPS:] > : run)----Bottom----

Fig. 6.2.4

: >>the state of ship-1 is c4

[*Now an attribute is asserted about ship-1. The system has no knowledge about the meaning of either "state" or "c4". C4 is the official Navy terminology for "non-operational".*]

PARSED

: >>state is expressed by rotate-jg and 180 represents c4

[*Now "state" receives meaning by binding it to a modifier function and "c4" receives meaning as an argument to this function.*]

PARSED

: >>show ship-1

[*Ship-1 is redisplayed, and this time the attribute is correctly included. In other words, we have solved the challenge problem. This is the situation in Fig. 6.2.6*]

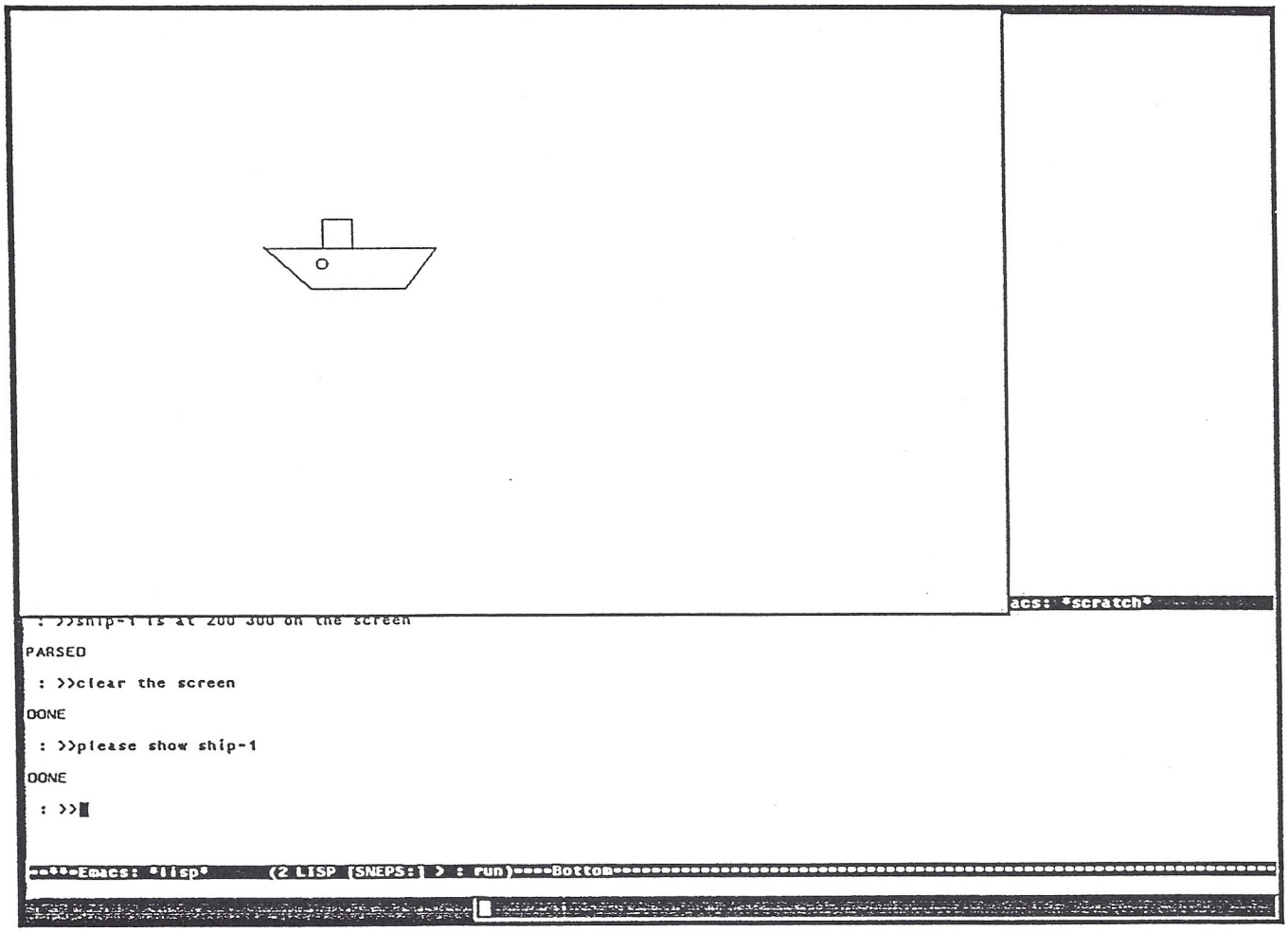

acs: *scratch*
: >>ship-1 is at 200 300 on the screen
PARSED
: >>clear the screen
DONE
: >>please show ship-1
DONE
: >>
--**-Emacs: *lisp* (2 LISP [SNEPS:1] > : run)----Bottom

Fig. 6.2.5

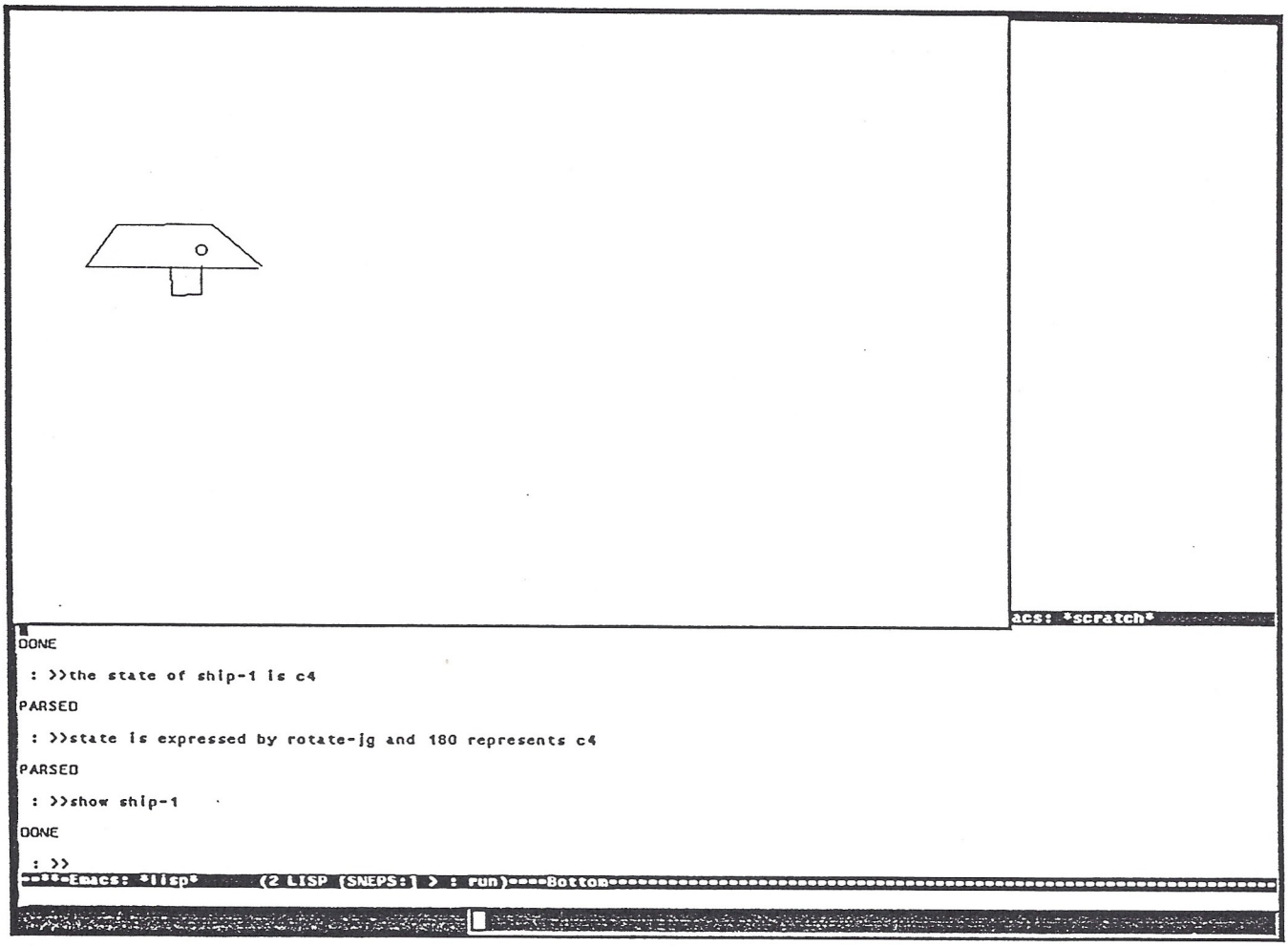

acs: *scratch*
DONE
: >>the state of ship-1 is c4
PARSED
: >>state is expressed by rotate-jg and 180 represents c4
PARSED
: >>show ship-1
DONE
: >>
----Emacs: *lisp* (2 LISP (SNEPS:] > : run)----Bottom----

Fig. 6.2.6

Demonstration 2

[*This is a small continuation of the previous demo. It shows that all the assertions made to the system are accessible to questioning.*]

: >>what is the form of ship-1?

[*The system responds in two different ways. It prints the name of the form, and it also demonstrates the form by drawing ship-1 (Fig. 6.2.7).*]

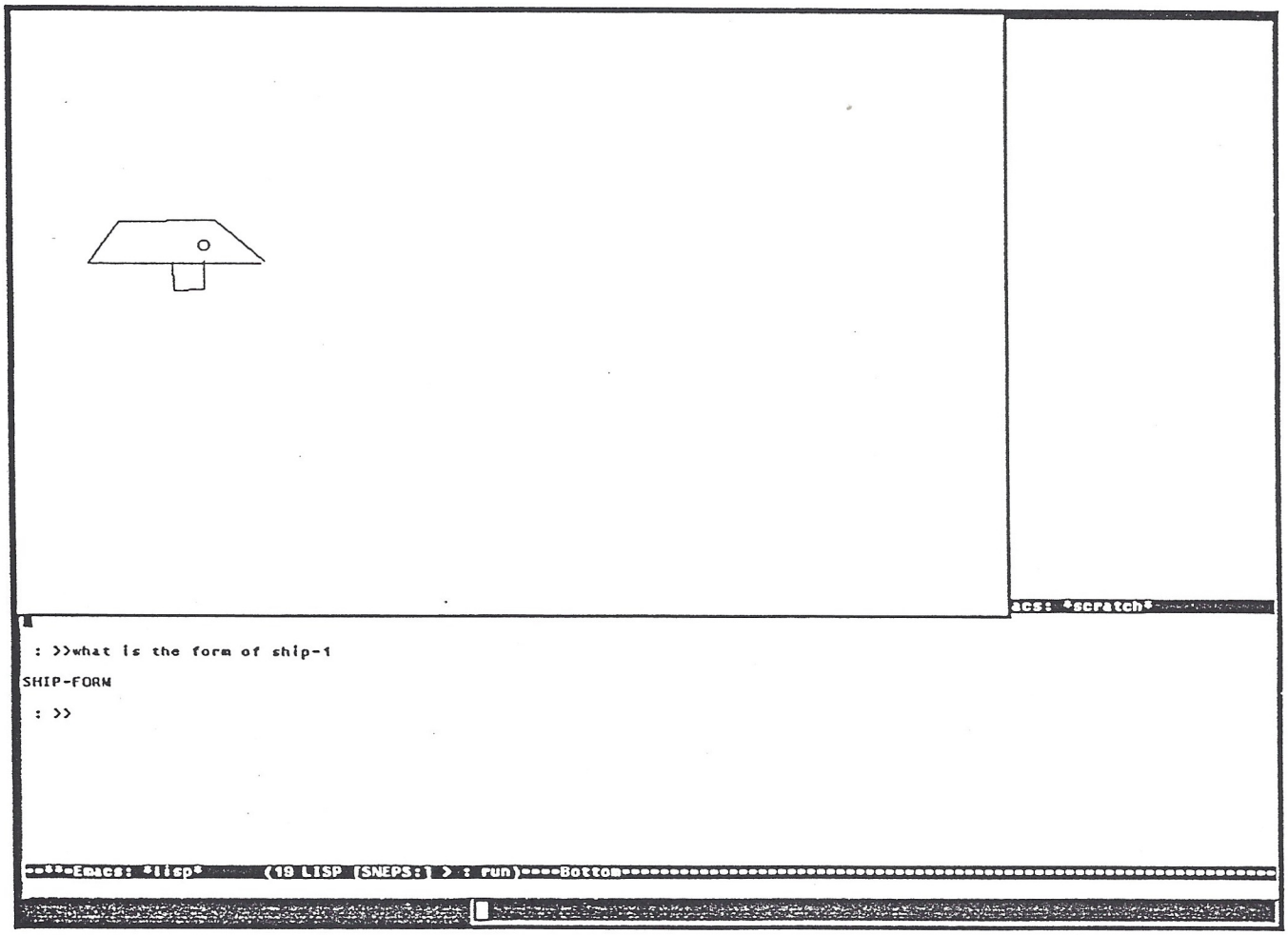


Fig. 6.2.7

SHIP-FORM

: >>what is the state of ship-1?

C4

: >>where is ship-1?

(200 300 " relative to" screen-center)

: >>what are the members of ship?

[*This is the state of Fig. 6.2.8*]

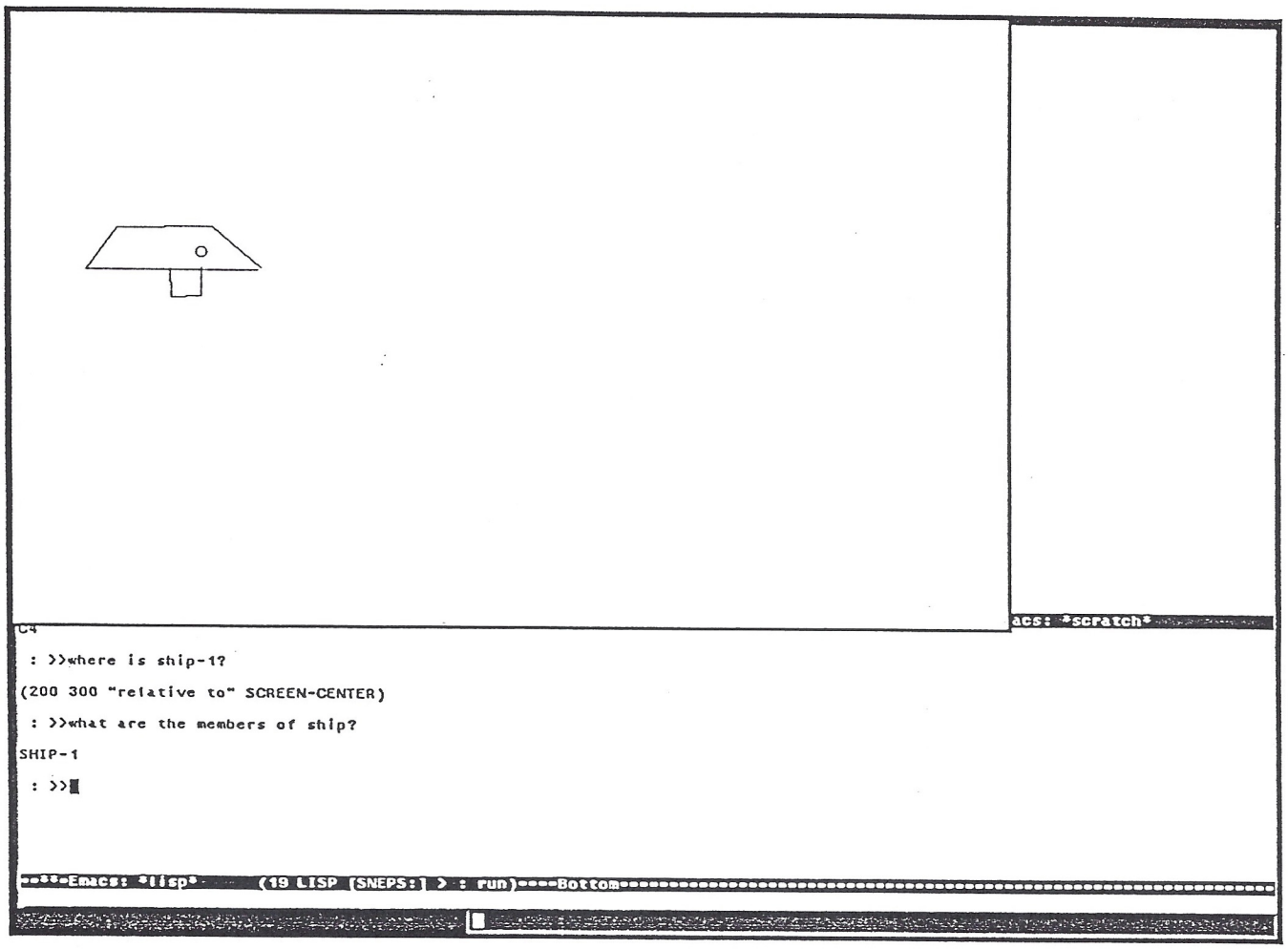

acs: *scratch*
C4
: >>where is ship-1?
(200 300 "relative to" SCREEN-CENTER)
: >>what are the members of ship?
SHIP-1
: >>
-:**-Emacs: *lisp* (19 LISP [SNEPS:] > : run)----Bottom

Fig. 6.2.8

SHIP-1

: >>elmer-montgomery is a ship

PARSED

: >>elmer-montgomery is here

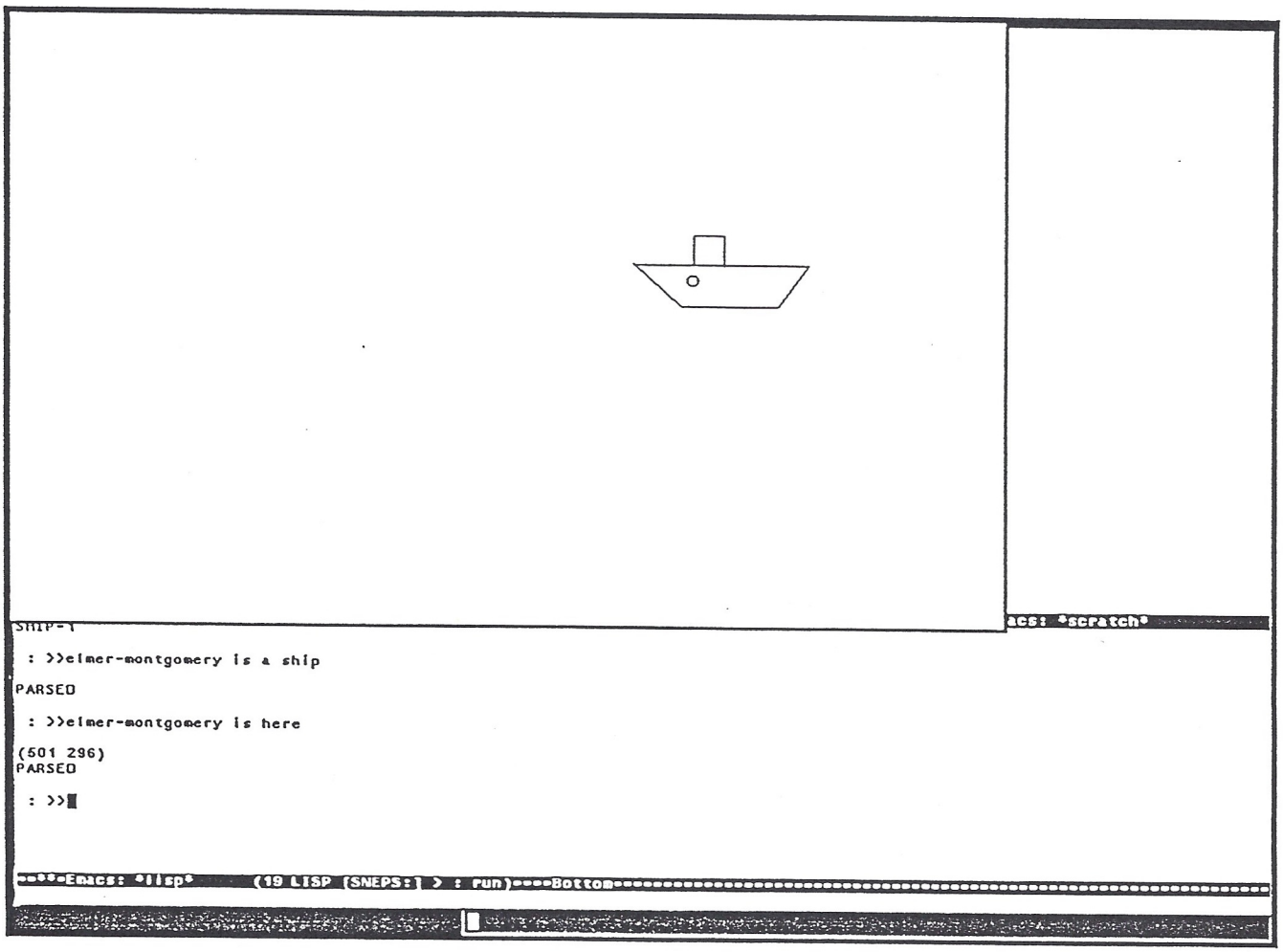


Fig. 6.2.9

[*"here" activates the graphics cursor read-out routine. After selecting a position the elmer-montgomery is immediately drawn and the position is printed out. This is shown in Fig. 6.2.9*]

(501 296)

PARSED

: >>what are the members of ship?

[*Now there are two members, and the system reacts appropriately (Fig. 6.2.10).*]

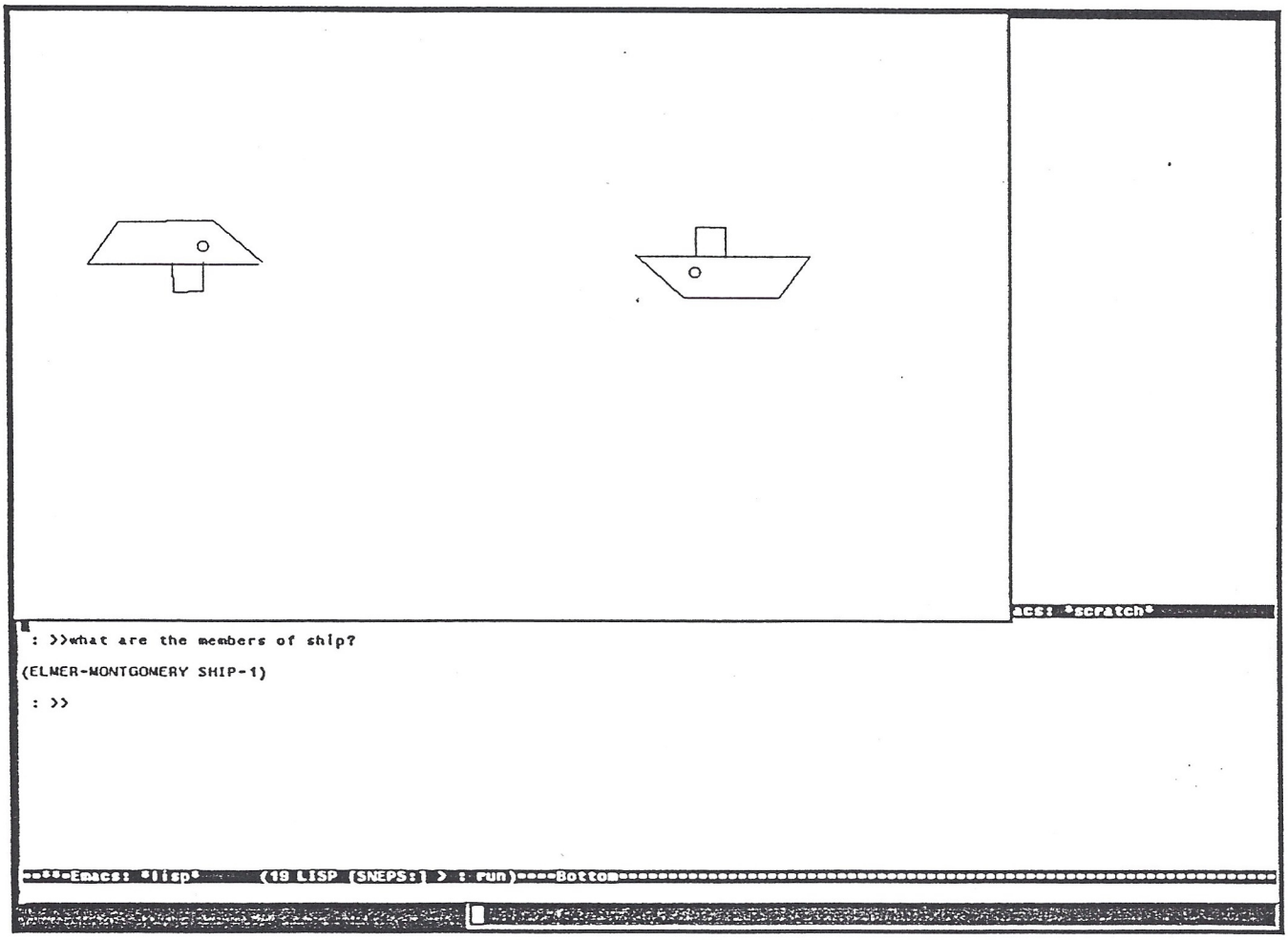


Fig. 6.2.10

(ELMER-MONTGOMERY SHIP-1)

: >>erase the screen

DONE

: >>how would elmer-montgomery look with state c4?

(M96! OBJECT (ELMER-MONTGOMERY) ATTR (M20))

node dismantled.

[*Above simple form of hypothetical reasoning is implemented by building and then erasing a structure asserting the c4 attribute. The two lines after the request are responses from the SNePS erase function. See Fig. 6.2.11*]

SO

: >>show elmer-montgomery

[*Now the ship is displayed in its normal position again. We did not tell the program that the elmer-montgomery actually IS c4. See Fig. 6.2.12 for this final screen dump.*]

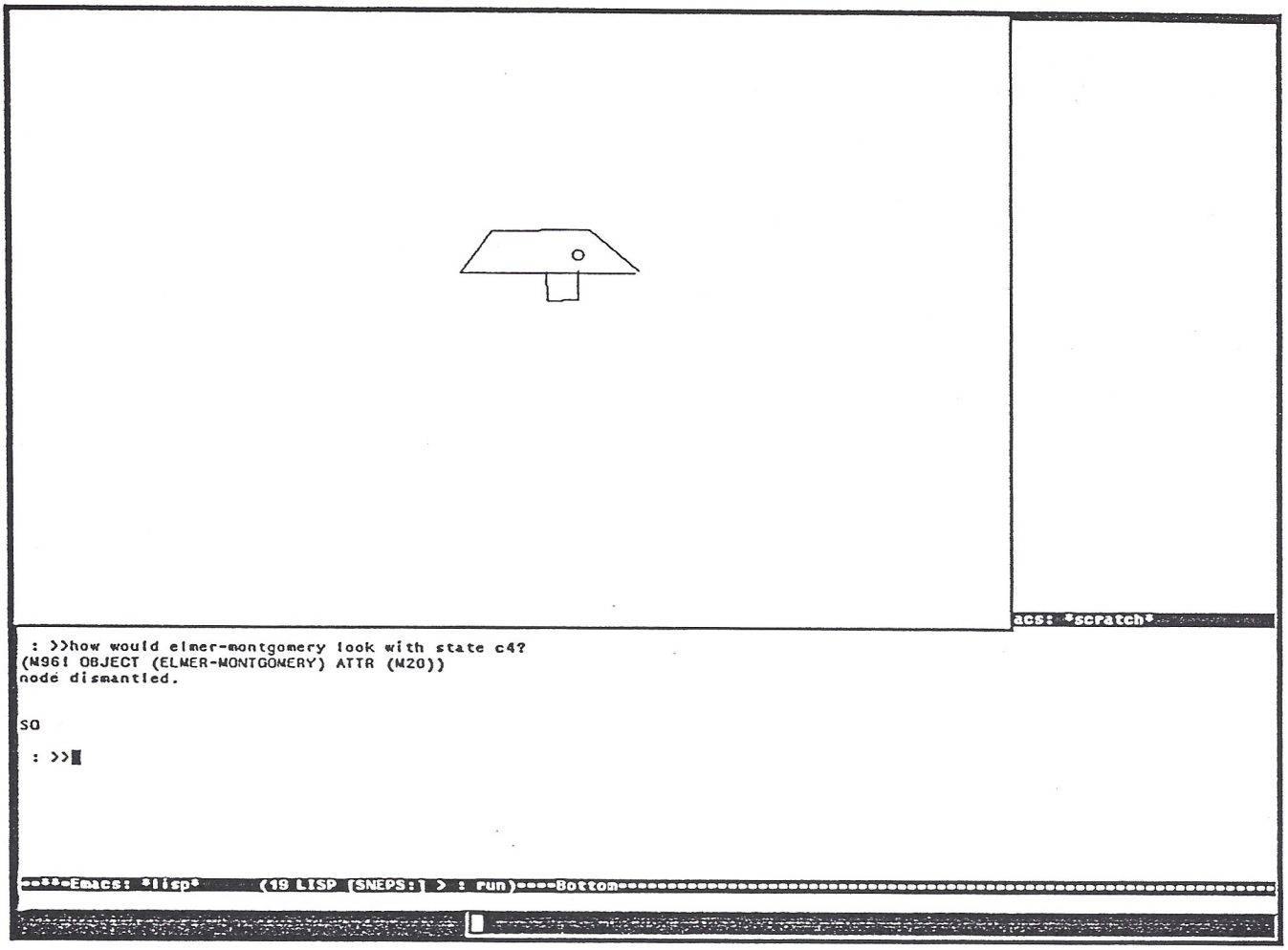


Fig. 6.2.11

DONE

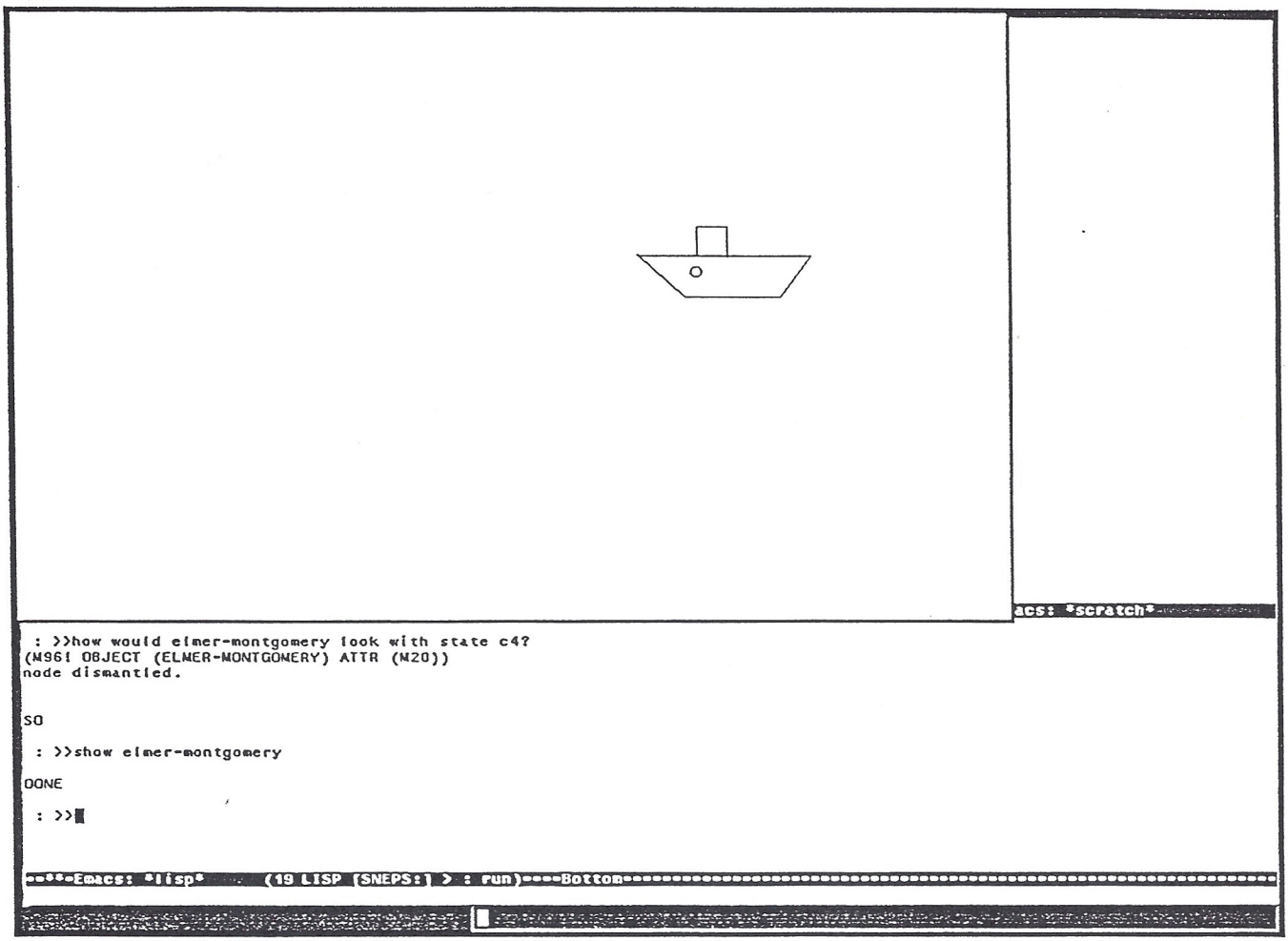

acs: *scratch*
: >>how would elmer-montgomery look with state c4?
(M961 OBJECT (ELMER-MONTGOMERY) ATTR (M20))
node dismantled.
SO
: >>show elmer-montgomery
DONE
: >>
--**-Emacs: *lisp* (19 LISP (SNEPS:) > : run)----Bottom

Fig. 6.2.12

Demonstration 3

[*This demo assumes that a number of electronic form primitives have already been defined. A screen coordinate system is also known. It demonstrates several features, most importantly reasoning with a merological syllogism, i. e. parts of a part of an object are also parts of the object.*]

: >>the form of a board is xboard

PARSED

: >>b2 is a board

PARSED

: >>b2 is at 2 6 inches on the screen

PARSED

: >>show b2

[*Fig. 6.2.13*]

DONE

: >>b2 has and-1 and and-2 as parts

[*Parts are introduced at this point.*]

PARSED

: >>and-1 and and-2 are members of and-gate

[*A one level class hierarchy is built.*]

: >>the form of an and-gate is xand

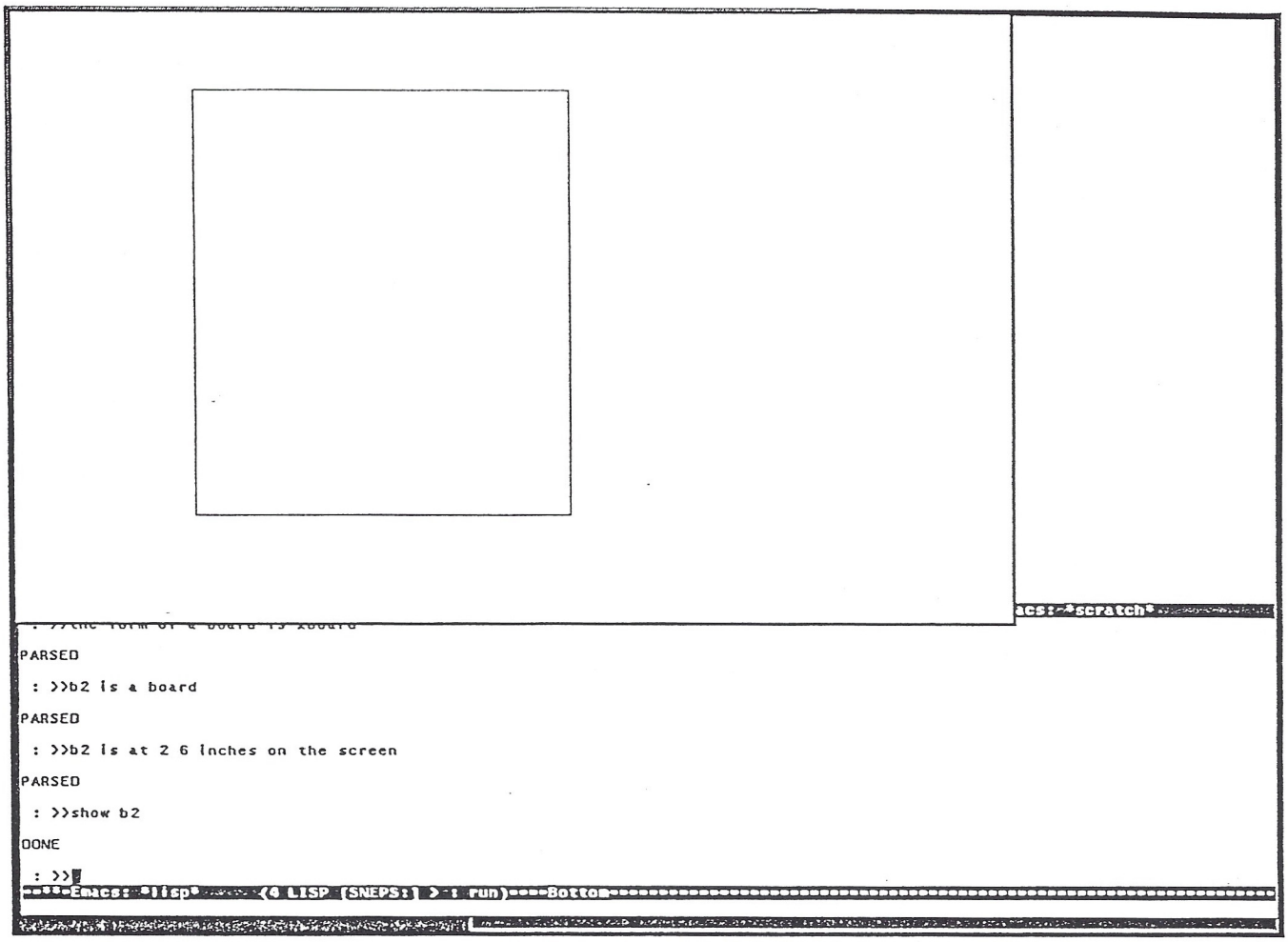


Fig. 6.2.13

PARSED

: >>and-1 is at 1 -1 inches relative to b2

PARSED

[*A relative position relative to a reference object is given.*]

: >>and-2 is at 1 -3 inches relative to b2

PARSED

: >>erase the screen

DONE

: >>show us b2

[*Simply asserting parts of an object does not influence the reaction to a natural language request for display. This is shown in Fig. 6.2.14*]

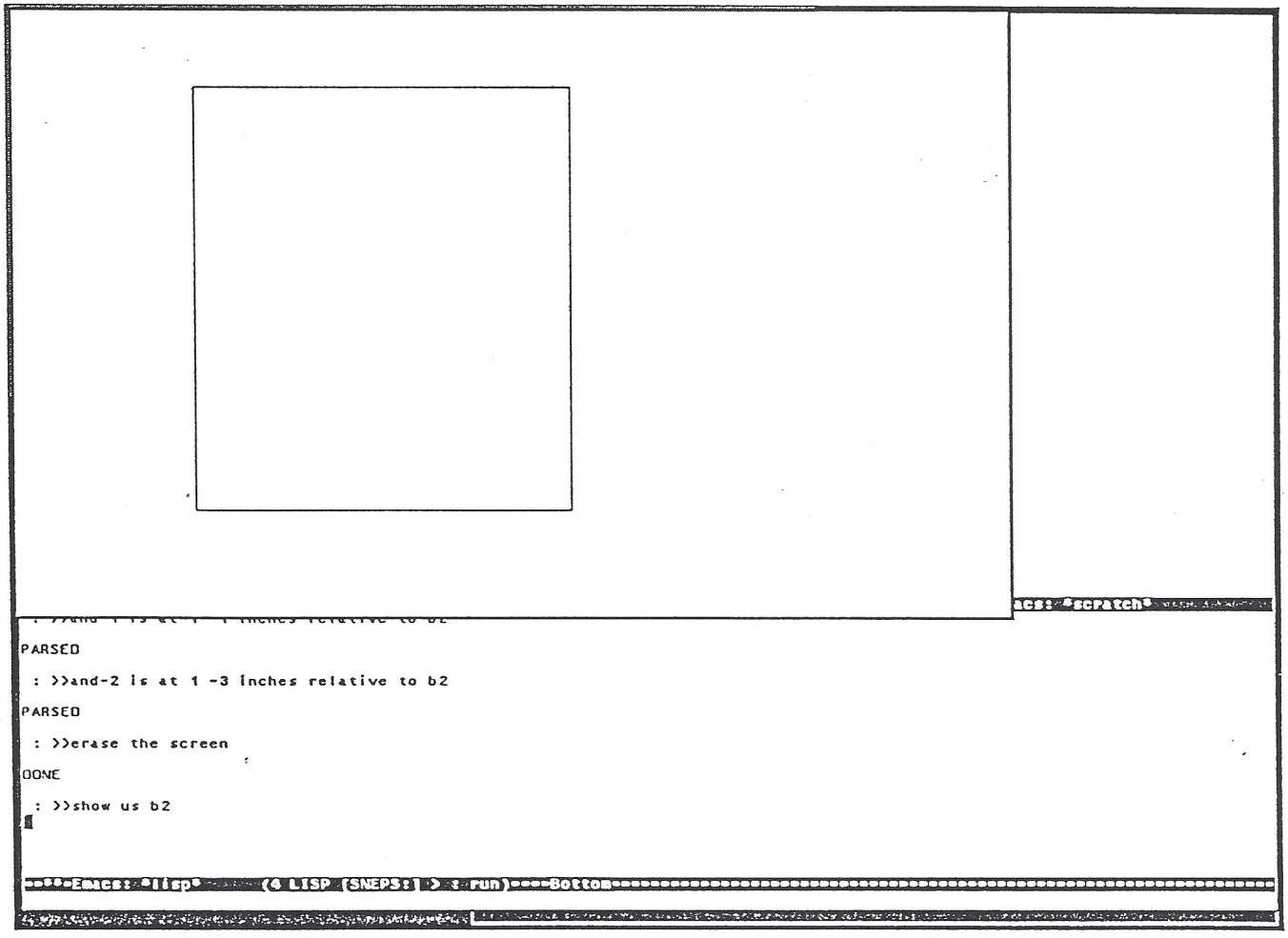


Fig. 6.2.14

DONE

: >>clear the screen

DONE

: >>show 2 levels of b2

[*By using the part hierarchy as selection mechanism, a display of b2 with its parts is generated. Refer to Fig. 6.2.15*]

DONE

: >>the form of a port is xport

PARSED

: >>port has upper-in-port, lower-in-port and out-port as sub-classes

[*A two level class hierarchy is built. The class "port" has three sub-classes: upper inports, lower inports and outports.*]

PARSED

: >>upper-in-port has and1-inp1 and and2-inp1 as members.

PARSED

: >>lower-in-port has and1-inp2 and and2-inp2 as members.

PARSED

: >>out-port has and1-outp and and2-outp as members.

[*Members of the sub-classes are asserted.*]

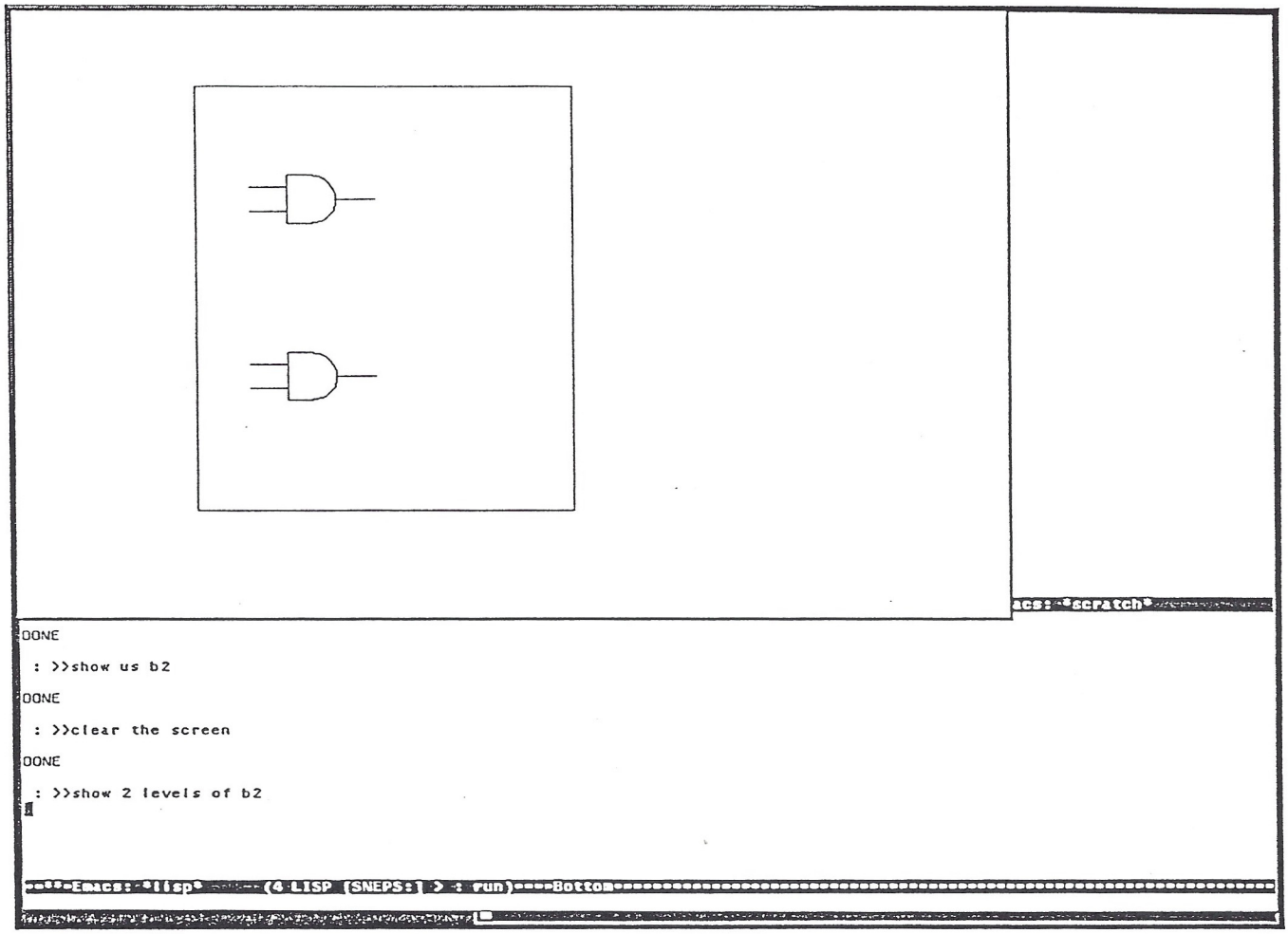


Fig. 6.2.15

PARSED

: >>and-1 has and1-inp1 and and1-inp2 and and1-outp as parts

PARSED

: >>and-2 has and2-inp1 and and2-inp2 and and2-outp as parts

[*A third level in the part hierarchy is introduced. and1-inp1 is a part of and-1, and therefore it is a second order part of b2.*]

PARSED

: >>members of upper-in-port are at -30 -5 relative to their super-part

PARSED

: >>members of lower-in-port are at -30 -25 relative to their super-part

PARSED

: >>members of out-port are at 60 -15 relative to their super-part

[*Above three sentences create relative positions that are associated with three classes and that may be inherited to individuals.*]

PARSED

: >>erase the screen

DONE

: >>display 2 levels of and-1

[*and-1 and its ports are displayed. Fig. 6.2.16 shows this.*]

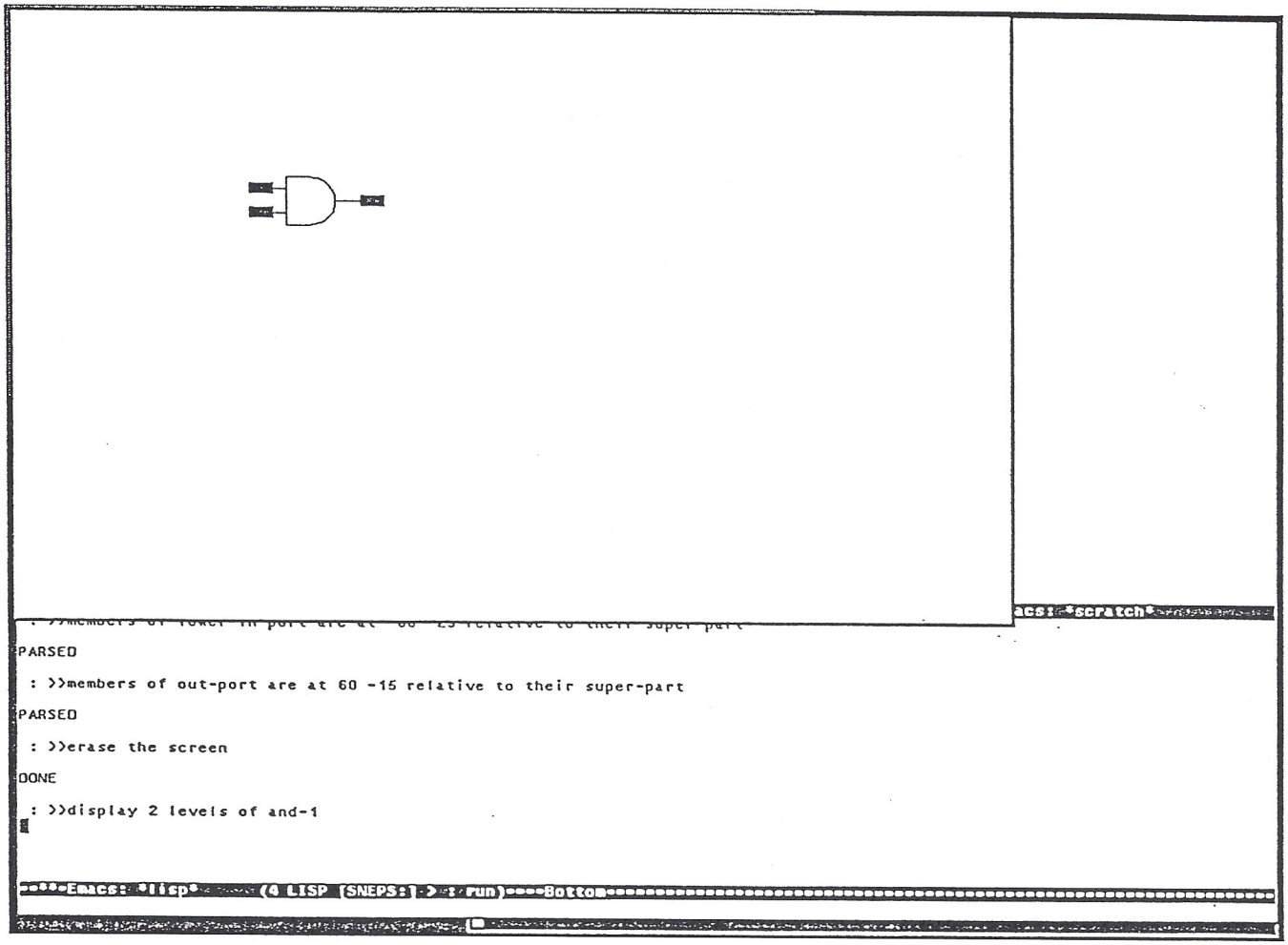


Fig. 6.2.16

DONE

: >>clear the screen

DONE

: >>display 2 levels of b2

[*The parts of and-1 are not displayed, because they are on the third level! (Fig. 6.2.17).*]

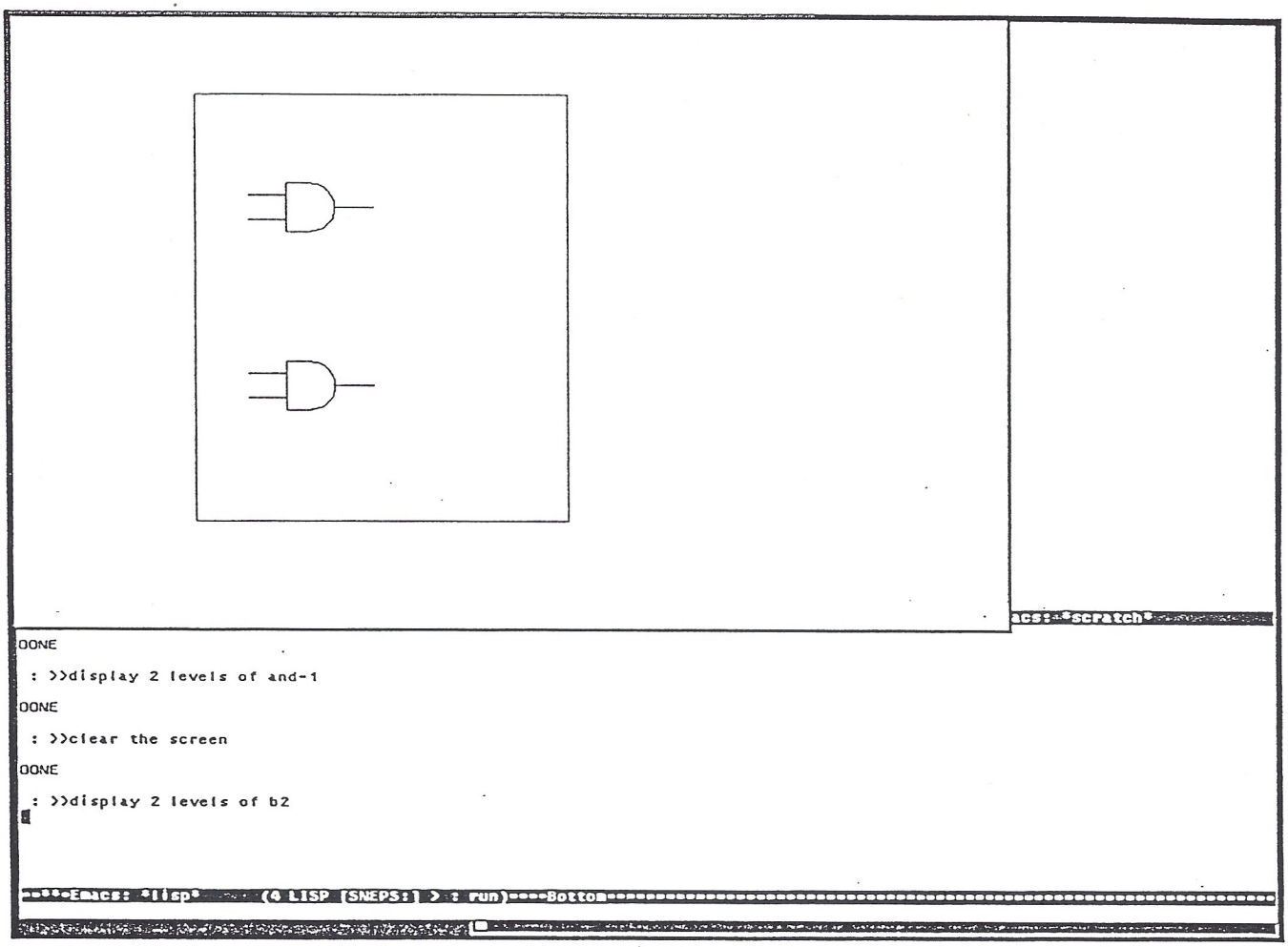


Fig. 6.2.17

DONE

: > >erase the screen

DONE

: > >display 3 levels of b2

[b2 with all its parts and the parts of its parts are displayed. In other words, this time the parts of and-1 are displayed. Fig. 6.2.18 shows this.]

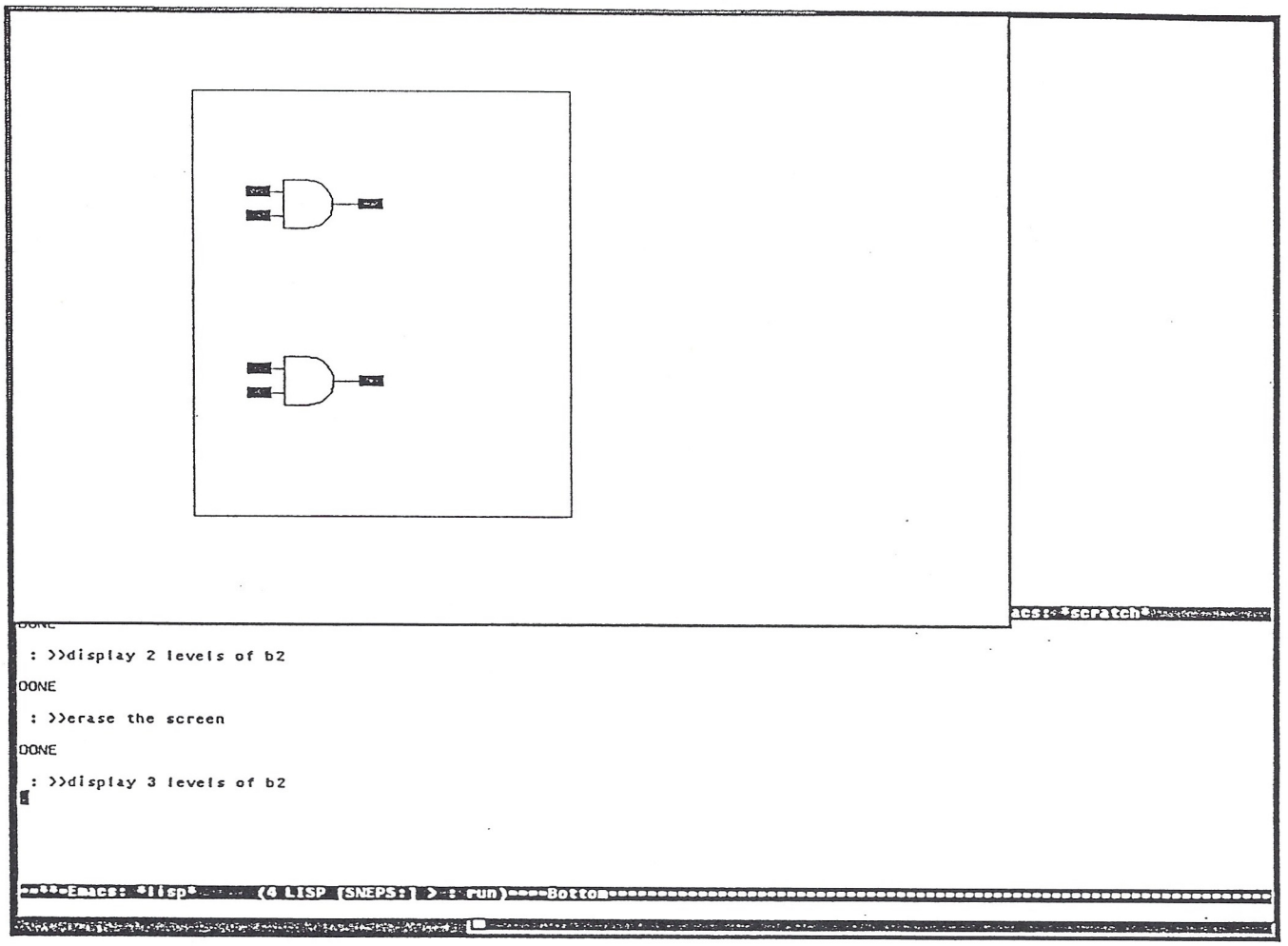


Fig. 6.2.18

DONE

: >>clear the screen

DONE

: >>fill 100 100 300 300 with and-1

[*This introduces another feature of the system. Objects can be displayed in sub-windows and will be stretched such that the window is optimally filled, however the x/y ratio of the object size is maintained. (Fig. 6.2.19).*]

DONE

: >>clear the screen

DONE

: >>show and-1 with its environment

[*This demonstrates "environment mode". It shows an object, its super-part and its siblings. In fact it shows also the first level parts of the object and the siblings of its super parts, up to the main*

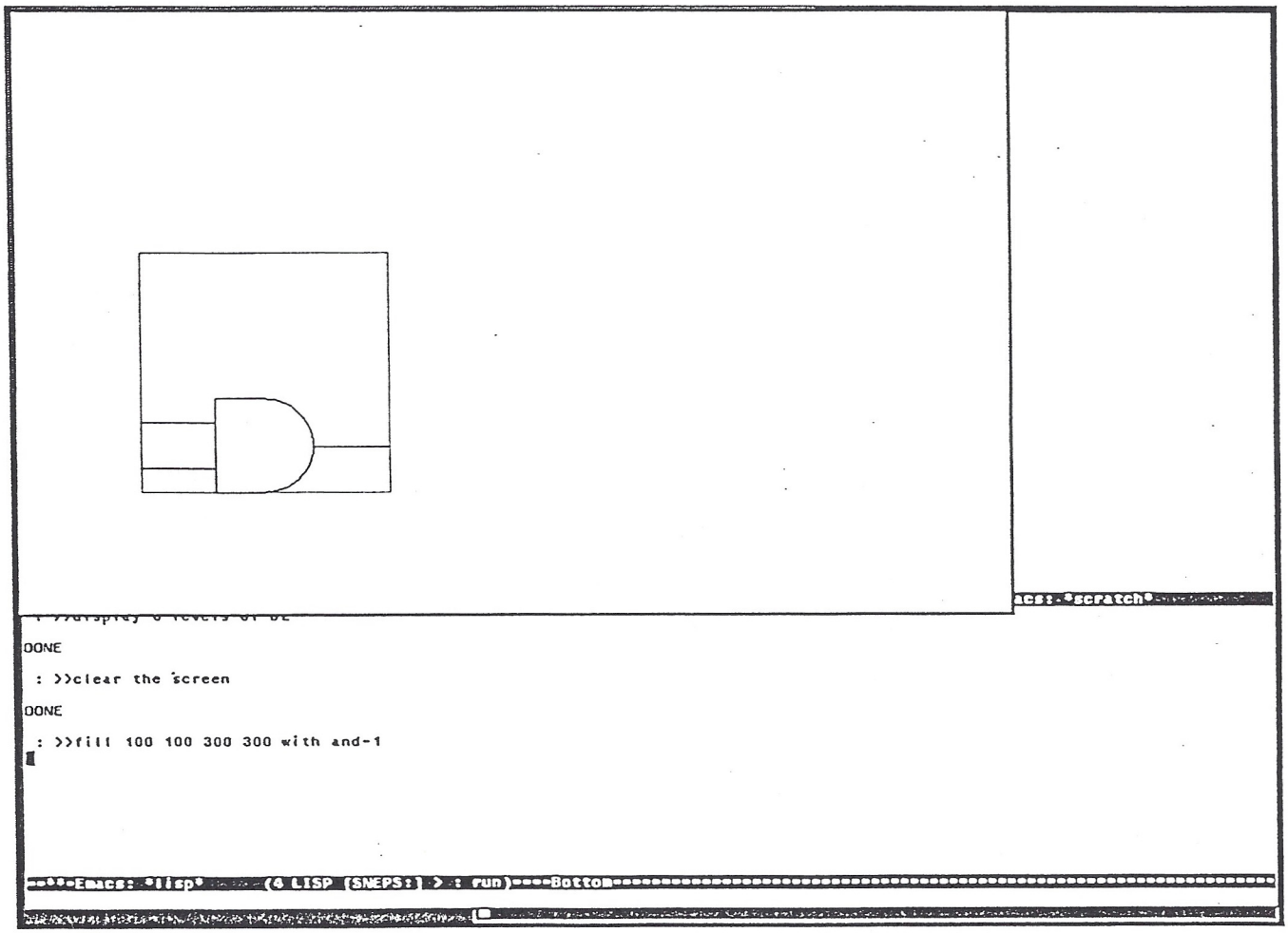


Fig. 6.2.19

object. However, the latter cannot be observed here, because the part hierarchy is too flat. The AND gate in the left window corresponds to the AND gate in the right window which is marked by "thickening". On some terminals this correspondence can be expressed by blinking instead of "thickening". The garbage collection has been edited out from the text. (Fig. 6.2.20)]

DONE

: >>clear the screen

DONE

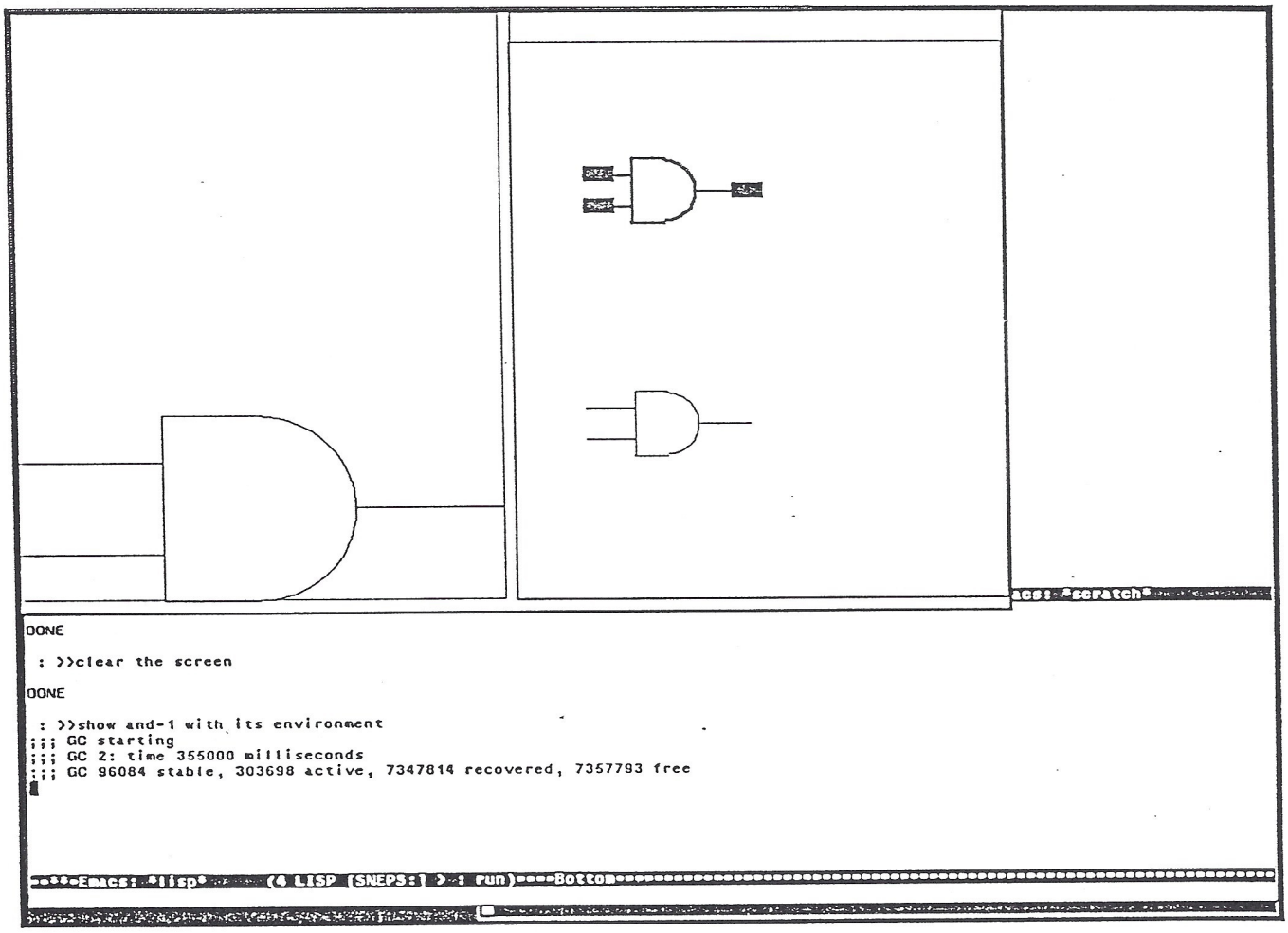


Fig. 6.2.20

: >>what are all the parts of b2?

[*Here the merological syllogism is demonstrated. The question is answered textually and graphically. (Fig. 6.2.21)*]

(AND-1 AND-2 AND1-INP1 AND1-INP2 AND1-OUTP AND2-INP1 AND2-INP2 AND2-OUTP)

: >>

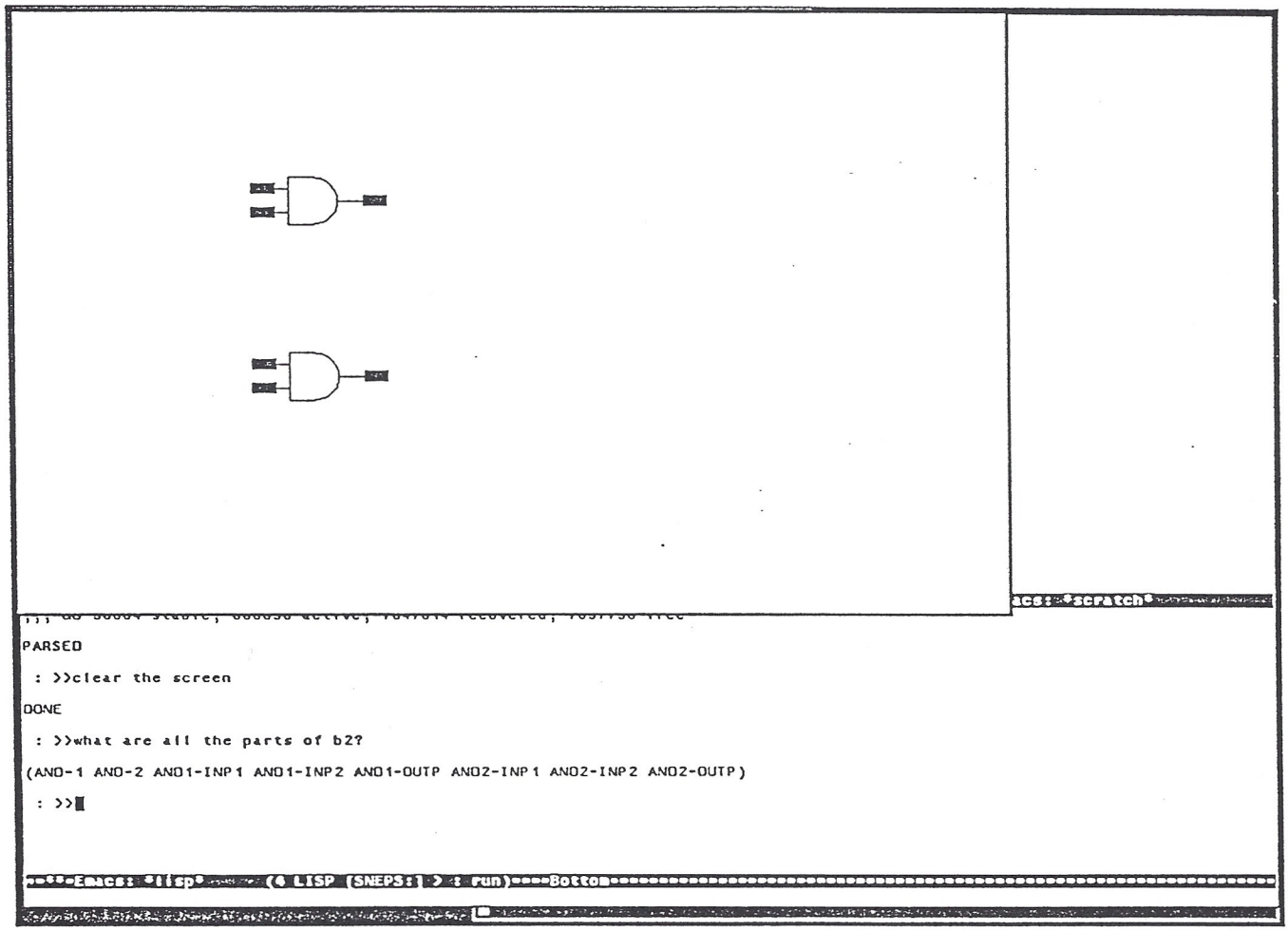


Fig. 6.2.21

Demonstration 4

[*The same coordinate system as in the previous demonstrations is assumed. This demonstration shows the use of two modalities and demonstrates the effect of using a sub-assembly instead of a real part.*]

: >>the form of and-1 is xand

PARSED

: >>and-1 is at 3 3 inches on the screen

PARSED

: >>show and-1

DONE

: >>the assumed modality is structure

[*The standard modality is now changed.*]

PARSED

: >>the form of and-1 is struct-and

[*It is necessary to supply a completely new form and position in the new modality.*]

PARSED

: >>and-1 is at 1 1 inches on the screen

PARSED

: >>show and-1

[*This is a structural display, and completely independent from the functional display. (Fig. 6.2.22).*]

: >>clear the screen

DONE

: >>the assumed modality is function

[*Now we are switching back to the previous modality.*]

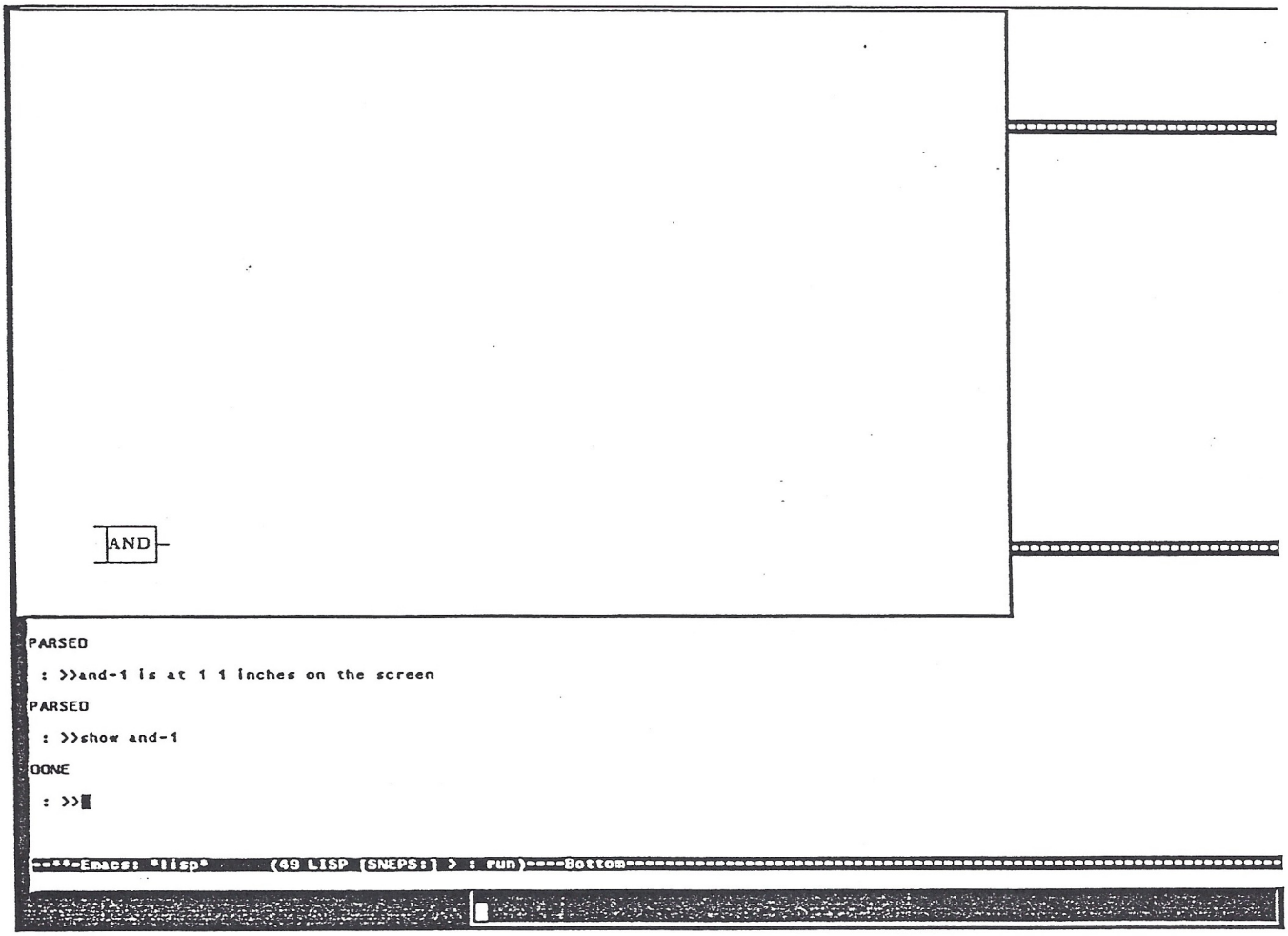


Fig. 6.2.22

DONE

: >>show and-1

[*All functional information is still maintained. Therefore we now get the familiar display of an AND gate. Fig. 6.2.23*]

DONE

: >>port-1, port-2 and port-3 are sub-assemblies of and-1

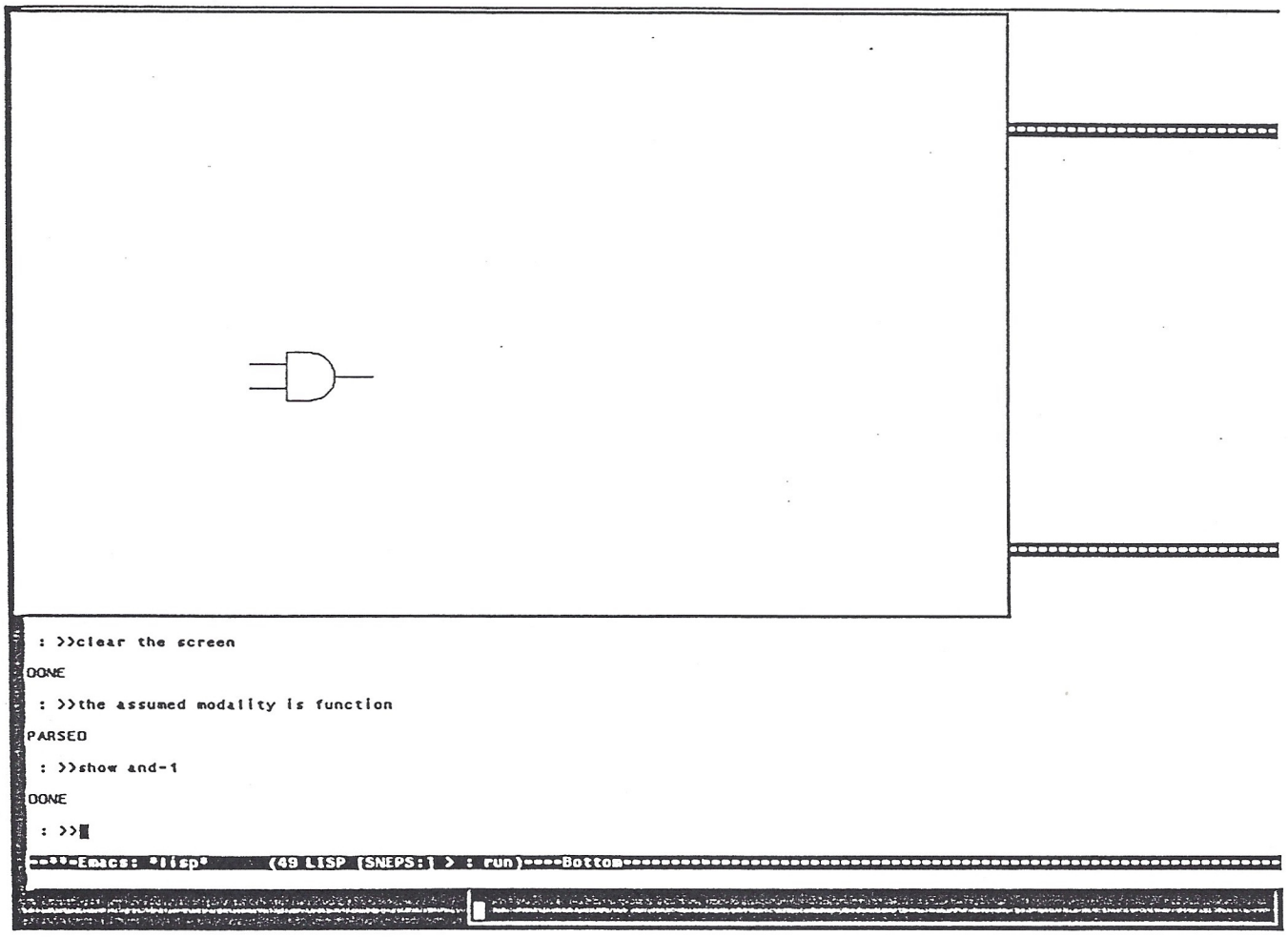


Fig. 6.2.23

[*Sub-assemblies are different from "real-parts" that were used before.*]

PARSED

: >>the form of a port is xport

PARSED

: >>port-1 and port-2 and port-3 are members of port

PARSED

: >>port-1 is at -30 -5 relative to and-1

PARSED

: >>port-2 is at -30 -25 relative to and-1

PARSED

: >>port-3 is at 60 -15 relative to and-1

PARSED

: >>show 2 levels of and-1

[*Fig. 6.2.24*]

PARSED

: >>clear the screen

PARSED

: >>show 1 level of and-1

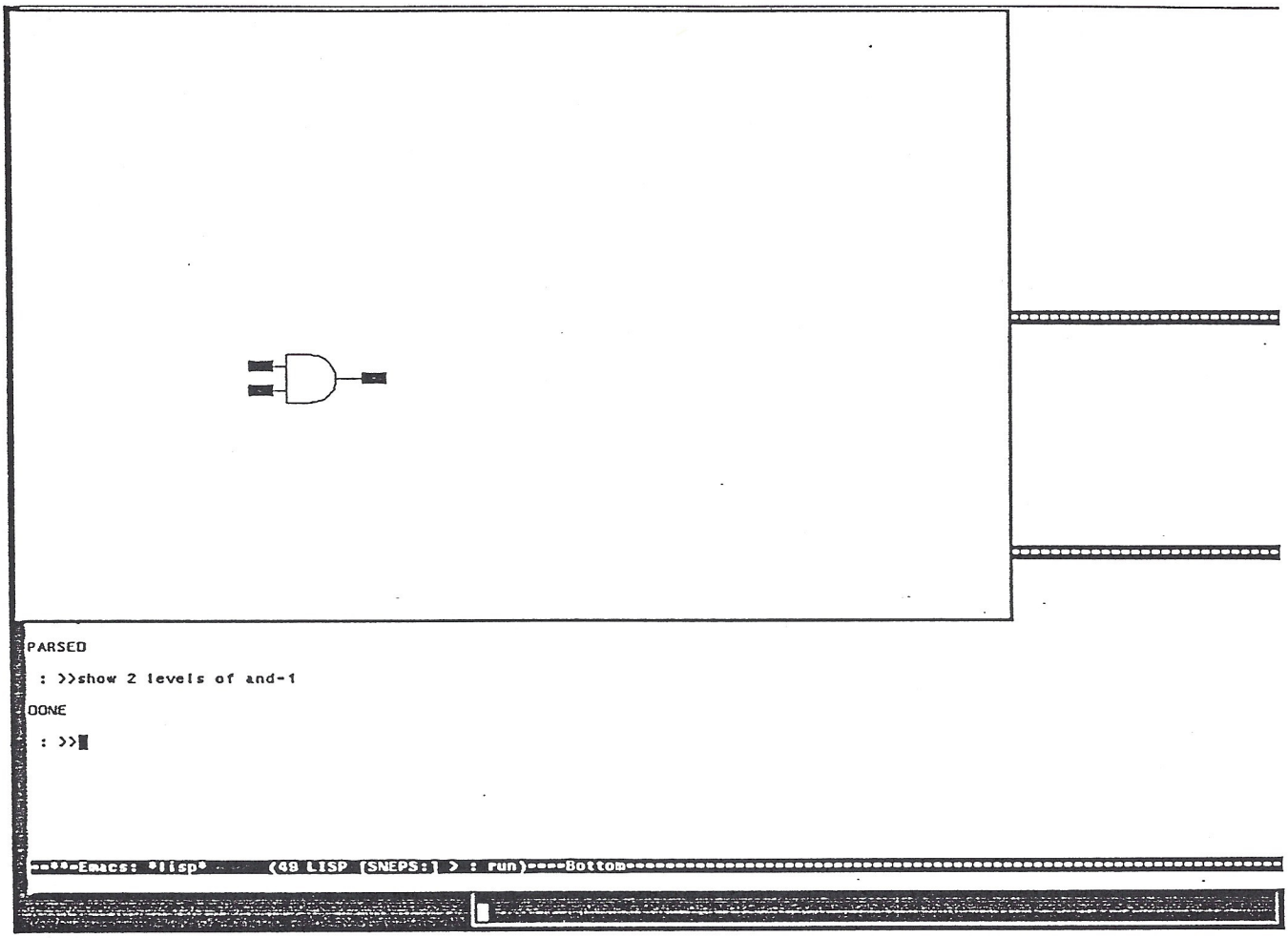


Fig. 6.2.24

[*Note that the display is identical to before! Sub-assemblies are non separable parts! Fig. 6.2.25*]

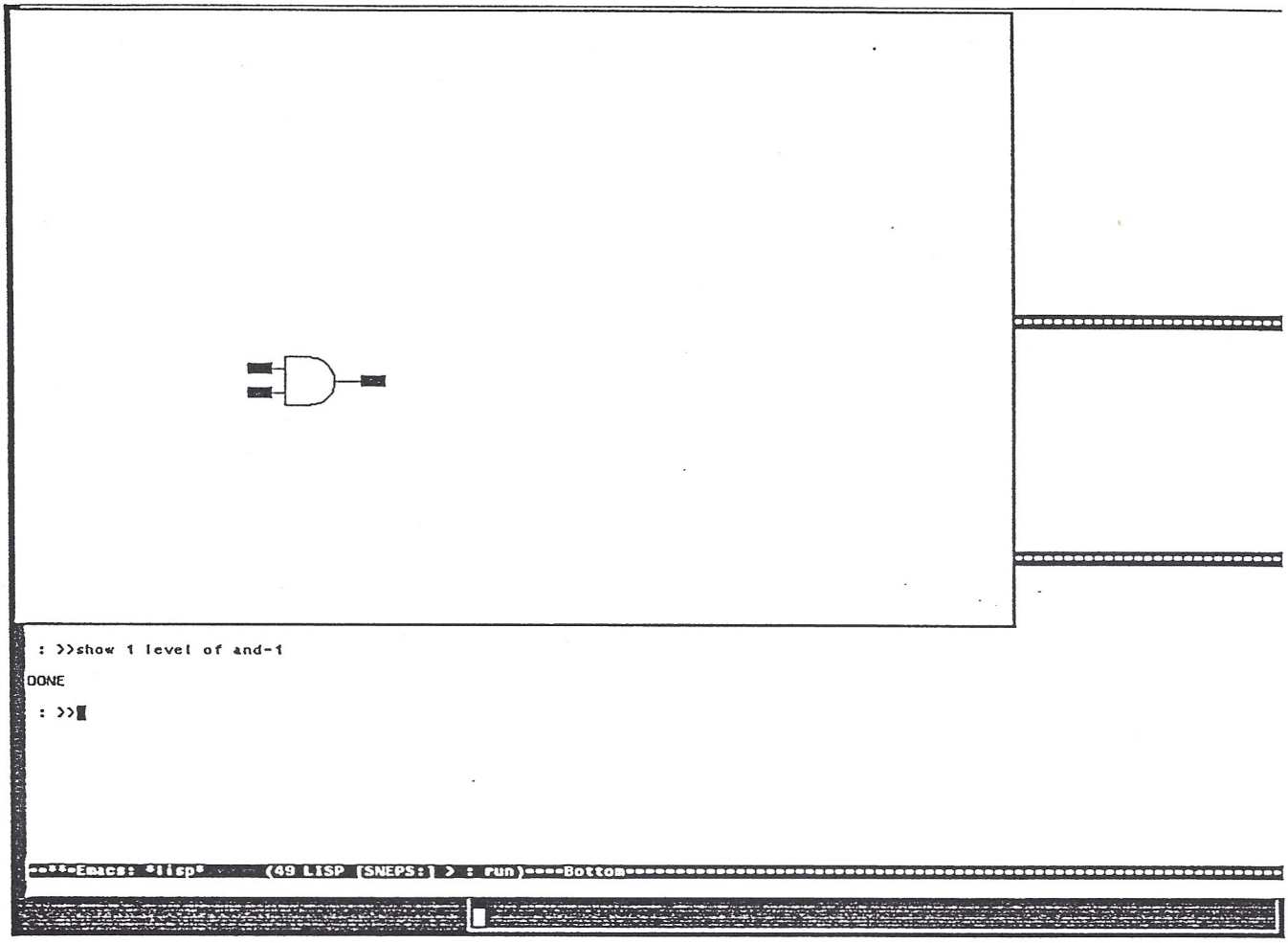


Fig. 6.2.25

Demonstration 5

[*As in the previous demos, a screen coordinate system and modality are established first. This demo shows the difference between plane and screen coordinates, and it also shows examples of fuzzy positioning.*]

: >>the form of and-1 is xand

PARSED

: >>the form of my-or-gate-99 is xor

PARSED

: >>the form of the-second-not is xnot

PARSED

: >>and-1 is at 3 4 inches on the screen

PARSED

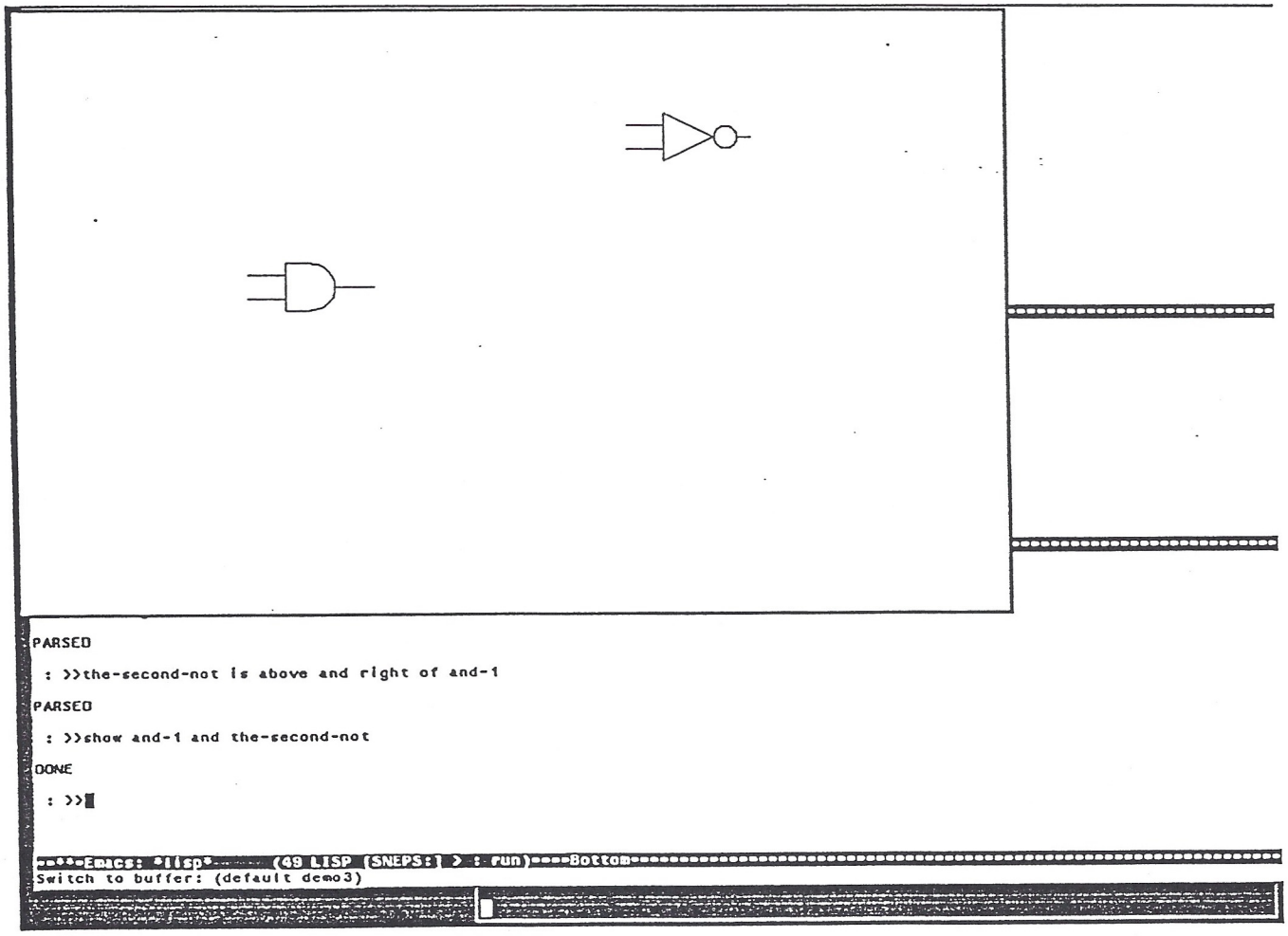


Fig. 6.2.26

: >>my-or-gate-99 is left of and-1

PARSED

: >>the-second-not is above and right of and-1

PARSED

: >>show and-1 and the-second-not

[*The basic placing algorithm operates the following way. The extent of the object for which only fuzzy information is available is located such that the empty strip between the two objects has the same width and/or length as the empty strip between the placed object and the screen border. "Extent" hereby denotes the smallest enclosing box. Fig. 6.2.26 shows the result of this placing.*]

DONE

: >>show my-or-gate-99, and-1 and the-second-not

[*Fig. 6.2.27*]

DONE

: >>fill 400 100 600 200 with and-1

DONE

: >>where is and-1

[*Note that the reply corresponds to the pixel values of the inch values specified by the user. One inch corresponds to 73 pixels. The given position was (3 4) inches. 3 * 73 = 219; 4 * 73 = 292.*]

(219 292 " relative to" SCREEN-CENTER)

: >>where is the picture of and-1?

[*Note in Fig. 6.2.28 that the system marks the actual position with a little plus sign at the upper left*

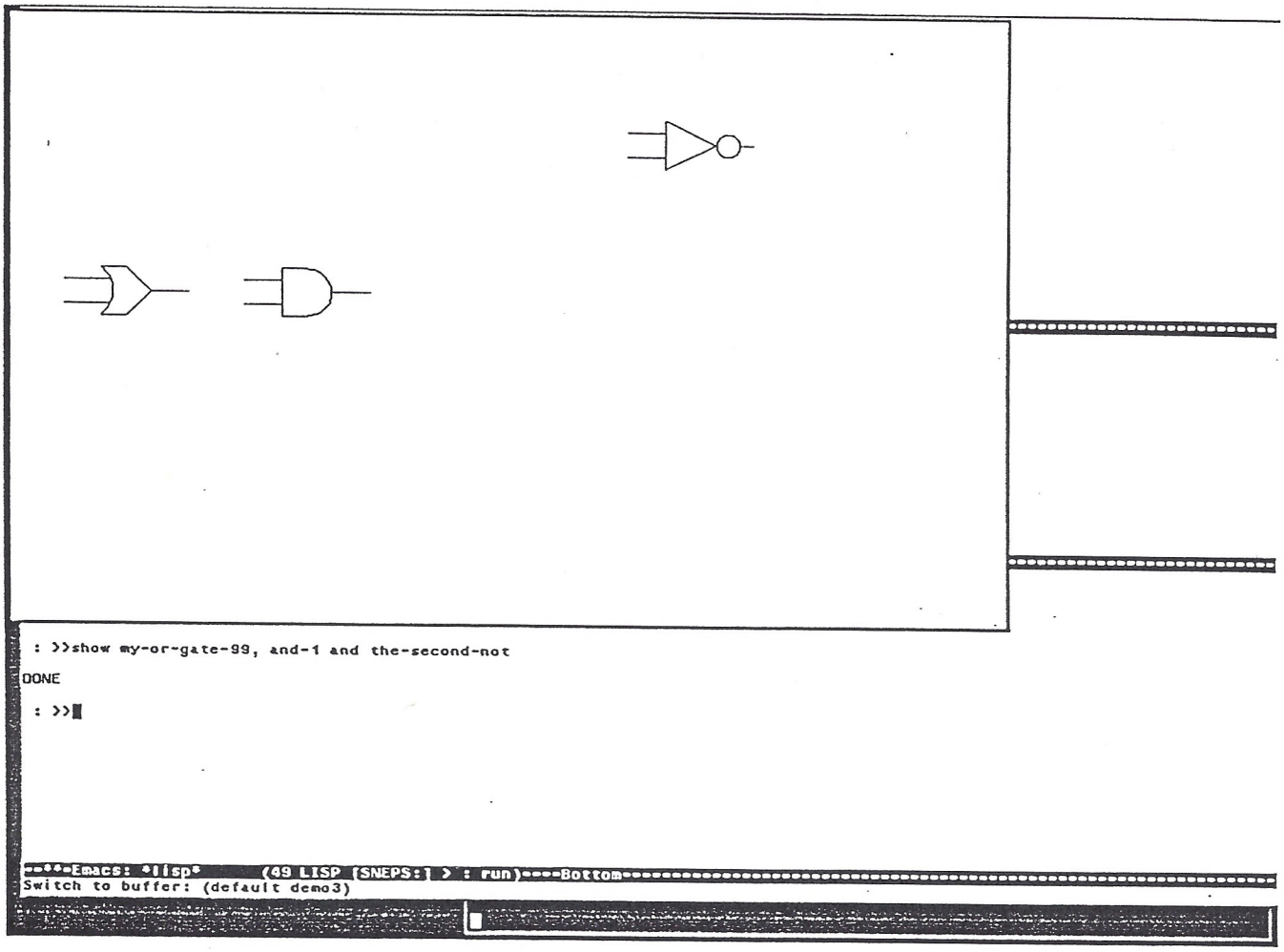


Fig. 6.2.27

corner of the body of the AND gate. This is helpful, because it shows where the reference point of the object is. More importantly note that this position is different from the one reported before. The system makes a difference between the position of an icon and the position of the object that it displays. This clarifies the difference between a plane coordinate and a screen coordinate value. What the user refers to as screen coordinates are in fact plane coordinates, because the system might change them for a :fill operation. Real screen coordinates are for positions of icons. Positions of icons are never asserted by the user but are a side effect of drawing an object. The system keeps a whole history of screens and the forms and positions of icons on them. So the answer (460 180) represents the position of the icon of and-1 relative to the lower left corner of the graphics window.]

(460 180)

Demonstration 6

[*The usual coordinate system definition is assumed. This is a short demonstration that shows why it makes sense to have an object "behind" another object, even if only a plane coordinate system is used.*]

: >>the form of icon1 is form1

[*Icon1 is a vertical blue rectangle FILLED white. Actually the graphics system does not permit to fill a rectangle with a different color than its border line, but this problem can be bypassed by specifying a second slightly smaller white rectangle which is filled. This white rectangle is of course*

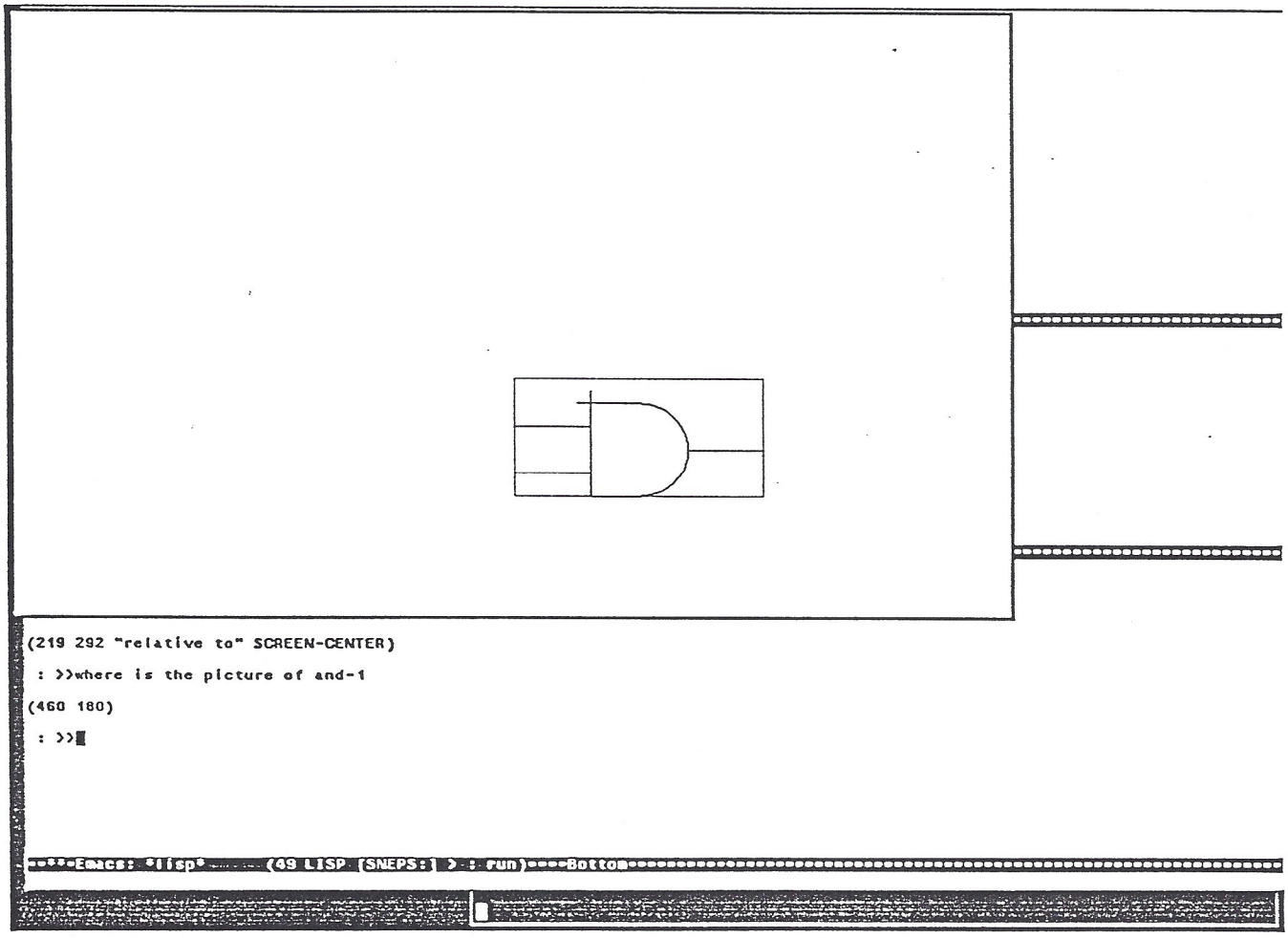


Fig. 6.2.28

invisible on white background.]

PARSED

: >>icon1 is at 200 300 on the screen

PARSED

: >>the form of icon2 is form2

[*Icon2 is a horizontal blue rectangle FILLED white.*]

PARSED

: >>icon2 is behind icon1

PARSED

: >>clear the screen

DONE

: >>show icon1 and icon2

[*icon2 is drawn first, which results in icon1 overdrawing it later, and this creates the behind effect. (Fig. 6.2.29)*]

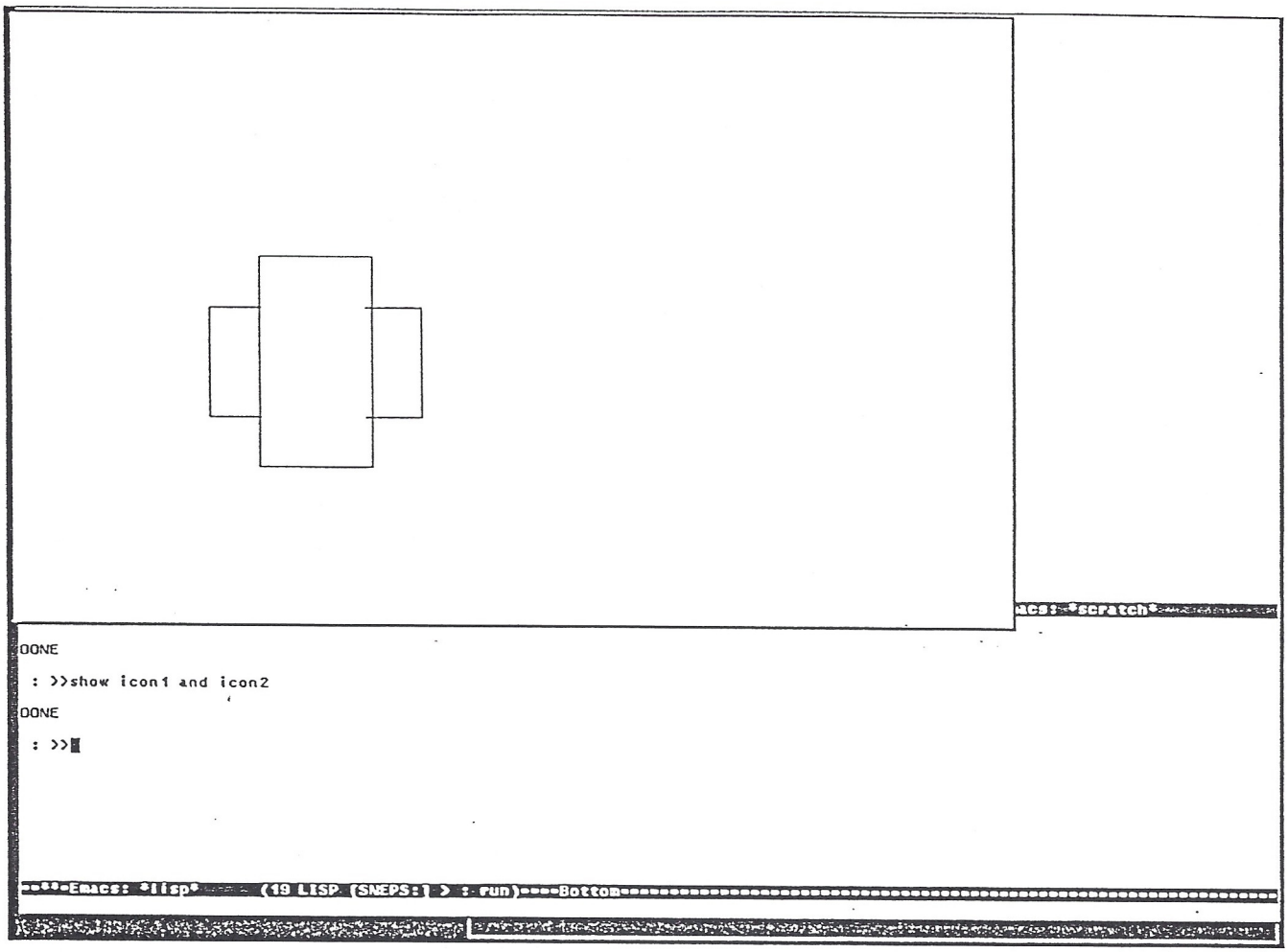

acs: *scratch*
DONE
: >>show icon1 and icon2
DONE
: >>
-**-Emacs: *lisp* (19 LISP (SNEPS:) > : run)----Bottom

Fig. 6.2.29

DONE

: >>clear the screen

DONE

: >>show icon2 and icon1

[*This demonstrates that the drawing order is independent of the request order. (Fig. 6.2.30)*]

DONE

: >>clear the screen

DONE

: >>the form of icon3 is form3

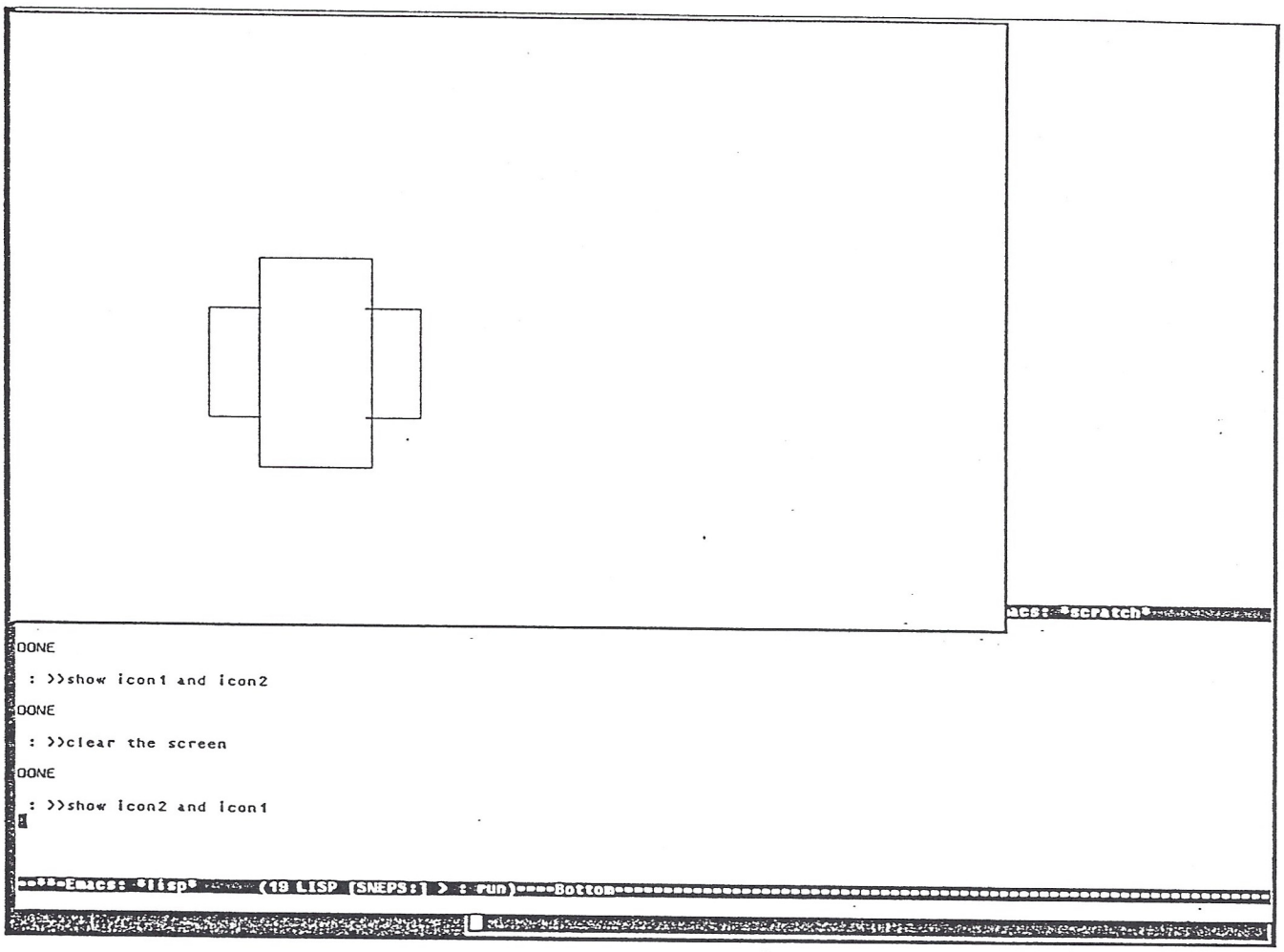

DONE
: >>show icon1 and icon2
DONE
: >>clear the screen
DONE
: >>show icon2 and icon1
(19 LISP (SNEPS:) > : run)
Bottom

Fig. 6.2.30

[*icon3 is a blue circle with white fill.*]

PARSED

: >>icon3 is in front of icon1

PARSED

: >>erase the screen

DONE

: >>show icon3 and icon1

[*The circle overdraws the vertical rectangle, creating the behind effect. (Fig. 6.2.31)*]

DONE

: >>erase the screen

DONE

: >>show icon1, icon2 and icon3

[*Icon3 overdraws icon1 which in turn overdraws icon2. The latter fact is visible dynamically, however does not come out in the final result. (Fig. 6.2.32).*]

DONE

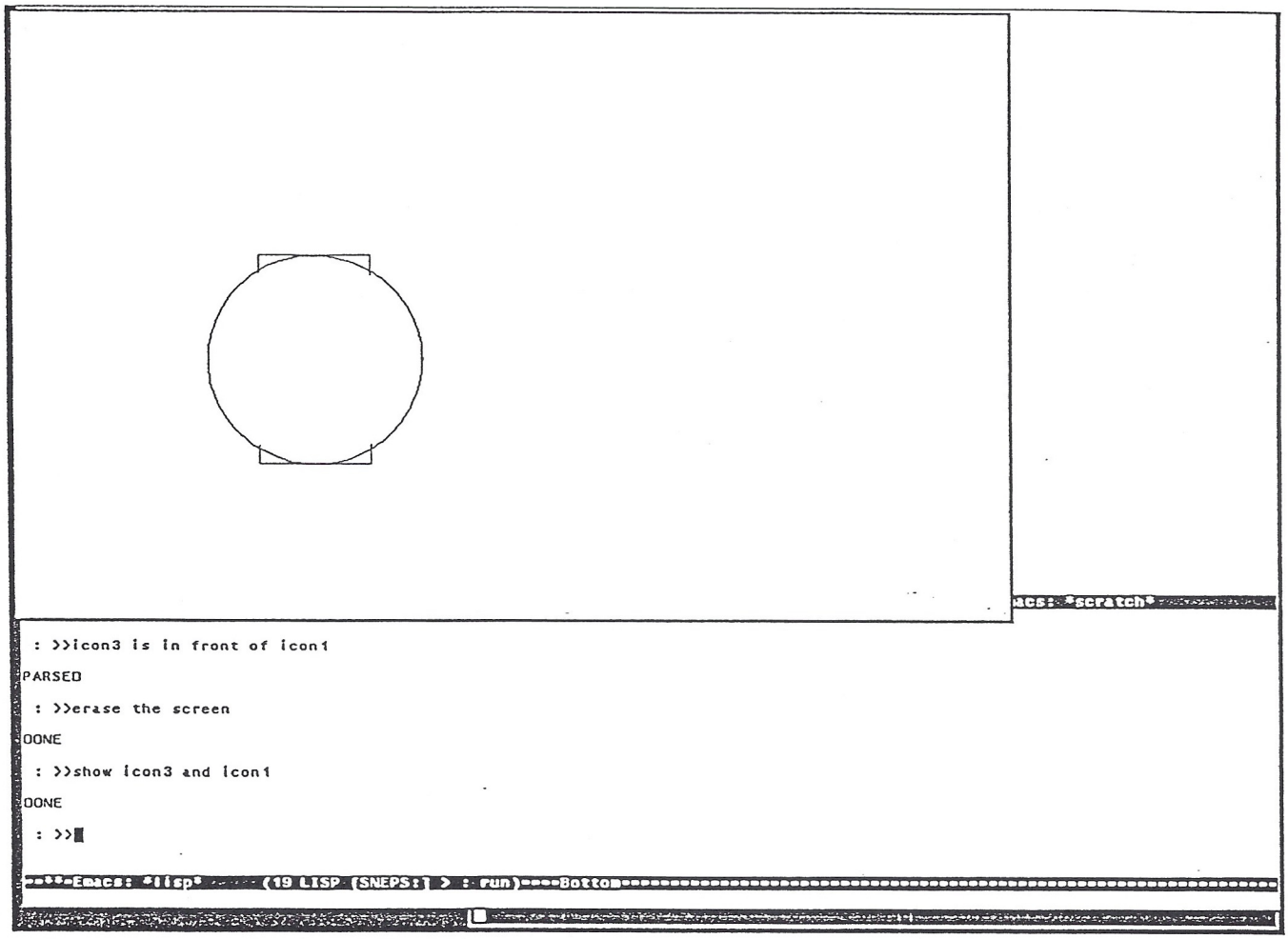


Fig. 6.2.31

Demonstration 7

[*This demonstration shows the use of clusters. A screen coordinate system with a top view is defined.*]

: >>the form of wood is wood-form

PARSED

: >>wood is at 2 6 inches on the screen

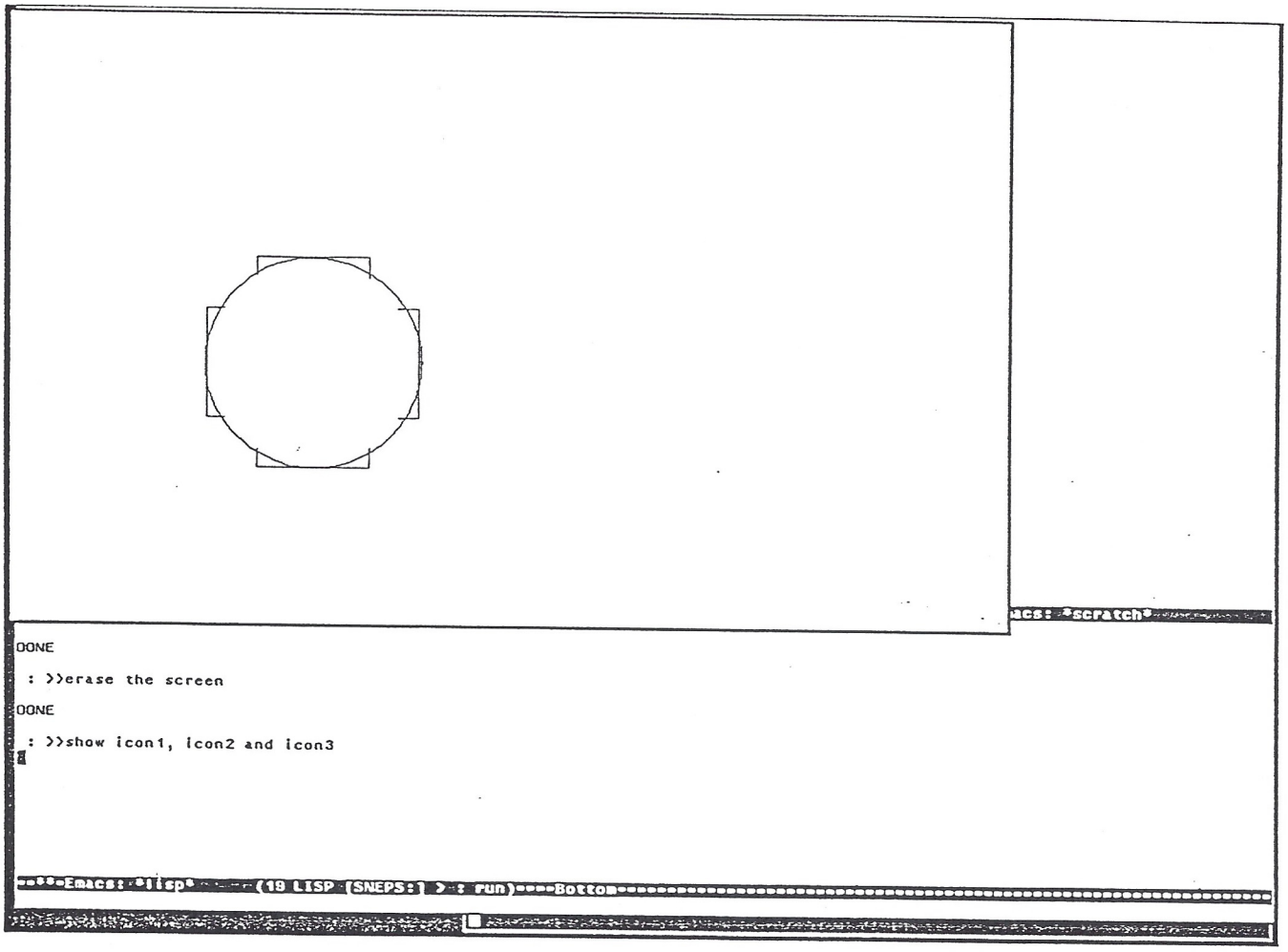


Fig. 6.2.32

PARSED

: > >clear the screen

DONE

: > >show wood

[*Fig. 6.2.33*]

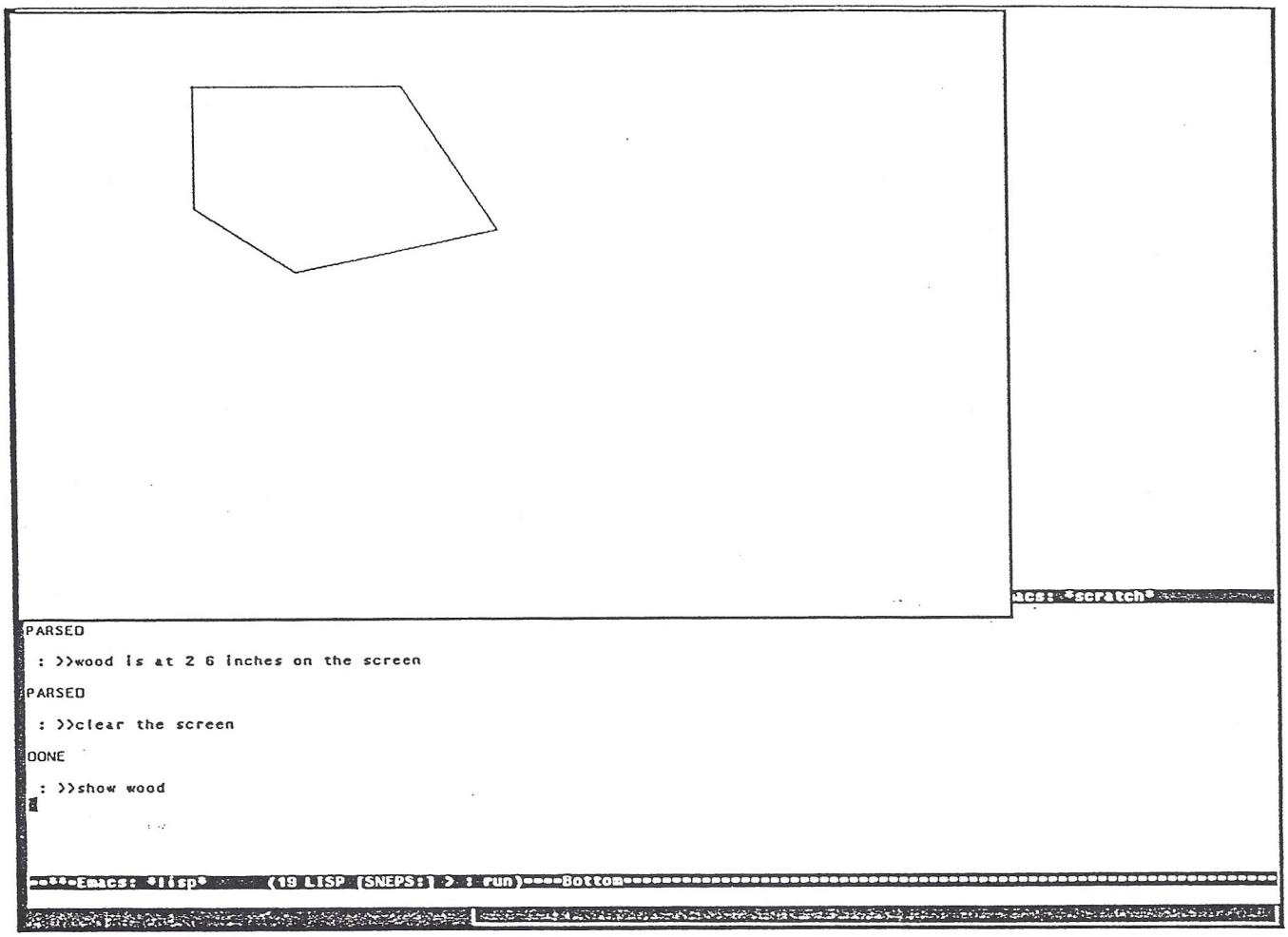


Fig. 6.2.33

DONE

: > >wood has tree1, tree2 and tree3 as sub-clusters

[*We limit ourselves to three trees to keep run times reasonable.*]

PARSED

: > >tree1 is at 30 -30 relative to wood

PARSED

: >>tree1, tree2, and tree3 are members of tree

PARSED

: >>tree2 is at 60 -30 relative to wood

PARSED

: >>the form of a tree is tree-form

PARSED

: >>tree3 is at 30 -60 relative to wood

PARSED

: >>clear the screen

DONE

: >>show us wood

[*Only the "map area" of wood is displayed. Adding information about trees has no effect. (Fig. 6.2.34)*]

DONE

: >>erase the screen

DONE

: >>show 2 levels of wood

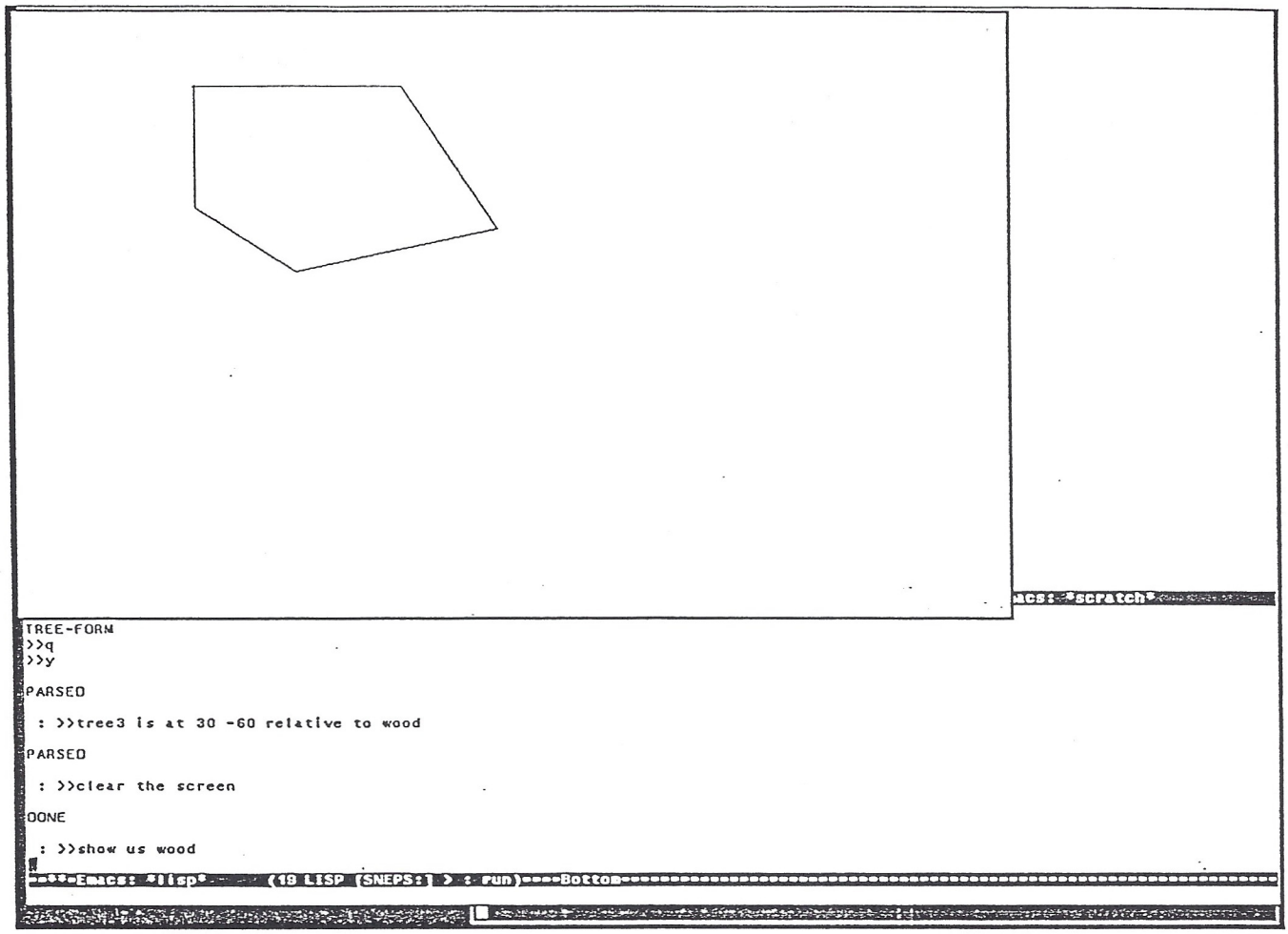

TREE-FORM
>>q
>>y
PARSED
: >>tree3 is at 30 -60 relative to wood
PARSED
: >>clear the screen
DONE
: >>show us wood
(18 LISP (SNEPS:1) > : run)----Bottom

Fig. 6.2.34

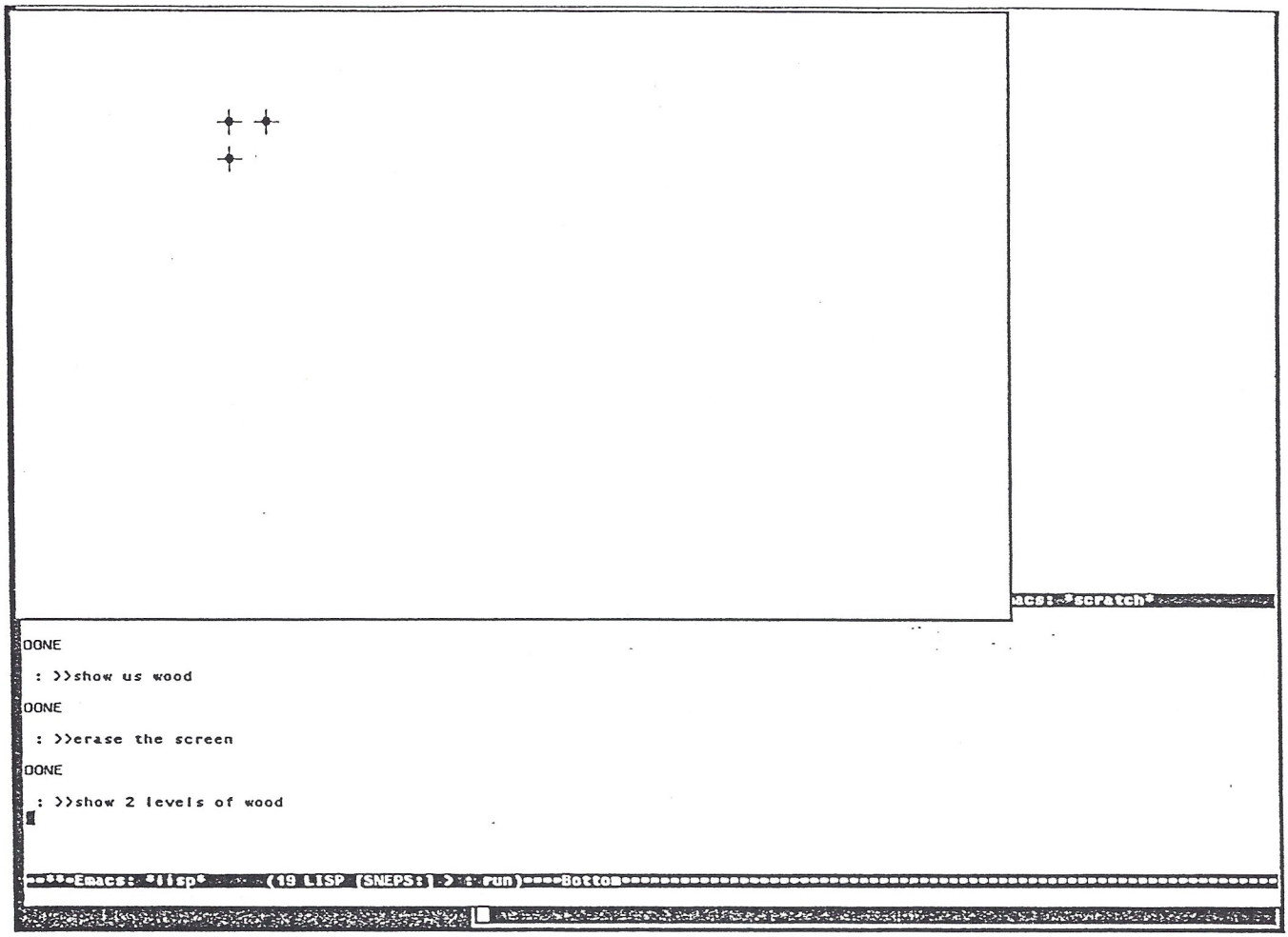

acs: *scratch*
DONE
: >>show us wood
DONE
: >>erase the screen
DONE
: >>show 2 levels of wood
-***-Emacs: *lisp* (19 LISP [SNEPS:] -> : run)----Bottom

Fig. 6.2.35

[*Now ONLY the trees are shown. The map area is not displayed. (Fig. 6.2.35).*]

DONE

: >>show wood

[*Note that NO clear command was given. Only the map area will be drawn, but the trees are already there from before, so Fig. 6.2.36 shows the combined picture.*]

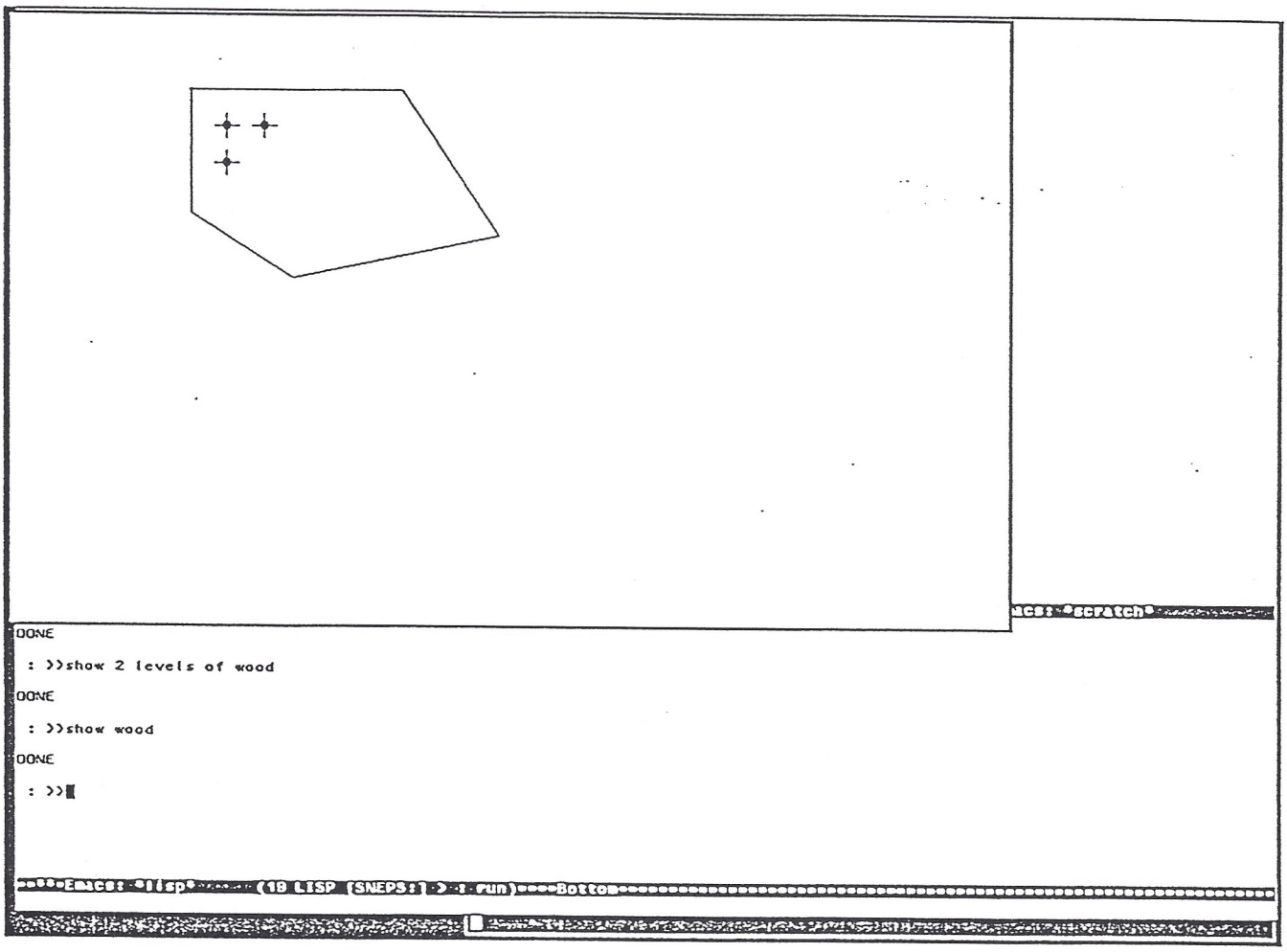


Fig. 6.2.36

Demonstration 8

[*The usual coordinate system definition is assumed. This is a small variation of demonstration 6, and it shows that it makes sense to postulate a view, even if only one plane coordinate system is used.*]

: >>the form of icon1 is form-1

[*Icon1 is a vertical blue rectangle filled white.*]

PARSED

: >>icon1 is at 200 300 on the screen

PARSED

: >>the form of icon2 is form-2

[*Icon2 is a horizontal blue rectangle.*]

PARSED

: >>icon2 is behind icon1

PARSED

: >>show icon1 and icon2

[*Icon2 is drawn first, which results in icon1 overdrawing it later, and this creates the behind effect. (Fig. 6.2.37)*]

DONE

: >>clear the screen

DONE

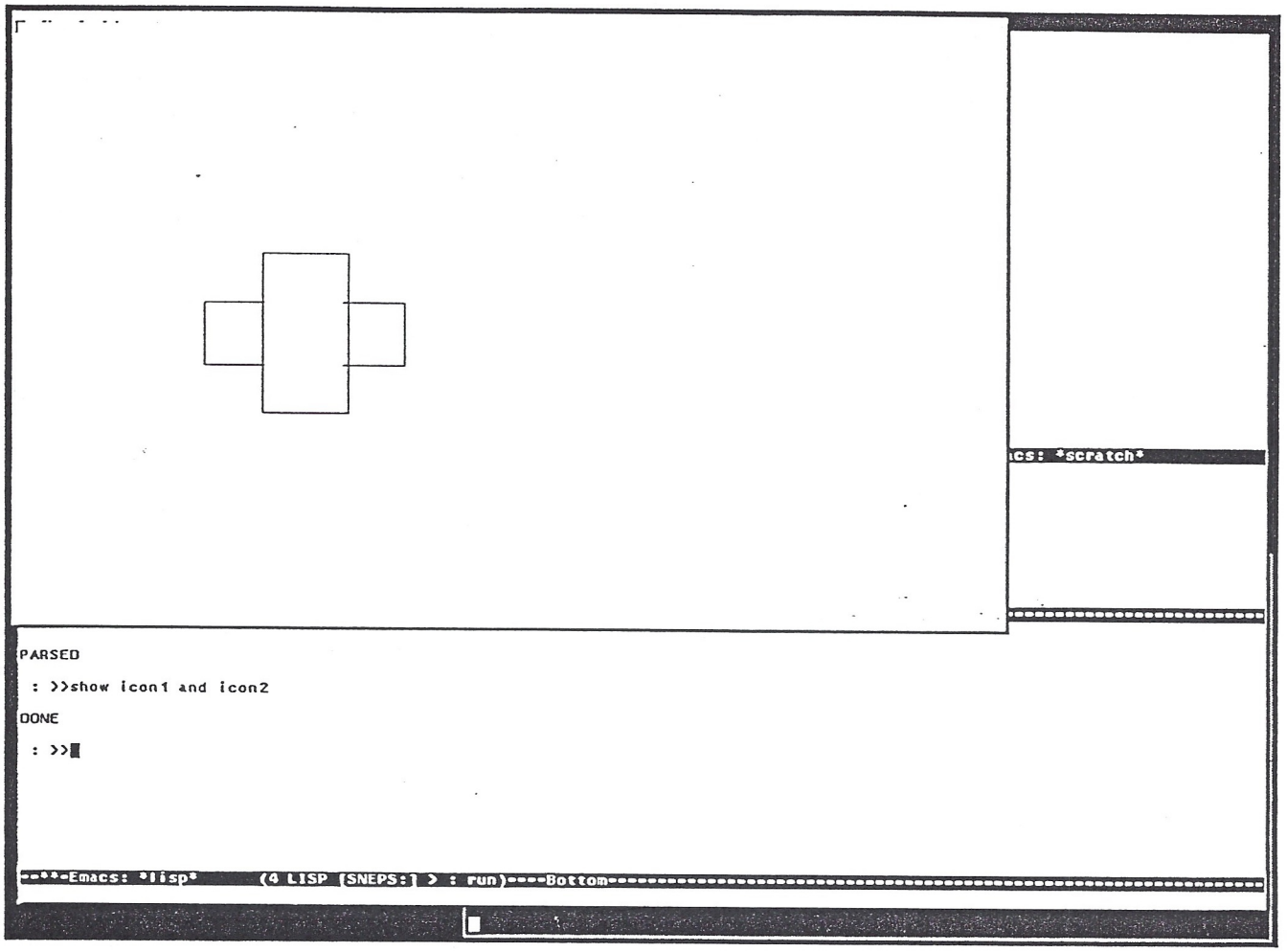


Fig. 6.2.37

: >>the assumed view of screen-coord-sys is top

(M4! VIEW (FRONT) COORD-SYS-P (SCREEN-COORD-SYS) LEFTNESS (USER))

node dismantled.

[*The previous view is erased from the network.*]

PARSED

: >>show icon1, icon2

[Now icon2 is displayed ABOVE icon1, because the user has expressed his desire to conceive of the figure as a top view, in which the term behind has to be interpreted differently. (Fig. 6.2.38)]

DONE

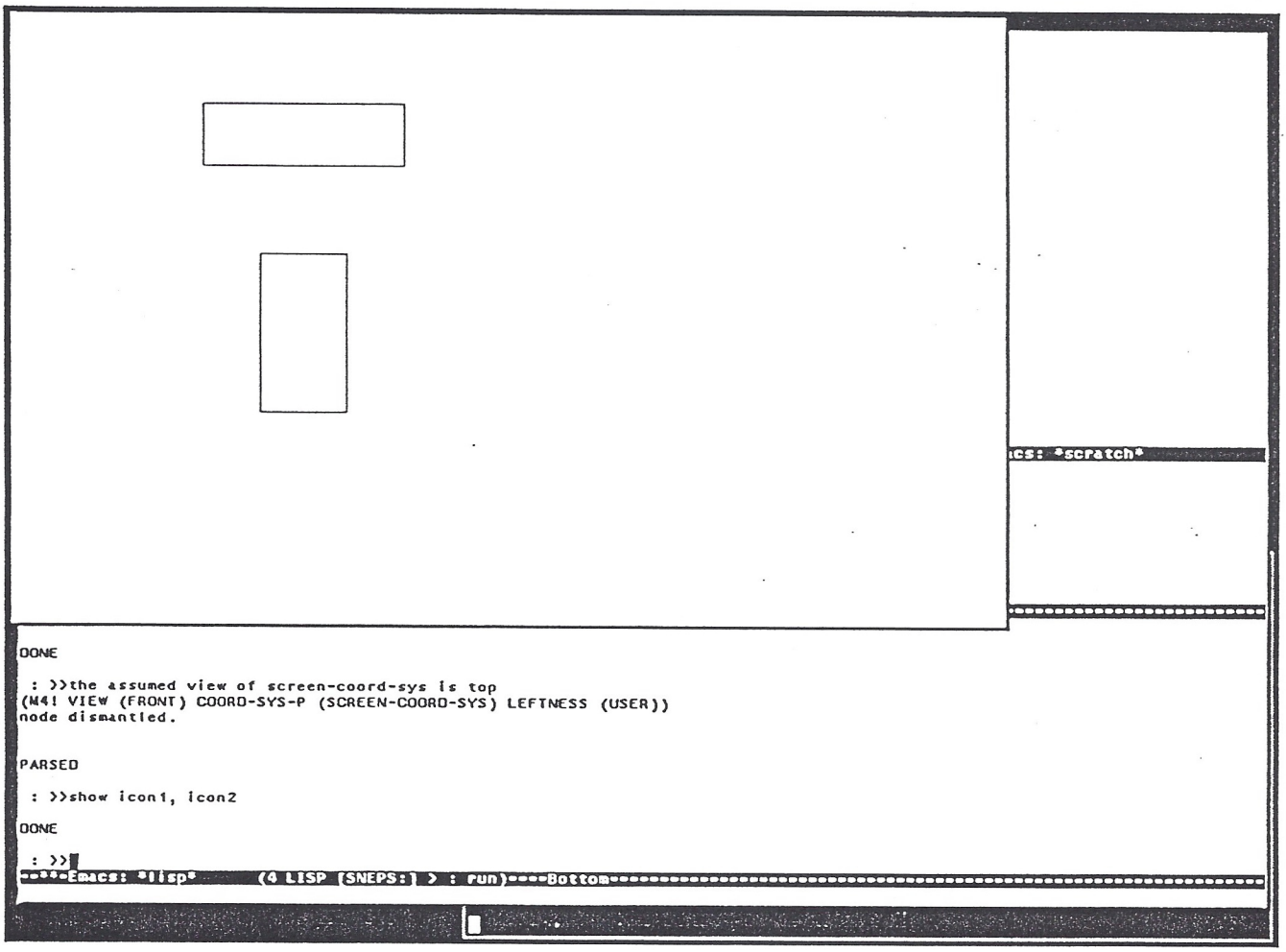


Fig. 6.2.38

Demonstration 9

[This demo requires not only a coordinate system, but also a few initial assertions that describe where "top", etc. are in the given window. It demonstrates the use of these "fuzzy absolute positions".]

: >>screen-coord-sys is a screen coordinate system with s-x s-y in screen

PARSED

: >>The assumed modality is function

PARSED

: >>one inch corresponds to 73 pixels

PARSED

: >>the view of screen-coord-sys is front

PARSED

: >>top is at 400 500 on the screen

PARSED

: >>center is at 400 250 on the screen

PARSED

: >>bottom is at 400 0 on the screen

PARSED

: >>left is at 0 250 on the screen

PARSED

: >>lower-left-corner is at 0 0 on the screen

PARSED

: >>the form of and1 is xand

[*Here the "interesting part" starts.*]

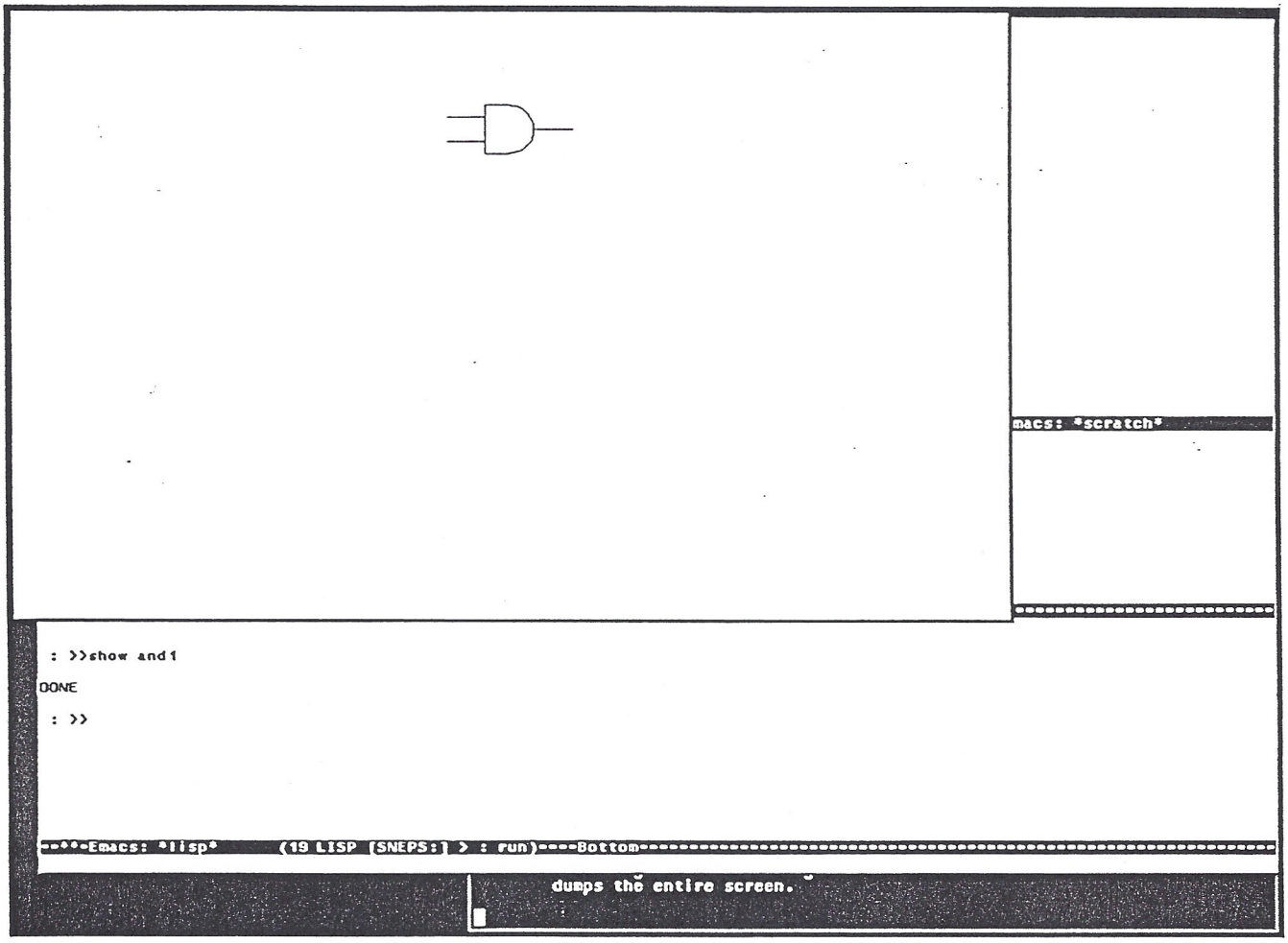


Fig. 6.2.39

PARSED

: >>and1 is at the top of the screen

PARSED

: >>show and1

[*Terms like "top" are interpreted as being near to the edge which in this case denotes the upper window border. Nearness is interpreted as dividing the given space by the extent of the object according to a ratio of 1:5. (Fig. 6.2.39)*]

DONE

: >>clear the screen

DONE

: >>the form of or1 is xor

PARSED

: >>or1 is at the lower-left-corner

PARSED

[*A corner position is interpreted as a 1:5 ratio in both directions. This is shown in Fig. 6.2.40*]

: >>show or1

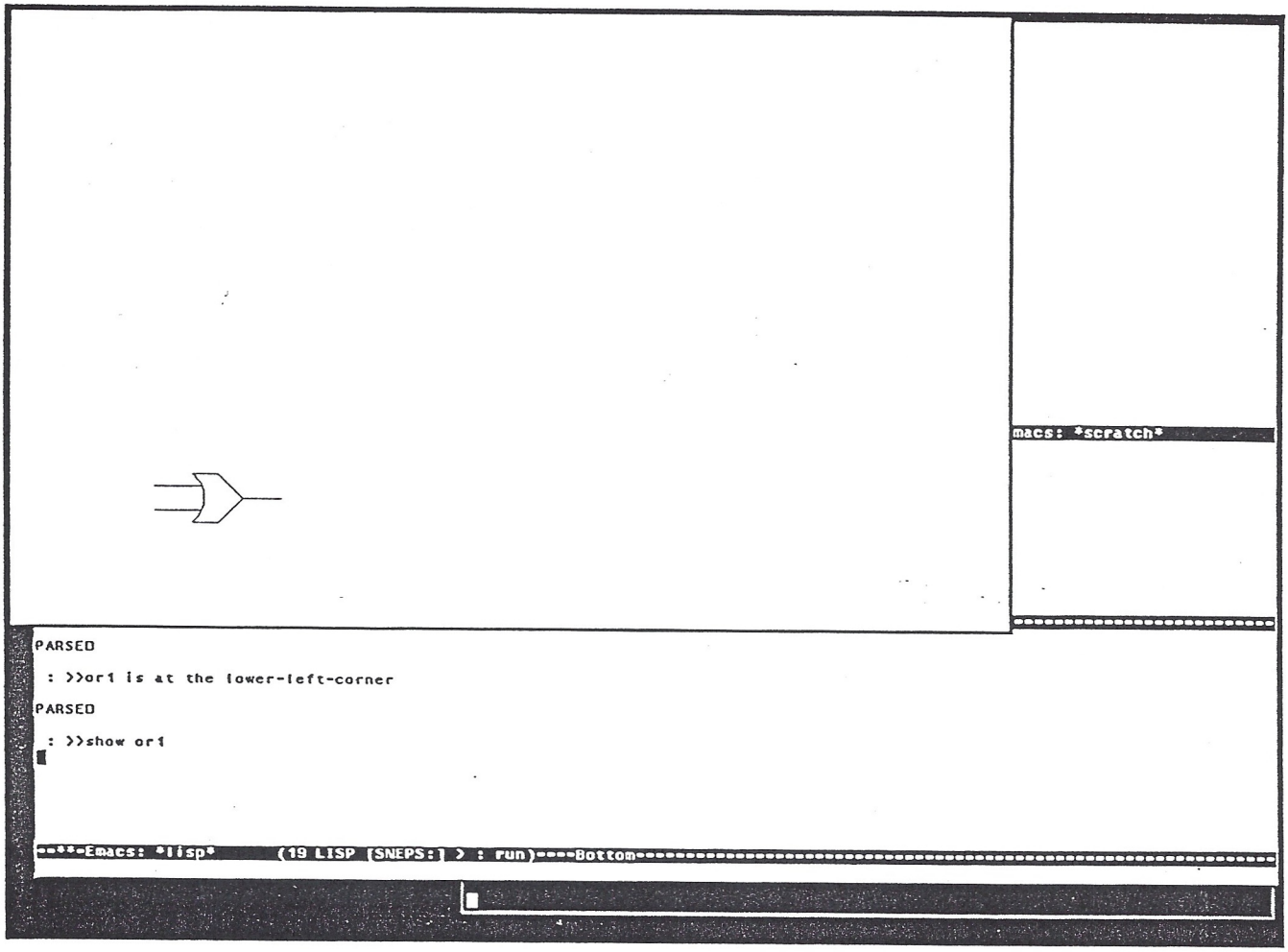


Fig. 6.2.40

Demonstration 10

[*This demonstration shows a special case of sub-clusters, namely sub-clusters that are at the same time real-parts of an object. The usual screen coordinate system exists.*]

: >>the form of board1 is xboard

PARSED

: >>board1 is at 2 6 inches on the screen

PARSED

: >>show board1

[*Fig. 6.2.41*]

PARSED

: >>the form of and1 and and2 is xand

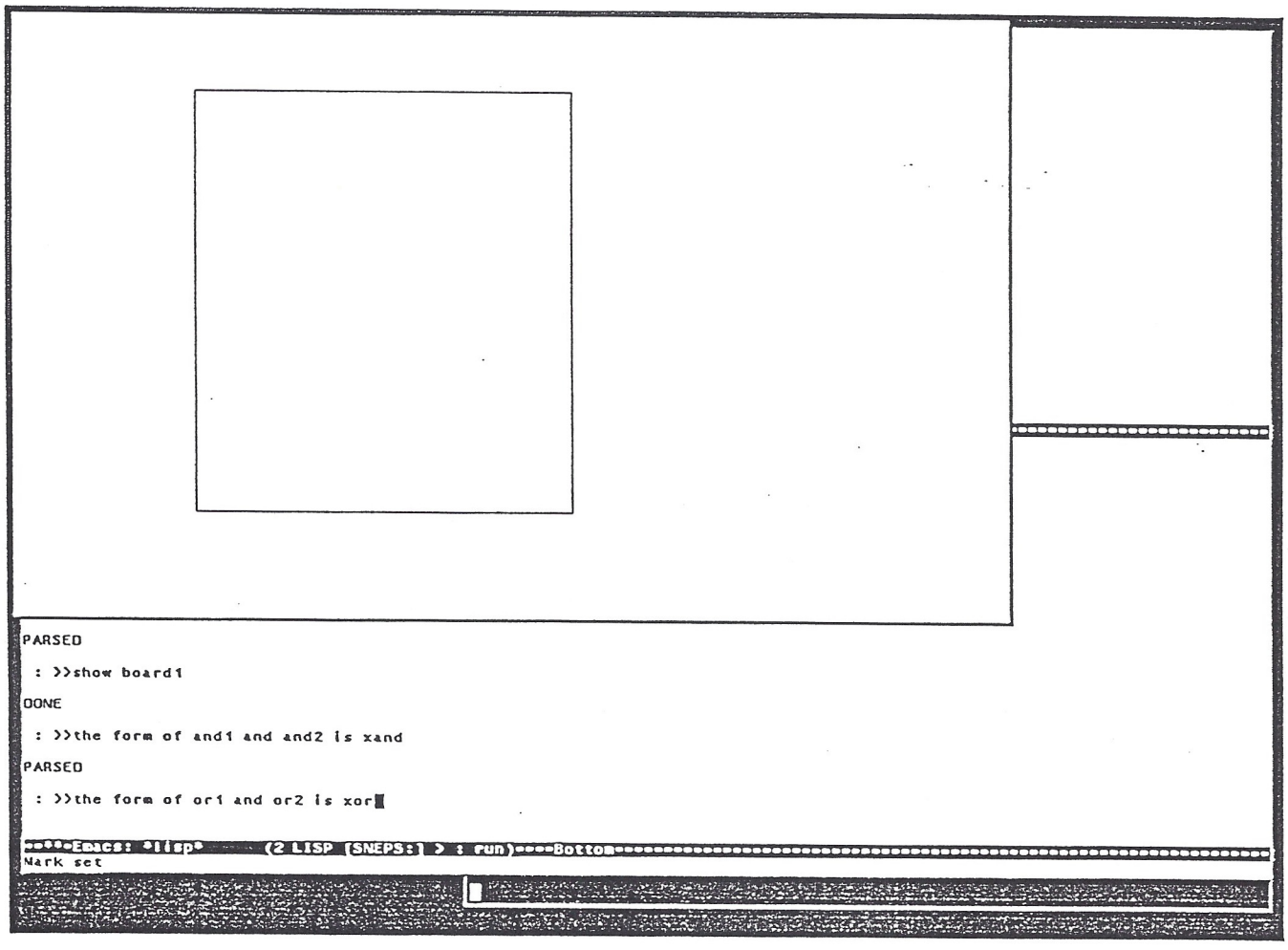


Fig. 6.2.41

PARSED

: >>the form of or1 and or2 is xor

PARSED

: >>and1 is here

(203 358)

PARSED

: >>and2 is here

(307 356)

PARSED

: >>or1 is here

(202 228)

PARSED

: >>or2 is here

[*Placing with "here" for an object of known form results in immediate display. (Fig. 6.2.42).*]

(309 227)

PARSED

: >>board1 has and1 and and2 as parts

PARSED

: >>board1 has or1 and or2 as parts

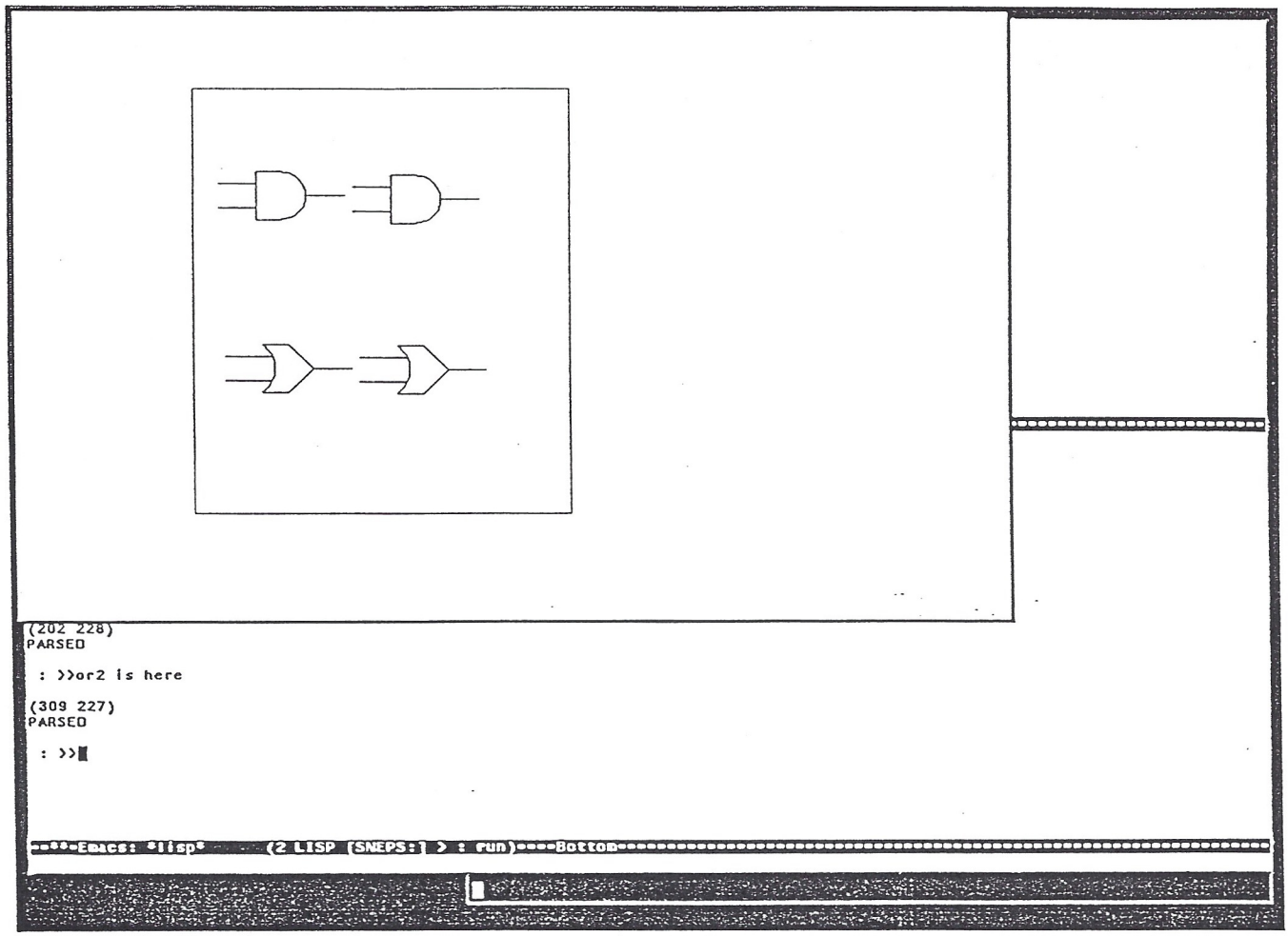


Fig. 6.2.42

PARSED

: >>the form of module1 and module2 is little-box

PARSED

: >>clear the screen

DONE

: >>show board1

[*The result of this display call adds nothing interesting. It is just needed for further placing. We omit the corresponding figure.*]

DONE

: >>module1 is here

(159 387)

PARSED

: >>module2 is here

(159 241)

PARSED

: >>module1 has and1 and and2 as sub-clusters

PARSED

: >>module2 has or1 and or2 as sub-clusters

PARSED

: >>clear the screen

DONE

: >>show 1 level of board1

[*Fig. 6.2.43*]

DONE

: >>show 2 levels of board1

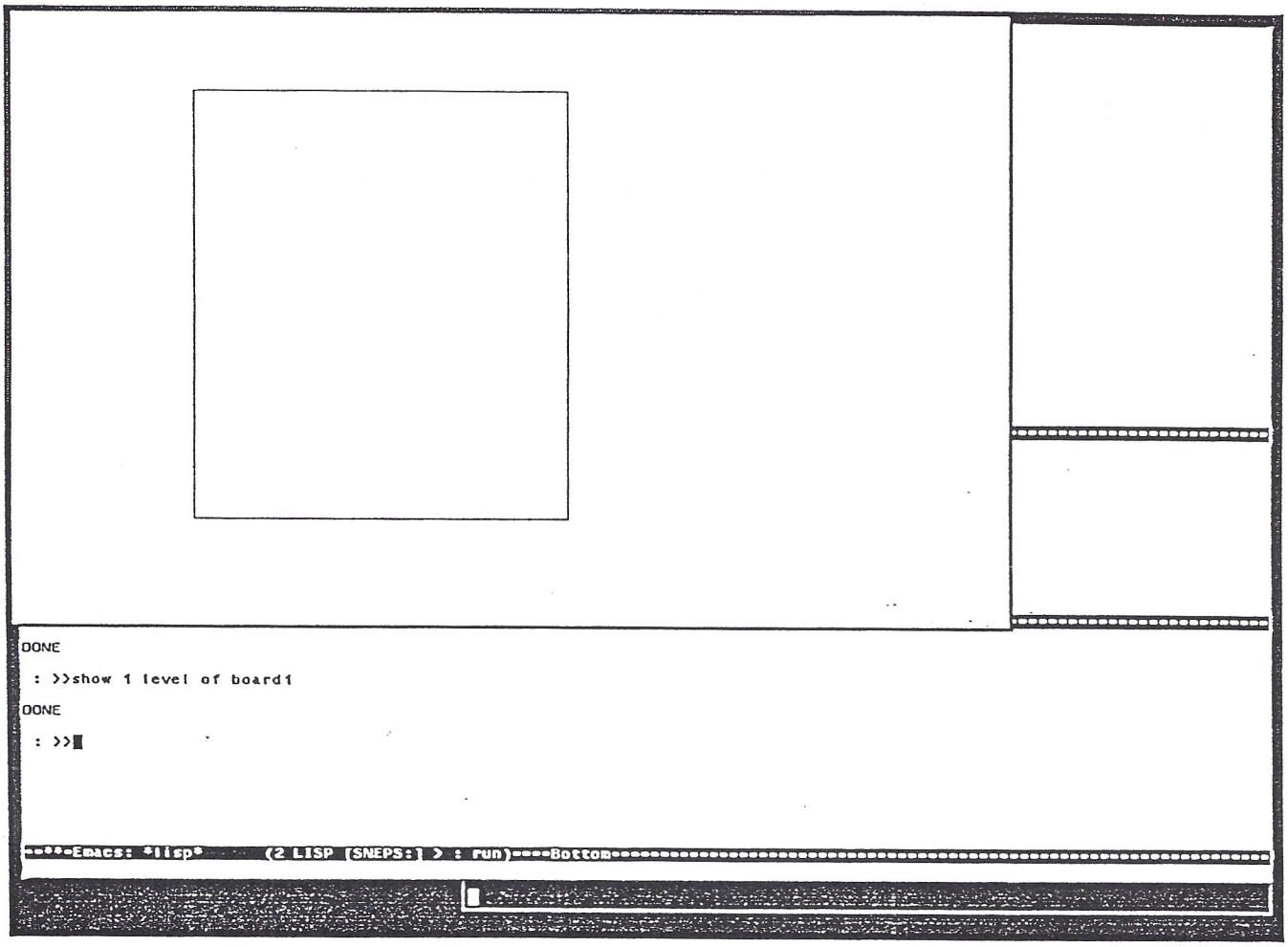


Fig. 6.2.43

[*Fig. 6.2.44*]

DONE

: >>clear the screen

DONE

: >>show 1.5 levels of board1

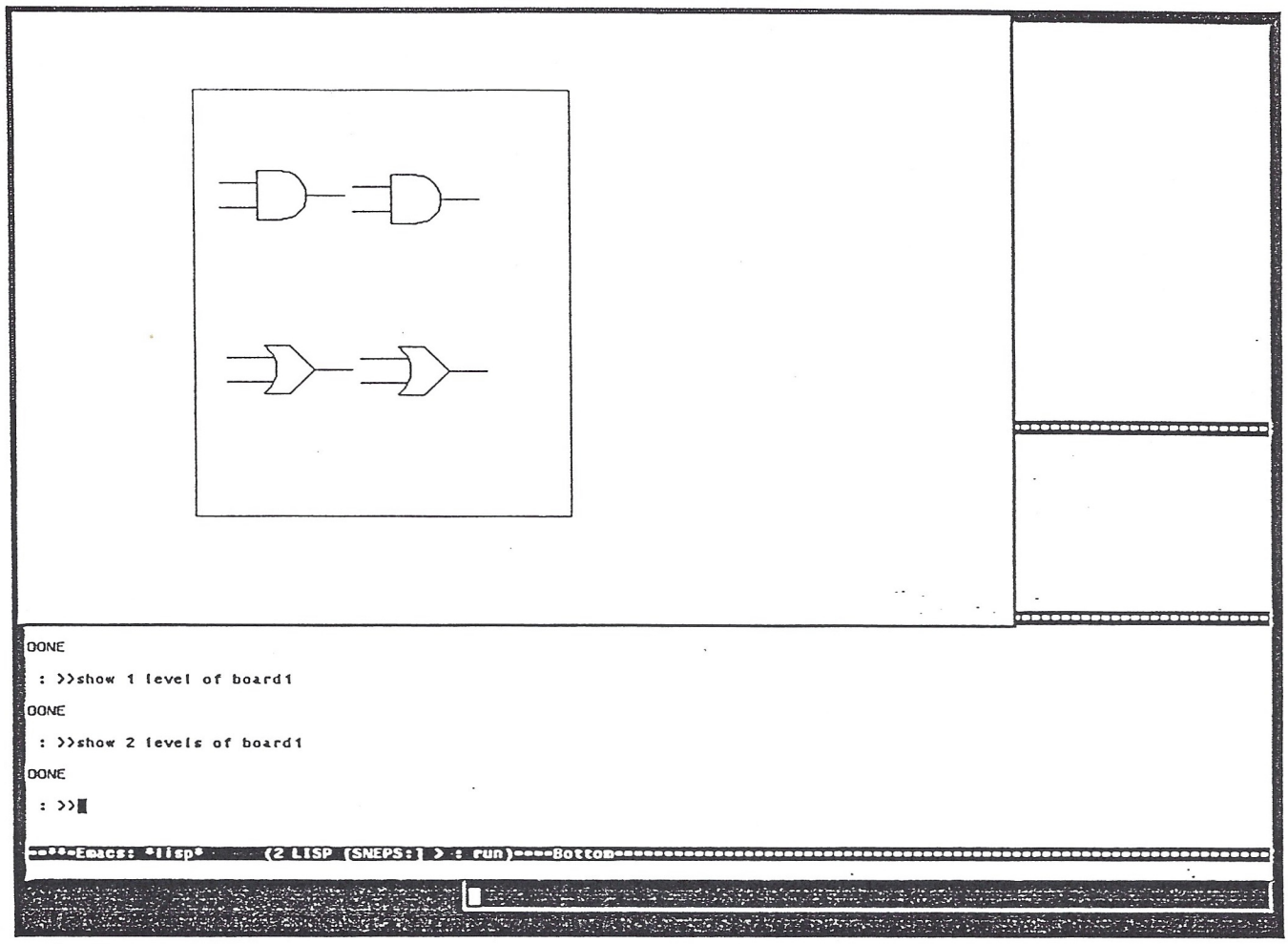


Fig. 6.2.44

[*This demonstrates cluster-summarization. If objects are at the same time real parts of an object A and form clusters of another object B, then one can display the the object A and summarize its sub-parts into clusters. The clusters are then virtual parts of the object. Summarization is activated by using a real number as level. (Fig. 6.2.45).*]

DONE

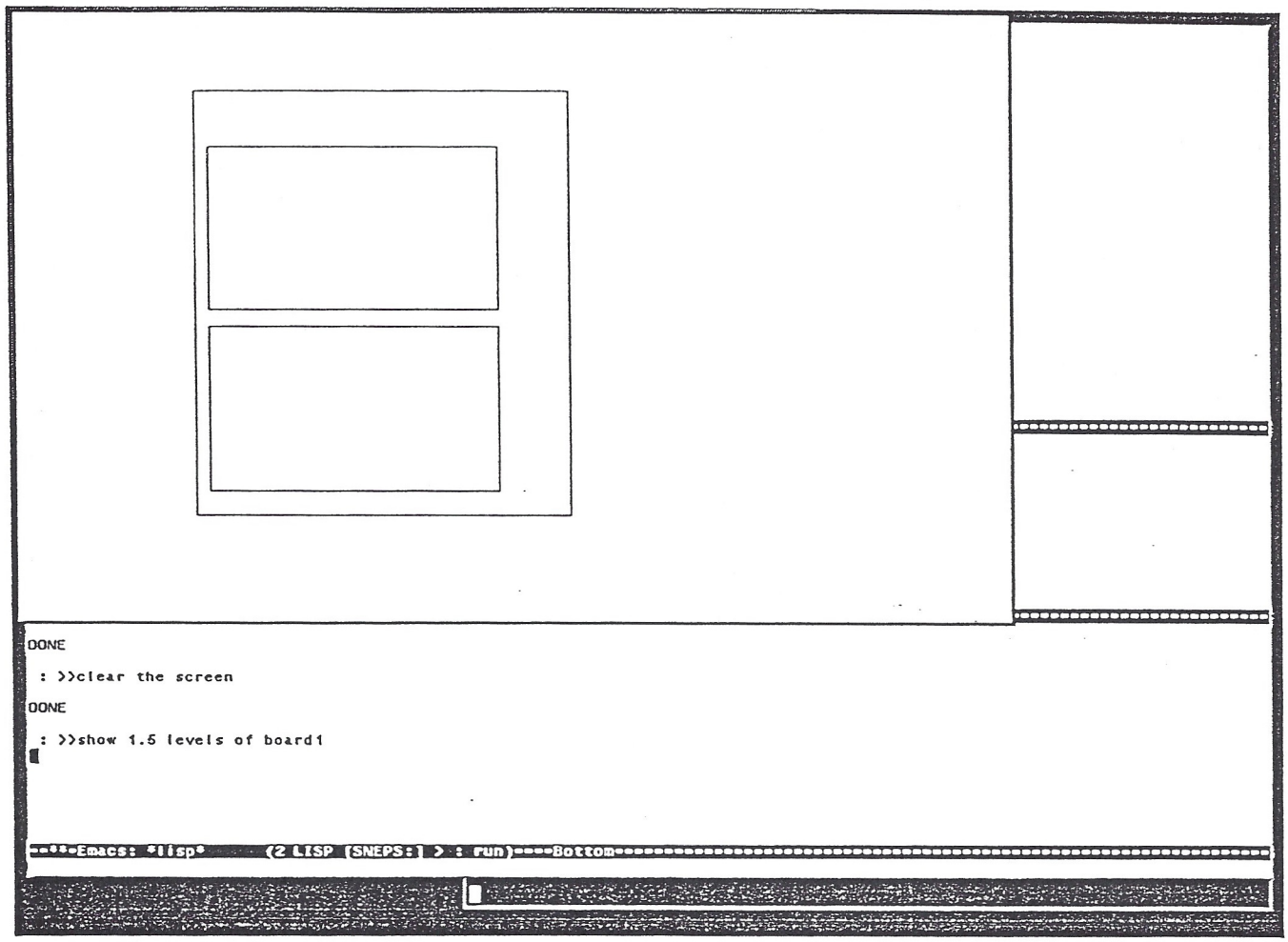


Fig. 6.2.45

Demonstration 11

[*This demonstration shows the use of a 3-d world coordinate system.*]

: >>myxyz is a world coordinate system with w-x w-y w-z in myworld

[*Note! This is different from the usual screen coordinate system!*]

PARSED

: >>The assumed modality is function

PARSED

: >>One inch corresponds to 73 pixels

PARSED

: >>screen-coord-sys is a screen coordinate system with s-x s-y in screen

PARSED

: >>The view of myxyz from screen-coord-sys is front

[*This view corresponds to an orthogonal coordinate projection.*]

PARSED

: >>chip-1 is at 200 200 0 in the coordinate system myxyz

PARSED

: >>the form of chip-1 is xand

PARSED

: >>board1 is at 100 400 10 in the coordinate system myxyz

PARSED

: >>the form of board1 is xxboard

[*This time board1 is filled white.*]

PARSED

: >>show board1 and chip-1

[*We assume a z axis pointing away from the viewer. Therefore both objects are now visible. (Fig. 6.2.46)*]

DONE

: >>the form of chip-2 is xor

PARSED

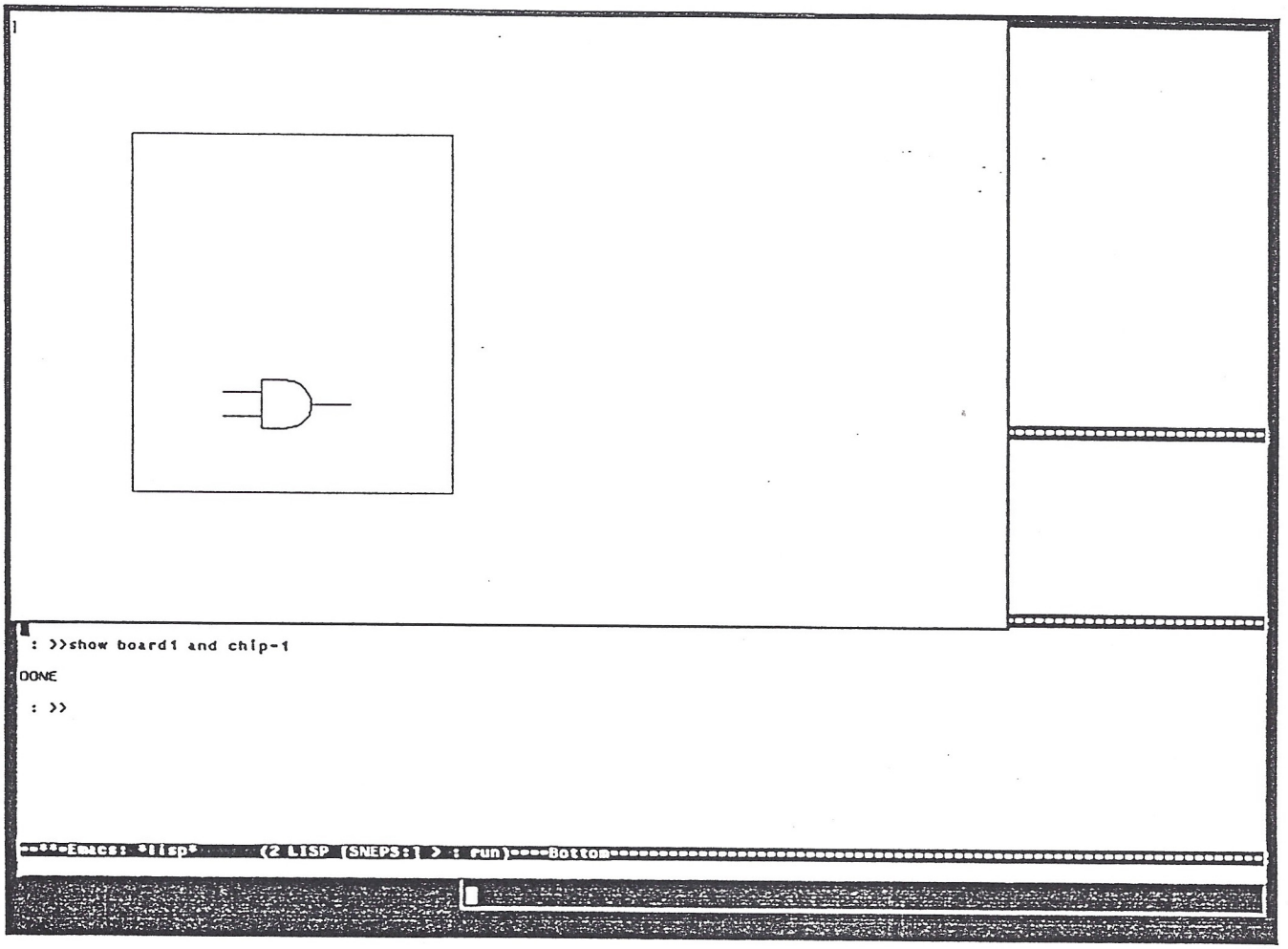


Fig. 6.2.46

: >>chip-2 is at 200 200 20 in the coordinate system myxyz

PARSED

: >>clear the screen

PARSED

: >>show chip-2

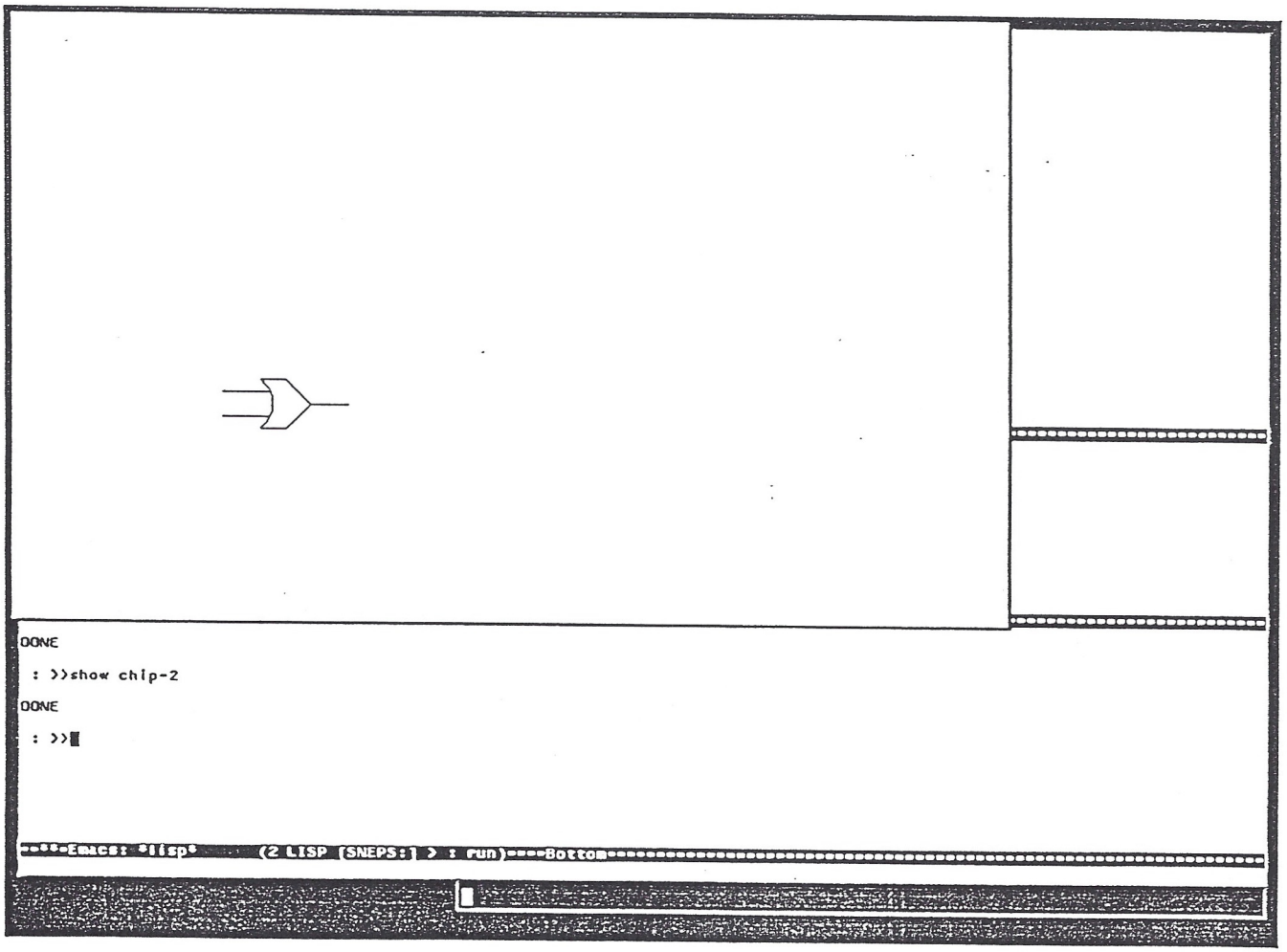


Fig. 6.2.47

[*chip-2 is at the same position as chip-1, but it is behind it. Fig. 6.2.47 shows chip-2.*]

DONE

: >>clear the screen

DONE

: >>show board1 and chip-2

[*Only the board is visible! The chip chip-2 is overdrawn because its z coordinate is higher than the z coordinate of the board. (Fig. 6.2.48)*]

DONE

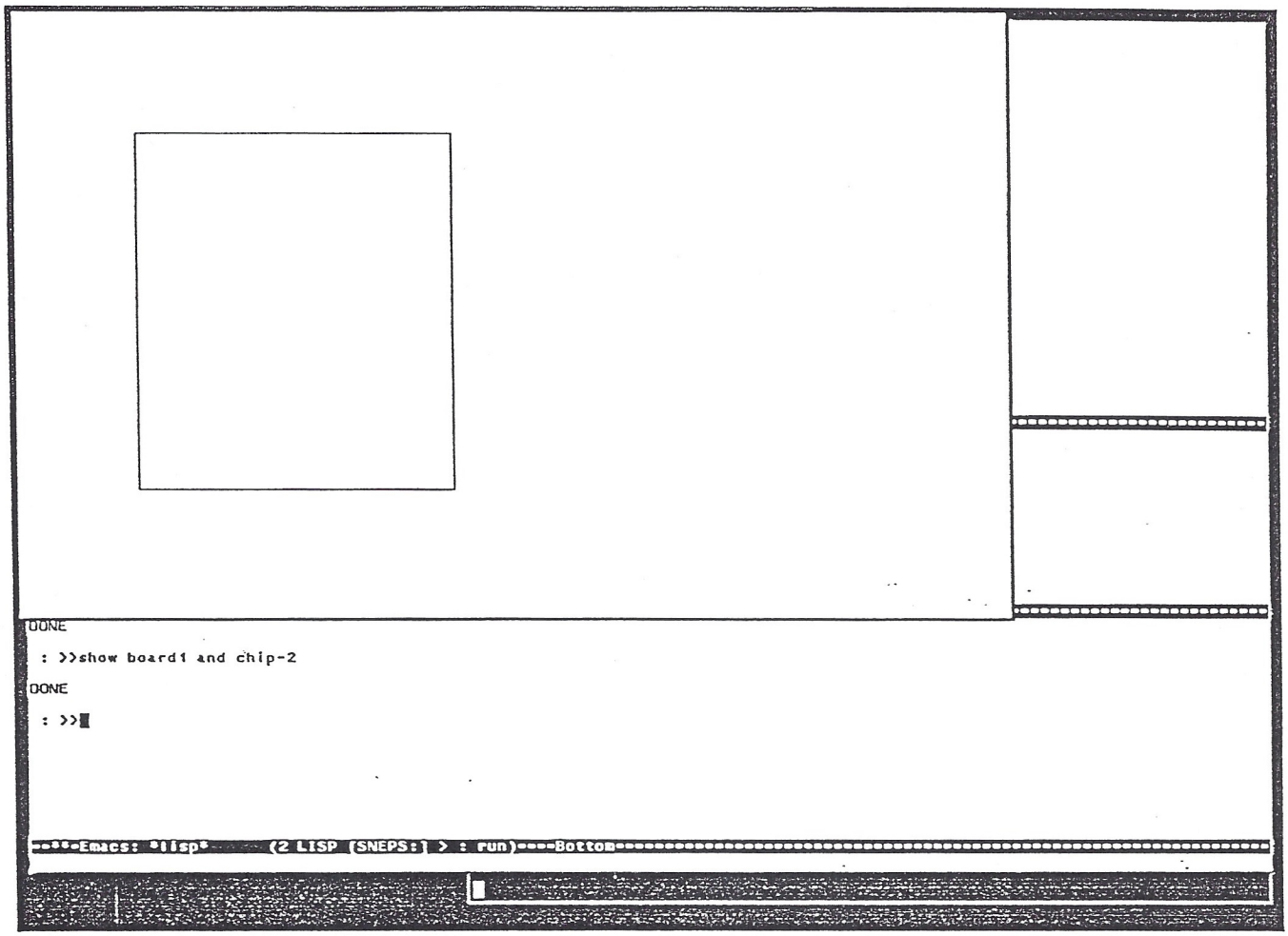


Fig. 6.2.48

Demonstration 12

[*This is back to the usual screen coordinates.*]

: >>and-1 is at 2 2 inches on the screen

PARSED

: >>the form of and-1 is xand

PARSED

: >>port-1, port-2 and port-3 are parts of and-1

PARSED

: >>port-1 and port-2 and port-3 are members of port

PARSED

: >>port-1 is at -30 -5 relative to and-1

PARSED

: >>port-2 is at -30 -25 relative to and-1

PARSED

: >>port-3 is at 60 -15 relative to and-1

PARSED

: >>the form of a port is xport

PARSED

: >>display 2 levels of and-1

[*Fig. 6.2.49*]

DONE

: >>the size of and-1 is large

PARSED

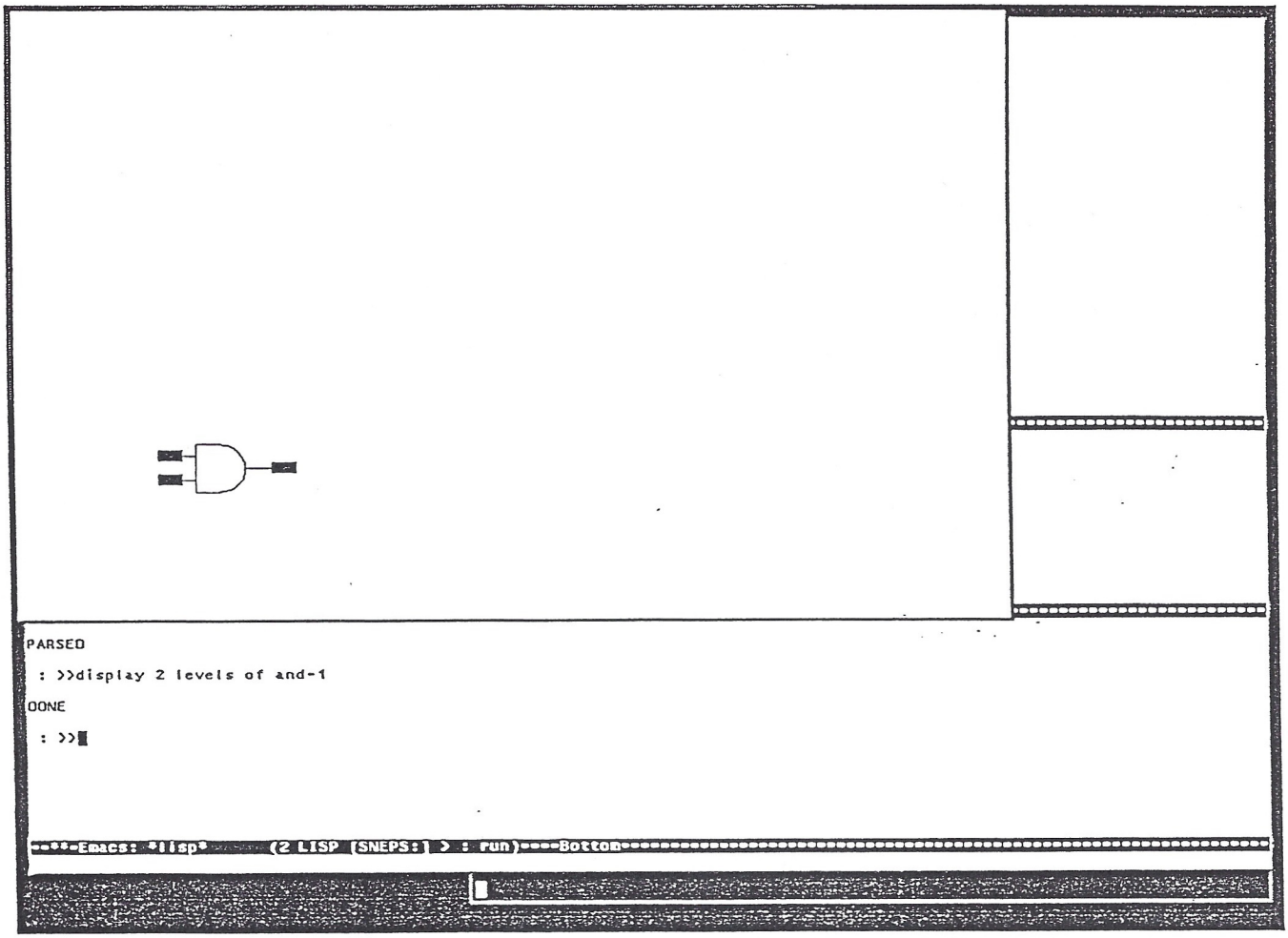


Fig. 6.2.49

: >>size is expressed by scalegr-jg and 2 represents large

PARSED

[*This is an attribute assignment, analogously to "state" in Demonstration 1.*]

: >>clear the screen

DONE

: >>show 2 levels of and-1

[*The gate has been scaled relative to the coordinate system. This results in making it larger, but also moving it away from its original position. But, note that the parts are where they were before! (Fig. 6.2.50). The system does not know that the "size" attribute is inheritable to parts.*]

DONE

: >>size is inheritable

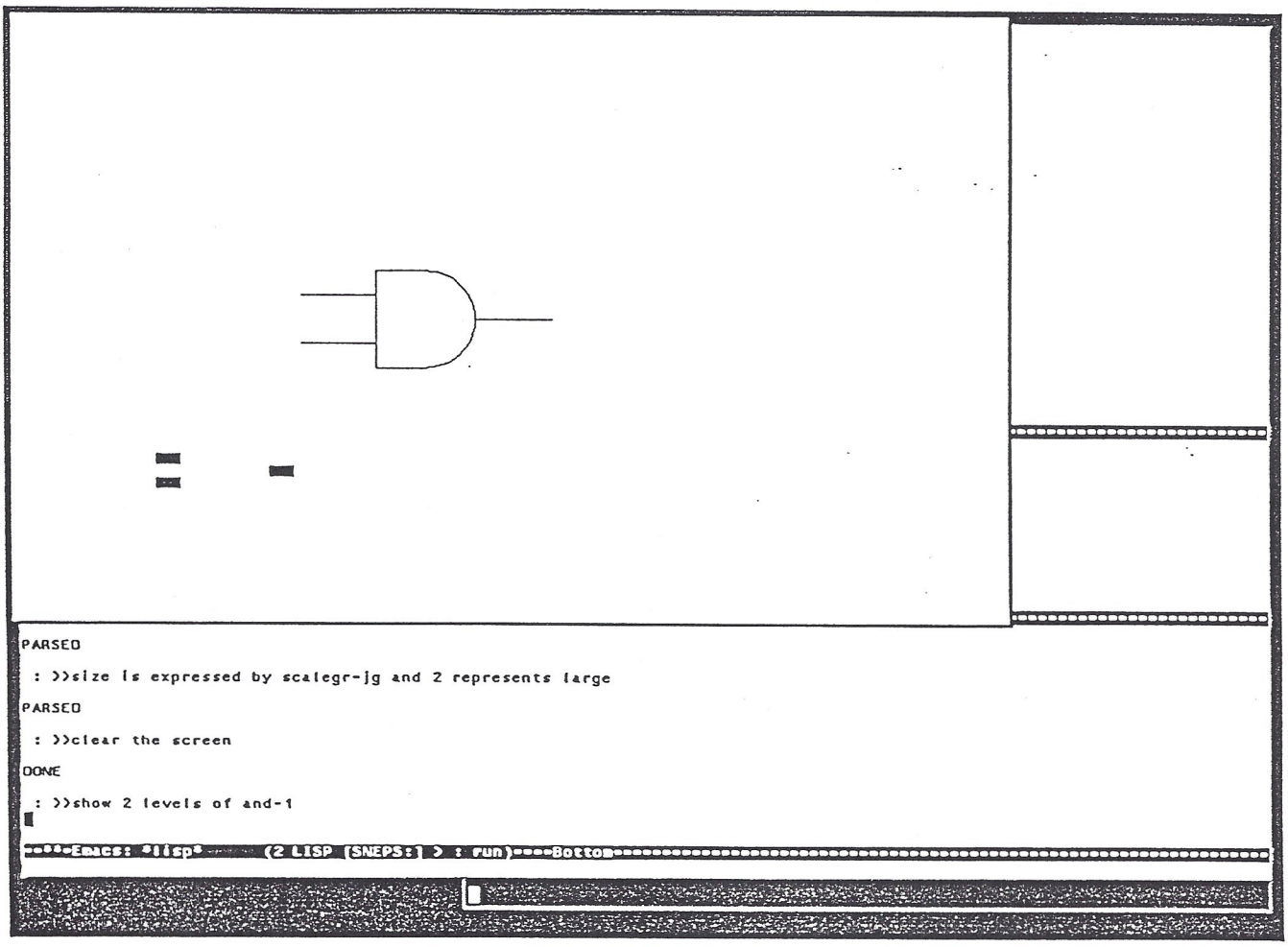


Fig. 6.2.50

[*Now the system does know that "size" is inheritable.*]

PARSED

: >>erase the screen

DONE

: >>show 2 levels of and-1

[*Now parts are correctly scaled, because it has been asserted that size is an inheritable attribute! (Fig. 6.2.51)*]

DONE

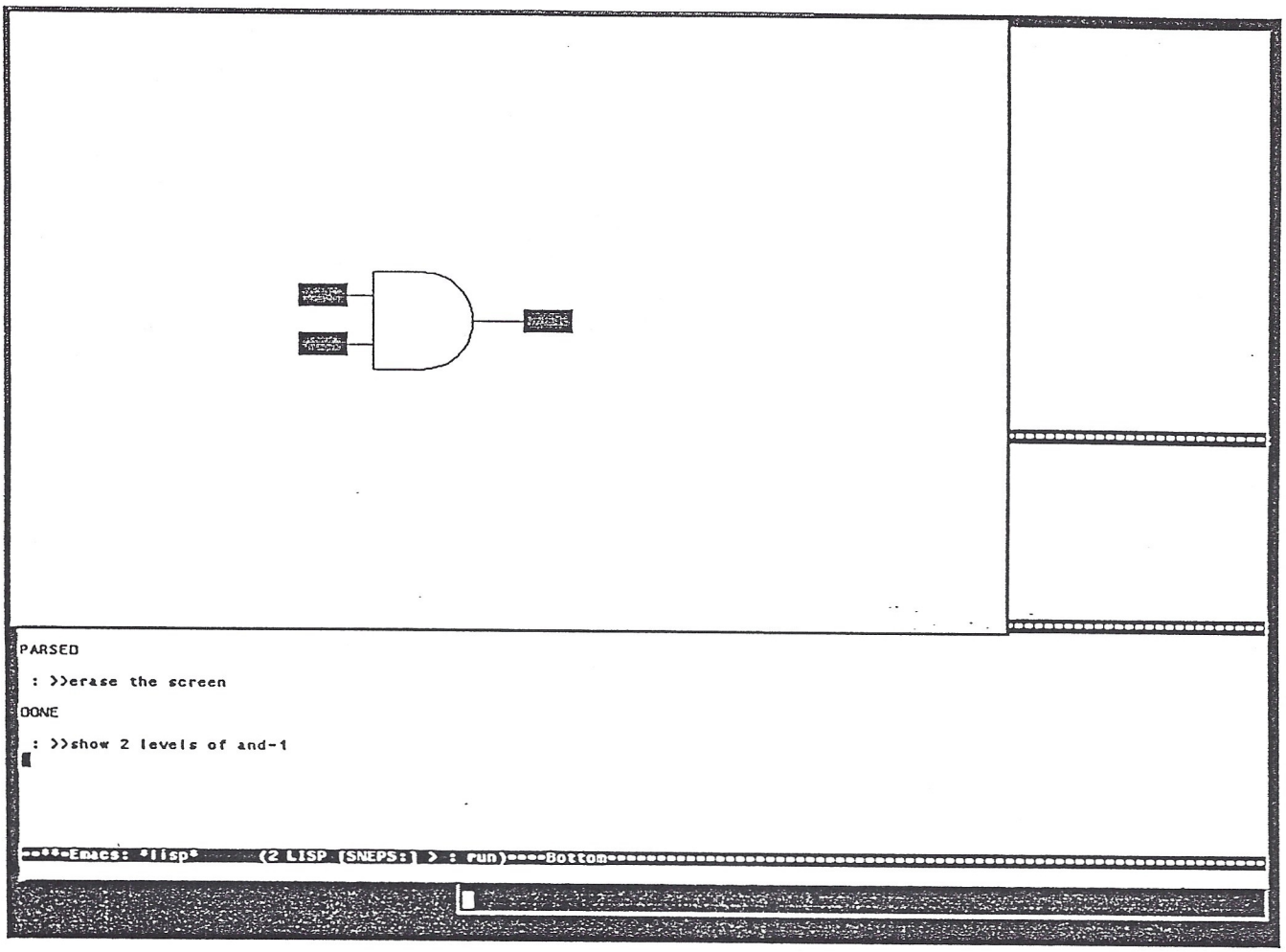


Fig. 6.2.51

6.3. Readform, the Graphics Editor

Readform is a program that permits a user to create the graphics code for an object by drawing the object. The result of a call to Readform in the VAX implementation is a new LISP function with two arguments, which, if applied to two coordinate values, will draw the original object in its original orientation, however with it's reference point at the position that is specified by the arguments of the function call. The reference point is specified by the user in the beginning of the object creation.

As pointed out repeatedly, any node in the network describing a form is at the same time the name of a LISP function. To do attribute modification it is necessary to recover the lambda expression containing calls to graphics primitives from this name. This can be done easily in Franz LISP, however, in the HP Common LISP implementation there is no operator to retrieve the lambda expression bound to an atom. It is possible to retrieve the function cell of a symbol, but the return value is a function descriptor only, not the actual lambda expression. Therefore it is necessary to store the graphics primitives invoked by a form function also in its value cell. In other words, the graphics primitives are stored in the value cell and the function cell of any symbol used as a form concept. This makes the HP implementation less elegant and more remote from the original theory.[2]

Readform works menu oriented and permits the user to create an object of lines, polygons, circles, disks, boxes, blocks (= filled boxes), arcs, and text. A number of auxiliary options permit storing and yanking of intermediate structures, duplication, etc. Fig. 6.3.1 shows the screen of the Readform main menu. The VAX version of Readform was implemented by Bill Eggers, an undergraduate at SUNY at Buffalo.[3] I ported VAX Readform to the X window environment and added all necessary graphics primitives.

6.4. TINA used as Maintenance Interface

The use of the TINA program as a graphics interface of the VMES project has been described in a number of earlier publications [GTS87, SST86, Tai87, TGS87]. Therefore only a short presentation will follow. The VMES system consists of a maintenance reasoner and a graphics interface. The graphics interface is an application of the TINA program described in this report. The task of the maintenance reasoner is to identify a faulty component in a given device, usually a circuit board. The maintenance reasoner and the display program share a knowledge

[2] There is reason to believe that the original theory of forms and attributes would never have been developed under the limitations of HP Common LISP. It seems that Sapir and Whorf were right after all.

[3] Thanks to Bill who has devoted much time to make the program useful and (in his own words) "idiot safe".

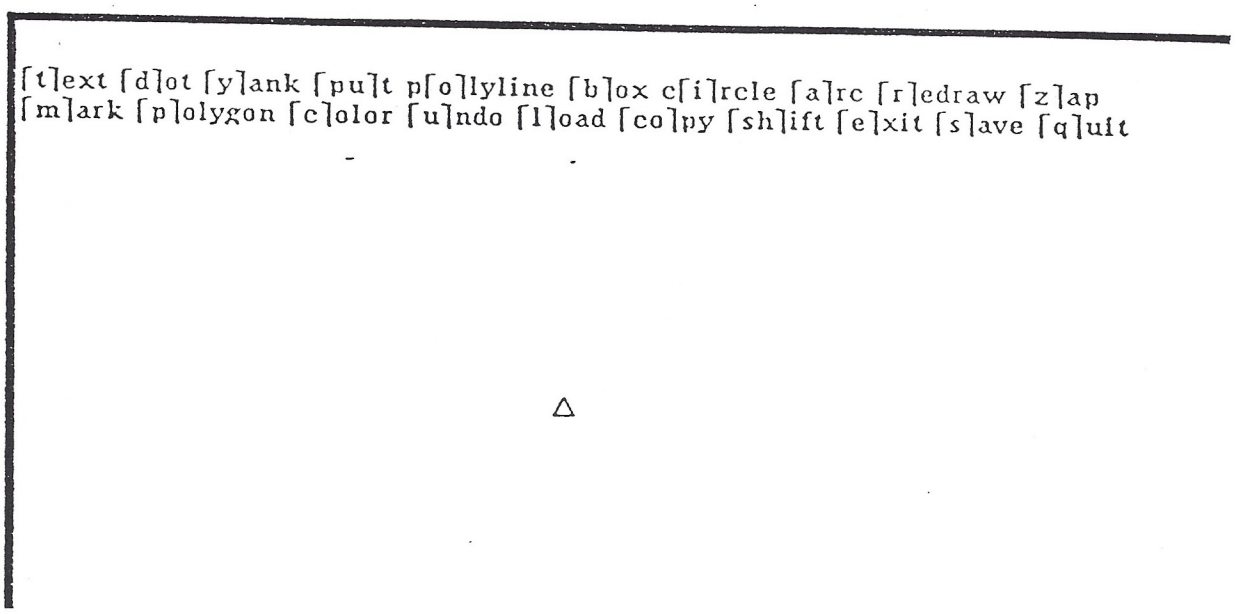


Fig. 6.3.1: The readform menu.

base realized as a SNePS network.

During the process of identifying a faulty component in a device, the maintenance reasoner repreatedly updates the shared knowledge base. It categorizes components as being in a "default state", being in a state of violated expectation, being recognized faulty or being suspected to be faulty. Information about any of these states is asserted in the network, using the attribute case frame described earlier on (Section 5.1.3.3.2). Whenever the maintenance reasoner wants to express changes in its state of knowledge about the analyzed device, it executes a call to TINA. TINA presents the current state of the maintenance process to the user. This is done by mapping attributes into signal colors (red = faulty, blue = default, green = suspect, magenta = violated expectation).

Typically a device will be displayed completely blue in the beginning. After finding a violated expectation, for instance a port that has a wrong voltage value, this port will receive an attribute "violated expectation". The device will now be blue, except for the port in question which will be magenta. The maintenance reasoner then does a path analysis and finds all the com-

ponents that could be responsible for the violated expectation. Every one of these components is asserted as being "suspect", and redisplaying the device will show these components in green. Finally, by using additional information supplied by the user and additional reasoning, one or more components will be identified as faulty and asserted as such. This results in a display of the device in blue with the faulty component(s) in red. This display is the final result of the maintenance run. By using the :environ option of TINA, it is possible to give the user a better idea what the current focus object of the maintenance reasoner is, and where this focus object is located in the overall diagram of the device.

The procedural interface between maintenance reasoner and display program is extremely narrow, it consists of one single function only, which is of course TINA. All other communication is done through the shared knowledge base that both parts of the program have access to. Practical experience with this type of programming has shown that it is exceedingly easy to combine two independent modules. The maintenance reasoner and the graphics interface were developed independently, with no more than an agreement about shared case frames. Nevertheless, the two parts of the program were working together immediately, without requiring any integratory debugging.

The most complicated device that was "maintained" with the combined maintenance reasoner/ graphics interface has been a 6 channel PCM board. In Fig. 6.4.1 a screen dump from a GIGI terminal is shown. The PCM board does analog/digital coding and decoding, and its main components are inverters, transformers, and one PCM chip per channel. The large number of components and the limited quality of the involved hardware resulted in somewhat fuzzy icons for most of the components.

6.5. Limitations of the Implementation

In AI the theory is always one step ahead of the implementation, and we will use this section to indicate parts of the theory that have not been implemented. (1) "User leftness" versus "com-

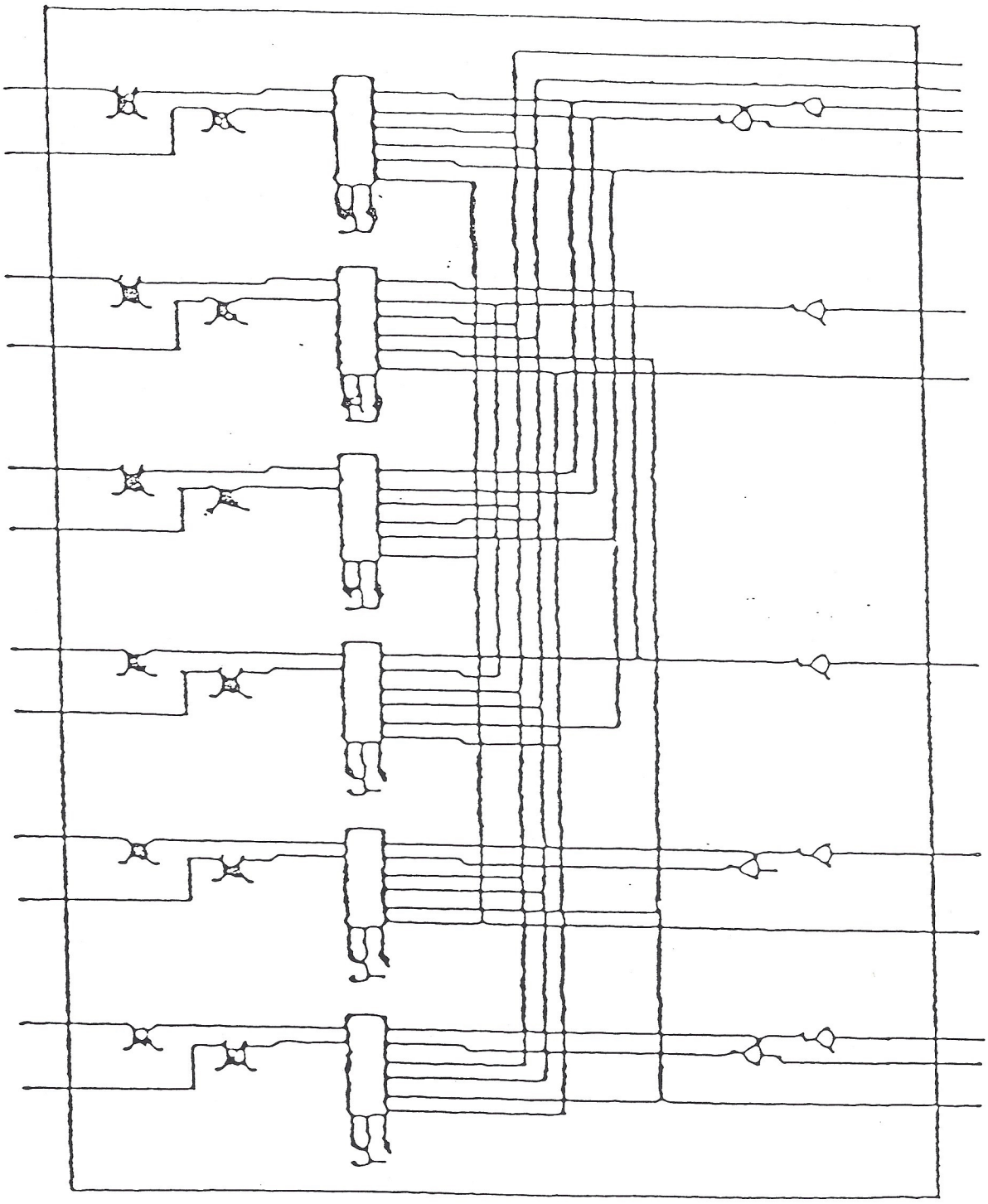


Fig. 6.4.1: A screen dump of the PCM board.

puter leftness" has not been implemented. The necessary structures are maintained though, and the necessary change to TINA follows closely the coding of "front" versus "top" view. (2) No

"current coordinate system" is maintained. The implementation of this feature would follow closely the "current attribute mapping" which is implemented. Only Cartesian coordinates are possible. However, "nearness" can still be expressed as described, substituting one of the Cartesian axes for the R axis. (3) For attributes only one case frame with one argument position is used; no differentiation between absolute and relative attributes has been implemented. The program nevertheless emulates the required behavior in most cases, because of the way the attribute mappings have been implemented. Attributes apply to objects only. Superlatives and comparatives are not implemented. (4) The discrimination between plane and screen coordinates is done by using an older representation for screen coordinates, and current screen coordinates for plane coordinates. Due to the fact that screen coordinates are system maintained and always concrete absolute coordinates, and because the user has no direct access to them, this incompleteness is completely transparent. (5) The representation of chains of fuzzy positions is certainly possible, but the graphical realization does not permit a number of cases and does not do a complete check for available space. Especially obstacles are not recognized. (6) Periodic structures are not implemented. (7) The natural language interface is neither general nor robust.

CHAPTER 7

INTELLIGENT MACHINE DRAFTING

In this chapter we will present a more specialized application of the GDK theory that has grown out of our work on circuit boards. The problem to be solved is the modeling of the behavior of a draftsman. This problem was introduced in [ShG86a] and [GeS87] and named the "Intelligent Machine Drafting Problem". Before explaining problem and solution in detail we will introduce a few non-GDK knowledge structures that will become necessary later on.

7.1. The Representations of Ports and Connections

In the domain of circuit board maintenance electrical connections between wires and components are of special importance. The representation for ports and connections has been borrowed from a companion dissertation [Tai87]. Every port is either an inport, or an outport, or a biport. The signal flow through a device can be followed by entering the device at one of its inports and following it to a component inport and leaving the component through an outport, etc. till finally a device outport is found. Biports are ports of wires. They have received their name, because signals might flow through them in two different directions.

Syntax:

object	inport-of	<device>	(7.1.1)
	id	<port-name>	
modality	<modality>		
type	PORT		
object	outport-of	<device>	(7.1.2)
	id	<port-name>	
modality	<modality>		
type	PORT		
object	biport-of	<wire>	(7.1.3)
	id	<id-number>	

modality	<modality>
type	PORT

Concerning the semantics of these structures for maintenance purposes, refer to [Tai87].

Semantics:

In all case frames shown above, the port is represented as a structured individual. The rest of the structure is a straight forward class membership assertion as given in (Structure 5.1.3.5.2). The structured port individual consists of an arc that describes the port type (inport, biport, or outport) and points to the <device> or <wire> it is a port of, and of an id arc that points to an <id-number> that permits to discriminate between all the ports of the same type and the same device.

Example:

m2(	object	m1(	inport-of	gate-1	(7.1.4)
			id	inp1)	
	modality	function			
	type	PORT)			

In the example, the inner molecular node (m1) represents an object which is the inport of the object at the end of its "inport-of" arc (D1) and which can be differentiated from all other inports of this object by the id at the end of the "id" arc (inp1). m2 represents the assertion that m1 is an object of type PORT.

Syntax:

type	POCON			(7.1.5)
modality	<modality>			
object	contact	<port-type-1>	<object-1>	
		id	<id>	
	contact	<port-type-2>	<object-2>	
		id	<id-2>	

For the use of this case frame for maintenance purposes we have to refer the reader again to [Tai87].

Semantics:

Above case frame asserts a connection by creating a structured individual of the class POCON. POCON stands for "Point Of Connection" and describes an abstract object that can best be understood as a representation of a soldering point between a port of a component and a port of a wire. The connection object itself is a structured individual with two structured sub-individuals describing the two ports.

Above case frame differs from all other case frames reported so far. It is a summary description of several case frames, because <port-type-1> and <port-type-2> are *slot names* not slot fillers. <port-type-1> will usually be either "inport-of" or "outport-of". <port-type-2> will usually be "biport-of". Because SNePS is set oriented, these two value assignments to the syntactic variables <port-type-1> and <port-type-2> may be exchanged. The bottom line is that one biport must be connected to either an inport or an outport.

Syntax:

equiv	<port-type>	<object>	(7.1.6)
	id	<id>	
equiv	biport-of	<wire-object>	
	id	<id-2>	

Semantics:

Any node dominating the above structure (7.1.6) expresses the identity between an abstract object "wire-biport" that is represented by its second sub-case-frame, and the concrete *system* inport or outport that is represented by its first sub-case-frame. For a deeper explanation of this structure the reader is again referred to Mingruey Taie's dissertation.

7.2. Layout and Routing as Intelligent Activities

Before the advent of sophisticated CAD equipment it was the normal way of life in an engineering company to have developers create (sometimes awful hand-) drawings of the circuits they wanted to be built. These diagrams then went to the draftsmen who created nicely laid-out wire plans following a few professional conventions. The draftsman does *not* have to understand the functioning of the device he is laying out! It is notable that the job of a draftsman has been considered a low intelligence job, so one should think that AI would have a ready made explanation for "how to do it".

On the other hand over 30 years of AI history have shown that the seemingly easiest problems, like recognizing a face, are often the most difficult problems, and while highly developed medical advisors (of the MYCIN family) [DBS85] have been built, we still have no comprehensive theory of solving many so called easy problems. It seems to us that it is precisely a "simple" job, like the job of a draftsman, which requires a lot of *perceptual* intelligence and is therefore difficult for a program to perform.

It has been a part of this project to model the abilities of a draftsman in creating circuit board diagrams. The problem setting considered is slightly different from one a real draftsman is confronted with, because he, as mentioned before, usually bases his work on a hand drawing. In this research the IMD program receives the knowledge base equivalent of a part list, plus complete connectivity information, including the correct inports and outports of every component instead.

At this point the question naturally arises, whether there is any formal theory of how to draw such circuit board diagrams. A look at textbooks for drafting yields a disappointment [Ren71]. One finds only a few conventions and vague explanations. Biesel, whose work concentrates on circuit board diagrams [Bie84] has collected other evidence for the vagueness of the state of the art in drafting.

IMD is an application of our theory of GDK, because all that is needed for laying out (a class of artificially simple) circuit boards are part, class, form, attribute and inheritability assertions as

introduced before. What is omitted from GDK for IMD are position assertions, they are replaced by non GDK structures that describe ports and connections.

7.3. Intelligent Machine Drafting

It comes as no surprise that Intelligent Machine Drafting *is* a hard problem. Therefore we have defined a small sub-class of circuit boards that will be used as the basis of our analysis. The class was designed around an artificial circuit board called the "adder-multiplier" that has been used in a number of publications on maintenance systems [DSH82, Dav84, DeK86, Tai87]. It was the purpose of this definition to supply a number of other (non-trivially different) circuits besides the adder-multiplier itself.

Before we present the definition of the class of circuit boards we have defined and the theory for their display, we would like to add a refutation against a criticism that has been levied against IMD in the past.

"You are just reinventing computer aided design!"

Study of the CAD literature [MTA83, Shi83] shows that the problems solved here look in fact superficially similar to two famous CAD problems, namely the CAD layout problem and the CAD routing problem. But this similarity is only superficial.

The goal of any CAD layout theory is to create VLSI chips or PWBs (printed wiring boards) that fulfill certain physical requirements. Typical examples would be to keep the length of the clock lines short. This is necessary to ensure that all the chips on a board get the same clock signals. Another requirement would be to keep signal paths in general short, to keep connection wires broad enough to avoid energy loss in the wire, to minimize the overall area of the whole chip or board, etc. The resulting systems are usually complicated mazes.

In *intelligent machine drafting* the goal is to create a representation that is easily readable and understandable by a person. Unnecessary corners in wires add irrelevant information and

should therefore be avoided. Circuits with a standard meaning should be drawn in a standard established layout pattern. All these are *not* the goals of normal layout programs in a CAD environment. Summarizing we can say that CAD designers are interested in the layout of *physical* wire plans, while we are interested in *logical* wire plans.

Similar claims as about layout can be made about routing. The fundamental problem for a routing program for a printed circuit board is to avoid any overcrossing of wires. If there is a danger of a crossing, a via to the second (or another) layer has to be taken. (Vias are the through connectors on a board that connect different layers of conducting strips). On the other hand, there is no problem with a wire passing through under an integrated circuit (as long as it does not come too near to any of the pins of the circuit). In an IMD approach it is unacceptable to have a wire run through any other component. But of course wires may (and will) intersect each other. In a CAD program two wires may run parallel at the same x and y coordinates, as long as they are on different layers. In an IMD program there is only one layer, and two wires that are running parallel have to have a minimum distance from each other. So the problem of logical routing as encountered in intelligent machine drafting is also fundamentally different from the problem of physical routing as encountered in CAD.

We will now present the formal definition of the class of circuit boards that we want to analyze. This class of boards will be referred to as A*M*.

Limitation 7.3.1:

Every device of the class A*M* consists of a main object with a one level part hierarchy. The main object as well as the parts have ports.

Limitation 7.3.2:

Three types of ports are defined, inports, biports, and outports. It is assumed that components are of (approximately) rectangular form with sides high enough to place all inports on the left side and all outports on the right side of every com-

ponent. The same placing must be possible for the main object. Ports will be placed at the sides of the enclosing rectangle (extent) of the component. Biports have been introduced by [Tai87] and are used as ends of wires only.

Definition 7.3.1: Column-1 element.

A column-1 element is a component every input of which is directly connected to a system input of the main object.

Definition 7.3.2: Column-n element.

A column-n element is a component every input of which is directly connected to a column-j element, such that for all j, j $<$ n, or to a system input of the main object.

Definition 7.3.3: Strict column-n element.

A column-n element is called strict iff it is connected to at least one column-(n-1) element.

Definition 7.3.4: Strict column classification.

A strict column classification of a set M of n components, M = { e_1, e_2,..., e_n} is an assignment of every component to a column k such that all elements in this column are strict-k elements.

Definition 7.3.5: k_{max}.

k_{max} is the non empty column with the largest column number.

Definition 7.3.6: Signal Length.

The signal length σ_L(M) of a set of elements is identical to the value k_{max}.

$$\sigma_L(M) = k_{max}.$$

Definition 7.3.7: Signal Width.

The signal width σ_W(M) of a set of elements is defined as

$$\sigma_W = \{\, m/m = MAX_{i=1}^{k_{max}} (\ K_i\ )$$

& K_i is the number of elements in column k}

Definition 7.3.8: Extrapolated Physical Length.

$$\rho_L = \sigma_L * MAX_{i=1}^{N} (X\text{-}Extension(e_i))$$

The X-Extension of a component is the length of the component in the x-direction.

Definition 7.3.9: Extrapolated Physical Width.

This is defined in total analogy to Definition 7.3.8

$$\rho_W = \sigma_W * MAX_{i=1}^{N} (Y\text{-}Extension(e_i))$$

In practice it is assumed that X-Extension and Y-Extension have a fairly limited range. (However they can be quite different from each other!).

Limitation 7.3.3:

All elements have their inputs connected to either other outputs, or to main object inputs. All elements have their outputs connected to either other inputs or to main object outputs. Connections between inputs (inports) and outputs (outports) are always via a system of a wire with two biports.

Limitation 7.3.4:

No feedback loops are permitted. In other words, all connections are going strictly forward.

Limitation 7.3.5:

Ports are small, compared to component sizes. Ports are consistently of the same

size in the whole system.

Limitation 7.3.6:

$$\rho_L < c_x * S_L$$

$$\rho_W < c_y * S_W$$

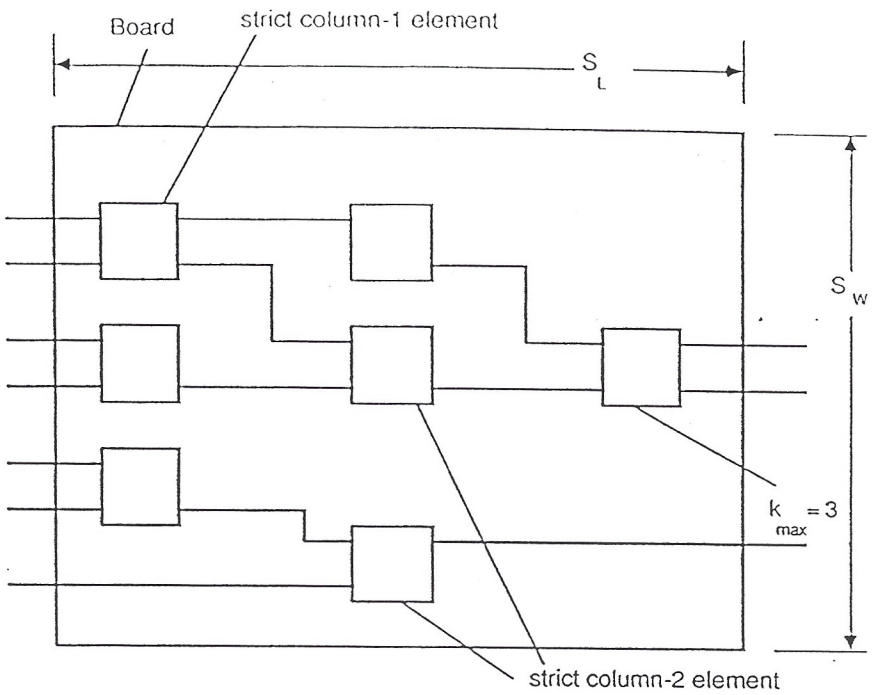
c_x and c_y are constant factors (> 0, < 1) that take care of the size of the ports and the necessary space for wiring. S_L and S_W are the length and the width of the main object (or the permissible display surface). A typical value for c_x would be 0.6.

Intuitively, the above limitations express a very simple condition. They make it possible to organize all the components on a display surface (a screen or a viewport on a screen) or inside of the main object, such that components which have the same signal distance from the main input can all be put above each other, and that all such arrangements of components can be placed next to each other without running out of available space.

Refer to Fig. 7.3.1 for clarification of these conditions. Fig. 7.3.2 shows a few members and non-members of A*M* and gives reasons for the non-membership.

The limitations and definitions given closely mirror the algorithm for displaying objects of the class A*M*. The first step consists of organizing all the components of the circuit into *columns*. The first column contains all components that are directly connected to system inputs. The second column contains all components that are directly connected only to elements of the first column or to inputs of the system, and so on.

Columns are treated as pseudo objects and laid



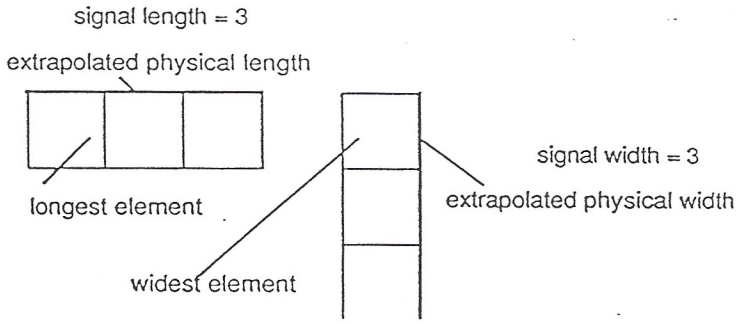
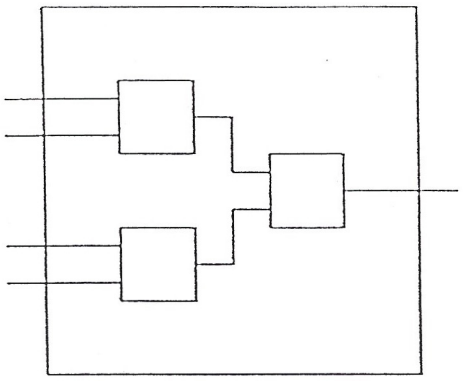
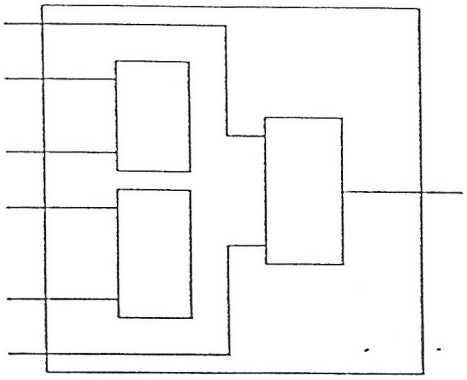


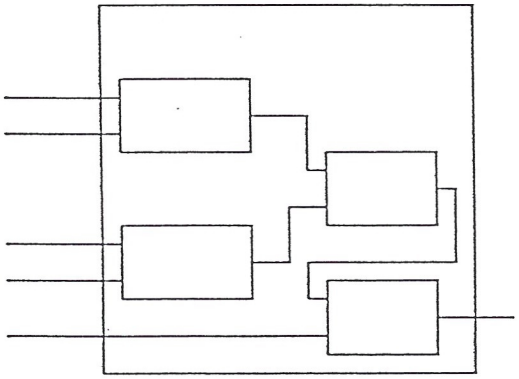
Fig. 7.3.1: The A*M* conditions.



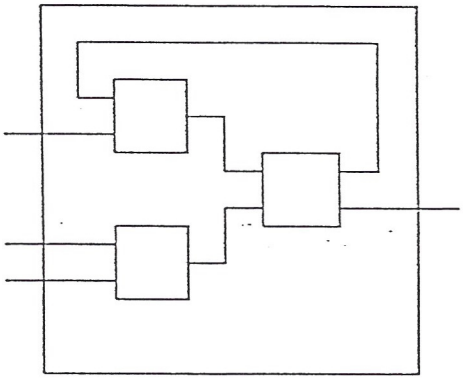
A member



A non-member: Extrapolated
Physical Width too large



A non-member: Extrapolated
Physical Length too large



A non-member:
Feedback prohibited

Fig. 7.3.2: Members and non-members of A*M*

out physically next to each other. The main principle that is used for this operation is the *equal spacing principle.* If there are k columns then the net length of all columns is subtracted from the available space, and the difference is divided into k+1 blank strips which are equally distributed between the columns. Analogously objects are equally spaced inside of their corresponding columns. Ports for all components are laid out using the same equal spacing algorithm. This time the ports are equally spaced over the height of their corresponding components, inports on the left side, outports on the right side.

Wire ports (bi-ports) are laid out such that they line up exactly with inports from the left side and with outports from the right side, and such that they overlay with system inports and outports. It has now become clear why it is necessary to represent the distinction between different types of ports and the connections between adjacent ports. Fig. 7.3.3 shows wires with their bi-ports and how they are combined with other ports.

The router first attempts to place a vertical line in the middle of the two ports nearest to each other that belong to neighboring columns. This is another application of the equal spacing

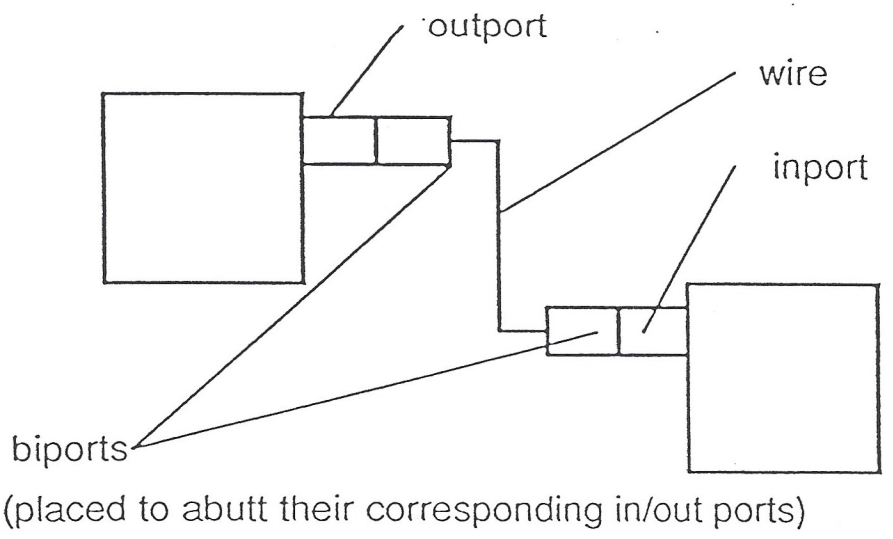


Fig. 7.3.3: The layout of biports.

principle. If the resultant line approaches any of the already laid out wires in a parallel way with a distance of less than a fixed Lambda (measured in pixels), then this solution is retracted, and another solution Lambda to the left of the original vertical line is attempted. If this also fails, then a step Lambda to the right is attempted. If this fails, then Lambda is temporarily doubled, and the same steps are repated, until the vertical wire would come too near to either of the columns. If that happens the router gives up.

Once the vertical line is established, the router tries to connect every port belonging to this line by a horizontal wire. If this is impossible (because an already existing wire is too near) then the router attempts a backtracking strategy resulting in an impulse structure (Fig. 7.3.4) instead of the former horizontal wire. The routing process involves two levels of backtracking at this stage, but it might nevertheless fail.

In the current implementation of the router no column jumps are supported (Fig. 7.3.5), although column jumps are permitted by the definition of the class A*M*. It should be noted that no claims about original research in routing are made. Excellent work has been done on routing

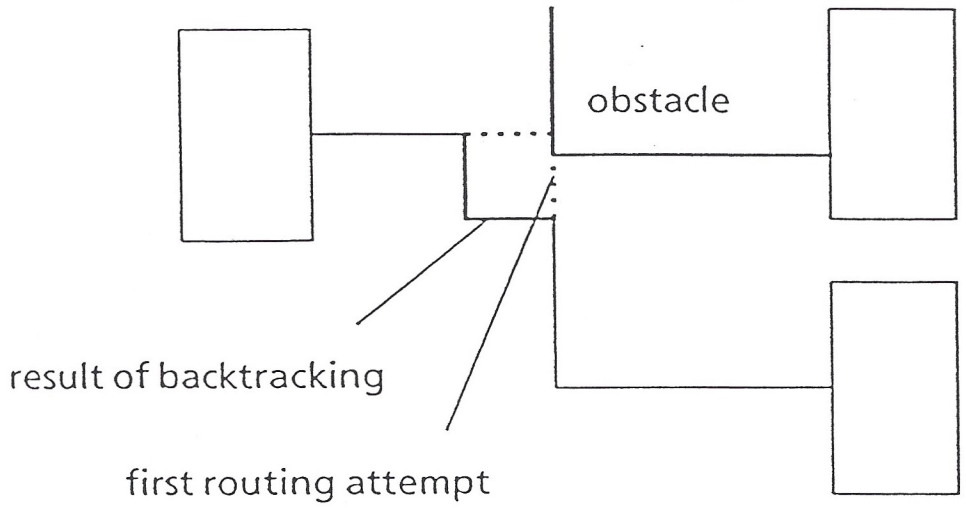


Fig. 7.3.4: The result of router backtracking.

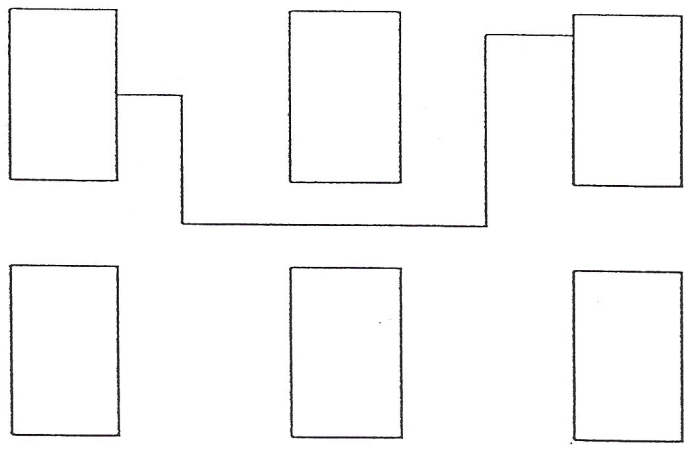


Fig. 7.3.5: An example for a column jump.

algorithms, and we see no reason to compete with people who have made it their job to design routing programs [Shi83]. On the other hand, we would like to remind that other existing routers cannot be adapted uncritically, because we are (as noted before) interested in a cognitive layout with different permitted crossing conditions than the ones valid for physical routing programs.

7.4. The IMD Grammar

A separate grammar was written for the IMD system. It permits the following operations by natural language.

(1) Creation of an object. The user can build objects of a subset of the class A*M* by natural language utterances. The limitations relative to A*M* consist mainly of a limited branching factor for connections.

(2) Display requests for created objects.

The IMD grammar does not contain a "query branch", i. e. no questions about a laid out device can be asked. The following piece of text shows an implemented set of natural language utterances that will build up the whole structure of the (in)famous adder-multiplier. This text has not

been edited in any way and is the original demo file that creates the necessary structure. The call to (nl) in the beginning activates the natural language interface from SNePS, while the term ^end at the end returns to normal SNePS interaction.

(nl) (7.4.1)
D1 is a board
D1M1 is a multiplier
D1M2 is a multiplier
D1M3 is a multiplier
D1A1 is an adder
D1A2 is an adder
D1 has 3 inports
D1 has 2 outports
D1M1 has 2 inports
D1M1 has 1 outport
D1M2 has 2 inports
D1M2 has 1 outport
D1M3 has 2 inports
D1M3 has 1 outport
D1A1 has 2 inports
D1A1 has 1 outport
D1A2 has 2 inports
D1A2 has 1 outport
connect input 1 of D1 with input 1 of D1M1 and input 1 of D1M2
connect input 2 of D1 with input 2 of D1M1 and input 1 of D1M3
connect input 3 of D1 with input 2 of D1M2 and input 2 of D1M3
connect output 1 of D1M1 with input 1 of D1A1
connect output 1 of D1M2 with input 2 of D1A1 and input 1 of D1A2
connect output 1 of D1M3 with input 2 of D1A2
connect output 1 of D1A1 with output 1 of D1
connect output 1 of D1A2 with output 2 of D1
D1M1, D1M2, D1M3, D1A1, and D1A2 are parts of D1
wires are parts of D1
the form of a board is xboard2
the form of a multiplier is xmult2
the form of an adder is xadd2
the form of a PORT is xport
show D1
^end

The result of the above input is indicated in Fig. 7.4.1 .[1]

[1] This is not a screen dump. At the time of writing this chapter no access to a dump facility was available.

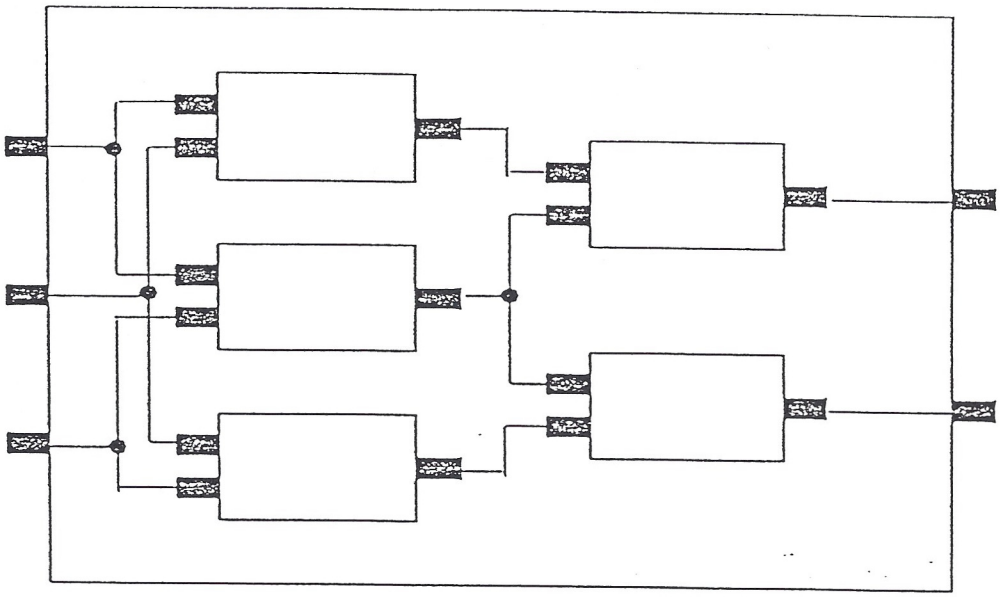


Fig. 7.4.1: The laid out Adder-Multiplier

CHAPTER 8

FUTURE WORK

The main direction into which we wish to develop this research is towards a knowledge based user interface management system (KBUIMS). So far it is not possible in our system to define a menu by a combination of natural language and mouse movements, and to link the menu choices to procedures that should be executed on buttoning one of them. However, it is clearly in the range of our paradigm to achieve this effect. The basic goal and unique characteristic of this approach is the complete integration of the language used to interact with the user interface and the language used to interact with the user interface management system.

Our second main objective is to extend our approach more towards the work of Kosslyn [Kos81b] and to permit "readback" from the generated diagrams to be used for question answering. Specifically we are interested in explaining the examples given by Waltz [Wal80] that engender surprise in a listener. A preliminary analysis shows that only close interaction between propositional and analog representations can cope with this problem.

The current implementation invites improvements in a number of directions. It is not possible to capture dynamic phenomena, for instance one cannot move icons on the screen. The 3-d abilities of the program are very limited. Some combinations of fuzzy position expressions are possible, but others will not result in correct diagrams. Finally, the language interface is not general enough, and the program as a whole is not very robust.

Intelligent Machine Drafting is certainly a completely open field, and we have only scratched the surface of it. Larger, more realistic device classes are necessary, as well as faster algorithms and imagery like "visual debugging". Concerning the Readform program there have been several

preliminary attempts to integrate knowledge handling directly into the graphical object creation.[1] This should be aided by integrating Readform better with the rest of the program.

---

[1] An initial version of such a program was written by one of my undergraduate students, Carl Mercer.

CHAPTER 9

CONCLUSIONS

The purpose of this dissertation has been to analyze the knowledge necessary for Natural Language Graphics. The notion of Graphical Deep Knowledge has been introduced and used as the leading theme throughout this dissertation. Graphical Deep Knowledge has been defined as declarative knowledge that is projectively adequate as well as deductively adequate.

Based on general observations about knowledge representation the First and Second Fundamental Conjecture about Knowledge Representation have been formulated. The First FCoKR describes the limitations of knowledge representations applied to real as opposed to toy domains. The Second FCoKR connects knowledge access and knowledge acquisition by claiming the usefulness of a common indexing mechanism for both these areas, called an internal relevance criterion.

The language side of Natural Language Graphics has led to the formulation of another important principle of knowledge representation, namely the Linearity Principle. The Linearity Principle permits to guide the development of differentially adequate knowledge representation systems by adding constraints for choosing between different possible representational structures for the same linguistic input.

The presented constructs of Graphical Deep Knowledge have been introduced in a frame like notation for SNePS semantic networks. For each construct the syntax was given in this case frame format, and a descriptive semantics was supplied. In addition procedural effects of introduced structures have been explained informally. The constructs of GDK will now be summarized.

Form descriptions are based on primitive forms (icons) which themselves consist of graphics primitives of a LISP graphics package. They are linked to objects by appropriate case frames, or inherited by objects from classes with an associated form. If no form can be derived this way for an object, exemplar inheritance (a. k. a. up-and-down inheritance) is attempted. We have argued

that the latter is a viable technique based on the exemplar view of categorization that should be used if no summary description is available for a class of objects.

Concrete (numerical) as well as fuzzy positions can be represented. All positions are relative, although the screen can be used as a reference object, creating de facto absolute positions. Reference objects of position specifications can be given explicitly or deduced from a part hierarchy. Positions can be inherited along the class hierarchy. Coordinates can be given in absolute units, in body coordinates, in body coordinates of the reference object, or in body coordinates of any other known object. Different coordinate systems can be defined, and projections explicitly selected. Natural language utterances in reference to objects in a plane can nevertheless express a view referring to a third dimension. Such utterances are correctly interpreted.

Attributes of objects have been differentiated into intrinsic, accidental and invisible attributes. Attributes of pictures have been differentiated into representative, symbolic, accidental, and situational. Case frames for relative and absolute attributes have been introduced. Knowledge structures have been introduced that permit to map object attributes into pictures attributes. The actual mapping mechanism of an abstract graphics machine is hereby explicated as a functional that transforms form-comprising functions into new form-comprising functions with an incorporated attribute. Attribute classes with two, three, and zero attribute values have also been described by variations of the attribute case frame. To permit changes in the graphical realization of an attribute, attribute mappings are maintained in a "CURRENT-MAPPING" pointer.

Display modalities permit the separation of a large and unstructured network. It has been shown that this mechanism is equivalent to a partitioned network, however it is superior in its drawability, is consistent with the rest of the representational system, and is more parsimonious than a partitioned network.

Part hierarchies, as used in many AI systems, have been replaced by three different part-like hierarchies, namely Real Parts, Assemblies, and Clusters. The (non-intuitive) relations between modalities and part hierarchies have been shown: even the division of an object into parts depends

on the modality. Inheritance of attributes is possible along the part hierarchy of the system. The inheritability of a specific attribute can be expressed declaratively.

SNePS based logical reasoning in the spatial domain has been discussed. The reasoning facility used in our implementation is called path based inference and permits to describe paths of arcs to derive necessary information about objects. All knowledge retrieval for graphics generation as well as question answering is based on such paths.

One major area of practical application of the work described is the design of intelligent interfaces. In the domain of maintenance systems the use of TINA, our graphics program, was shown as a graphics interface to the VMES (Versatile Maintenance Expert System). The natural language graphics interaction as well as the usefulness of the structures of GDK were demonstrated with a large number of interactive trial runs and with screen dumps.

A second CAD-like interface based on the same theory was also introduced, which however differs from a CAD system in trying to optimize the "readability" of a pictorial representation. With this interface the completely new field of Intelligent Machine Drafting was defined, and a logical (as opposed to physical) layout program as well as a logical router were described that fulfill the requirements of Intelligent Machine Drafting. A formal definition of the artificial device class A*M* was given that has been used as the domain for layout and routing.

In the section on philosophy and Natural Language Graphics it has been shown that the Gricean Maxims of cooperative communication can be used as guidance to graphics design, and as a touchstone that reveals how far current graphics systems are away from human like communication. Of special importance for practical applications are the maxims of quantity and the maxim of manner ("be orderly"). The latter is the case because graphics systems are much less constrained in the order they can create representations than natural language generators and order can be used to exhibit an understanding of the drawn object.

A new maxim for technical languages was introduced, the Maxim of Unnecessary Variation. Two sub maxims, consistency and conservativity have been discussed. Based on this new maxim it

has been criticized that most symbolic graphical representations in most areas of science are introduced without an explicit explanation which of the features of the representation are semantically meaningful, conventional, or accidental. A number of different symbolic representation systems have been analyzed according to this feature classification. It has been argued that only by explicitly representing such feature classifications, a system can achieve understanding of the diagrams it is drawing.

REFERENCES

[AMG84a] G. Adorni, M. D. Manzo and F. Giunchiglia, "From Description to Images: What Reasoning in Between?", *European Conference on AI*, Pisa, 1984.

[AMG84b] G. Adorni, M. D. Manzo and F. Giunchiglia, "Natural Language Driven Image Generation", *COLING 84*, 1984.

[All86] B. P. Allen, "Constructing User Interfaces with a Rule-Based User Interface Management System", in *AAAI 86 Workshop on Intelligence in Interfaces*, B. Neches and T. Kaczmarek (editor), August 14, 1986, 31-36.

[Alm87] M. J. Almeida, *Reasoning about the Temporal Structure of Narratives*, Dept. of Computer Science, State University of New York at Buffalo, (dissertation), 1987.

[AnB75] A. Anderson and N. Belnap, *Entailment: The Logic of Relevance and Necessity*, Princeton University Press, 1975.

[And83] J. R. Anderson, *The Architecture of Cognition*, Harvard University Press, Cambridge MA, 1983.

[AMS88] Y. Arens, L. Miller and N. Sondheimer, "Presentation Planning Using an Integrated Knowledge Base", *Architectures for Intelligent Interfaces: Elements and Prototypes*, Monterey Workshop, March 1988.

[Arn86] R. Arnheim, *New Essays on the Psychology of Art*, University of California Press, Berkeley, CA, 1986.

[BaB82] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, 1982.

[Bat78] M. Bates, "The theory and practice of augmented transition network grammars", in *Natural Language Communication with Computers*, vol. 63 , L. Bolc (editor), Springer, Lecture Notes in Computer Science, New York, 1978, 191-259.

[BMR82] I. Biederman, R. J. Mezzanotte and J. C. Rabinowitz, "Scene Perception: Detecting and Judging Objects Undergoing Relational Violations", *Cognitive Psychology 14* (1982), 143-177.

[Bie87] I. Biederman, "Recognition-by-Components: A Theory of Human Image Understanding", *Psychological Review 94*, 2 (1987), 115-147.

[Bie84] H. D. Biesel, *On encoding functional and schematic description of complex systems*, Dissertation at Rutgers Univ., publ. by University Microfilms International, Ann Arbor, MI, 1984.

[BoW77a] D. G. Bobrow and T. Winograd, "Experience with KRL-0: One Cycle of a Knowledge Representation Language", *IJCAI-77*, 1977, 213-222.

[BoW77b] D. G. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language", *Cognitive Science 1*, 1 (1977), 3-46.

[Bol77] D. Bolinger, Neutrality, Norm, and Bias, 1977.

[Bra79] R. J. Brachman, "On the Epistemological Status of Semantic Networks", in *Associative Networks*, N. Findler (editor), Academic Press, New York, 1979.

[BFL83] R. J. Brachman, R. E. Fikes and H. J. Levesque, "KRYPTON: A Functional Approach to Knowledge Representation", *Computer 16*, 10 (1983), 67-73.

[BFL85] R. J. Brachman, R. E. Fikes and H. J. Levesque, "Krypton: A functional approach to Knowledge Representation", in *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque (editor), Morgan Kaufmann Publishers Inc., Los Altos, California, 1985, 411-439.

[BrS85] R. J. Brachman and J. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science 9*, 2 (1985), 171-216.

[BrK77] D. C. Brown and S. C. Kwasny, "A Natural Language Graphics System", OSU-CISRC-Tech. Rep.-77-8, Dept. of Computer and Information Science, The Ohio State University, 1977.

[BrC81] D. C. Brown and B. Chandrasekaran, "Design Consideration for Picture Production in a Natural Language Graphics System", *Computer Graphics 15*, 2 (July 1981), 174-207.

[BrM87] B. Bruce and M. G. Moser, "Case Grammar", in *Encyclopedia of Artificial Intelligence*, S. C. Shapiro (editor), John Wiley, New York, 1987, 333-339.

[BuB79] R. R. Burton and J. S. Brown, "Toward a Natural Language Capability for Computer-Assisted Instruction", in *Procedures for Instructional System Development*, H. O'Neill (editor), Academic, New York, 1979, 273-313.

[Car70] J. R. Carbonell, "AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction", *IEEE Transactions on Man-Machine Systems MMS-11*, 4 (December 1970), 190-202.

[ChM85] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Addison Wesley, Reading, Mass., 1985.

[CoQ69] A. M. Collins and R. M. Quillian, "Retrieval Time from Semantic Memory", *Journal of Verbal Learning and Verbal Behavior 8* (1969).

[CoL75] A. M. Collins and E. F. Loftus, "A Spreading Activation Theory of Semantic Processing", *Psychological Review 82*, 6 (1975).

[Con72] C. Conrad, "Cognitive Economy in Semantic Memory", *Journal of Experimental Psychology 92*, 2 (1972), 149-154.

[DSH82] R. Davis, H. Shrobe, W. Hamscher, K. Wiechert, M. Shirley and S. Polit, "Diagnosis Based on Description of Structure and Function", *Proceedings of AAAI-82*, Aug. 1982, 137-142.

[Dav84] R. Davis, "Diagnostic Reasoning Based on Structure and Behavior", *Artificial Intelligence 24* (1984), 347-410.

[DBS85] R. Davis, B. Buchanan and E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program", in *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque (editor), Morgan Kaufmann Publishers Inc., Los Altos, California, 1985, 371-387.

[DeK86] J. DeKleer, "Reasoning about Multiple Faults", *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 1986, 132-139.

[EaL86] P. Eades and A. J. Lee, "Perception of Symmetry", 67, University of Queensland, St. Lucia, March 1986.

[Fah79] S. E. Fahlmann, *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, Mass., 1979.

[Fil68] C. J. Fillmore, "The Case for Case", in *Universals in Linguistic Theory*, E. Bach and R. T. Harms (editor), Holt, Rinehart, and Winston, New York, 1968, 1-88.

[FoD83] J. D. Foley and A. V. Dam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, 1983.

[Fri84] M. Friedell, "Automatic Synthesis of Graphical Object Descriptions", *Computer Graphics 18*, 3 (1984), 53-62.

[Fri72] N. H. Frijda, "Simulation of human long-term memory", *Psychol. Bull. 77* (1972), 1-31.

[GaS86] B. R. Gaines and M. G. L. Shaw, "From Timesharing to the Sixth Generation: the Development of Human-Computer Interaction (Part 1)", *International Journal of Man Machine Studies 24* (1986), 1-27.

[GeS87] J. Geller and S. C. Shapiro, "Graphical Deep Knowledge for Intelligent Machine Drafting", *Tenth International Joint Conference on Artificial Intelligence*, August 1987.

[GTS87] J. Geller, M. R. Taie, S. C. Shapiro and S. N. Srihari, "Device Representation and Graphics Interfaces of VMES", *Applications of Artifical Intelligence in Engineering*, August 1987.

[Ger85] J. S. Gero, *Knowledge Engineering in Computer Aided Design*, North Holland, 1985.

[GLM78] R. D. Giustini, M. D. Levine and A. S. Malowany, "Picture Generation Using Semantic Nets", *Computer Graphics and Image Processing 7*, 1 (1978), 1-29.

[Gri75] H. P. Grice, "Logic and Conversation", in *Syntax and Semantics: Speech Acts*, vol. 3 , P. C. J. L. Morgan (editor), Academic Press, New York, 1975, 41-59.

[Gri78] H. P. Grice, "Further Notes on Logic and Conversation", in *Syntax and Semantics 9: Pragmatics*, P. Cole (editor), Academic Press, New York, 1978, 113-127.

[HaS86] J. Haan and L. K. Schubert, "Inference in a Topically Organized Semantic Net", *Proceedings of the Fifth National Conference on Artificial Intelligence*, Los Altos, CA, 1986, 334-338.

[Har72] L. D. Harmon, "Automatic Recognition of Print and Script", *Proceedings of the IEEE 60*, 10 (October 1972), 1165-1176.

[Hay77] P. J. Hayes, "In Defence of Logic", *IJCAI-77*, Los Altos, CA, 1977, 559-565.

[Hen79] G. G. Hendrix, "Encoding Knowledge in Partitioned Networks", in *Associative Networks*, N. Findler (editor), Academic Press, New York, 1979.

[Her86] A. Herskovits, *Language and Spatial Cognition*, Cambridge University Press, Cambridge, UK, 1986.

[Hin79] G. E. Hinton, "Some demonstrations of the effects of structural descriptions in mental imagery", *Cognitive Science 3*, 3 (1979), 231-250.

[HMR88] J. Hollan, J. Miller, E. Rich and W. Wilner, "Knowledge Bases and Tools for Building Integrated Multimedia Intelligent Interfaces", *Architectures for Intelligent Interfaces: Elements and Prototypes*, Monterey Workshop, March 1988.

[Hul85] J. J. Hull, "Word Shape Analysis in a knowledge based system for reading text", *Second IEEE Conference on Artificial Intelligence Applications*, 1985.

[Jam83] A. Jameson, "Impression Monitoring in Evaluation-Oriented Dialog. The Role of the Listener's Assumed Expectations and Values in the Generation of Informative Statements", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, 1983, 616-620.

[Kad86] R. R. Kadesch, "Subjective Inference with Multiple Evidence", *Artificial Intelligence 28*, 3 (1986), 333-341.

[KeB81] G. Keren and S. Baggen, "Recognition Models of Alphanumeric Characters", *Perception and Psychophysics 29*, 3 (1981), 234-246.

[Kir64] R. A. Kirsch, "Computer Interpretation of English Text and Picture Patterns", *Transactions on Electronic Computers 13* (August 1964), 363-376, IEEE.

[KoS77] S. M. Kosslyn and S. P. Shwartz, "A Simulation of Visual Imagery", *Cognitive Science 1* (1977), 265-295.

[Kos80] S. M. Kosslyn, *Image and Mind*, Harvard University Press, Cambridge MA, 1980.

[Kos81a] S. M. Kosslyn, "Research on Mental Imagery: Some Goals and Directions", *Cognition 10* (1981), 173.

[Kos81b] S. M. Kosslyn, "The Medium and the Message in Mental Imagery: A Theory", *Psychological Review 88/ N.1* (January 1981).

[Kos85] S. M. Kosslyn, "Stacking the Mental Image", *Psychology Today*, May 1985.

[LeB85] H. J. Levesque and R. J. Brachman, "A Fundamental Tradeoff in Knowledge Representation and Reasoning", in *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque (editor), Morgan Kaufmann Publishers Inc., Los Altos, California, 1985, 42-70.

[Ley86] M. Leyton, "A theory of information structure II: A theory of perceptual organization", *Journal of Mathematical Psychology 30* (1986), 257-305.

[Lyo77] J. Lyons, *Semantics*, Cambridge University Press, New York, 1977.

[MaB87] R. MacGregor and R. Bates, "The LOOM Knowledge Representation Language", ISI/RS-87-188, USC/Information Sciences Institute, 1987.

[MGP84] M. D. Manzo, F. Giunchiglia and E. Pino, "Space Representation and Object Positioning in Natural Language Driven Image Generation", *International Conference on AI Methodlogy - Systems - Applications, AIMSA Varna (Bulgaria)*, 1984.

[MaN78] D. Marr and K. H. Nishihara, "Visual Information Processing: Artificial Intelligence and the Sensorium of Sight", *Technology Review*, Oct. 1978, 28-49.

[MaS83] J. P. Martins and S. C. Shapiro, "Reasonig in Multiple Belief Spaces", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, 1983, 370-373.

[McC83] G. McCalla and N. Cercone, "Approaches to Knowledge Representation", *Computer*, Oct. 1983, 12-18.

[McD81] D. McDermott, "Artificial Intelligence meets Natural Stupidity", in *Mind Design*, J. Haugeland (editor), The MIT Press, Cambridge MA, 1981, 143-160.

[MiR85] H. Mili and R. Rada, "A statistically built knowledge base", *Expert systems in government*, 1985, 457-463.

[Mil56] G. A. Miller, "The magical number seven, plus or minus two: some limits of our capacity for processing information", *Psychological Review 63* (1956), 81-97.

[MiJ76] G. A. Miller and P. N. Johnson-Laird, *Language and Perception*, The Belknap Press of Harvard University Press, Cambridge MA, 1976.

[Min68] M. L. Minsky, "Matter, Mind, and Models", in *Semantic Information Processing*, M. L. Minsky (editor), The MIT Press, Cambridge, MA, 1968, 425-432.

[Min75] M. Minsky, "A Framework for Representing Knowledge", in *The Psychology of Computer Vision*, P. H. Winston (editor), McGraw-Hill, New York, 1975, 211-277.

[MTA83] H. Mori, F. Tomoyuki, M. Annaka, S. Goto and T. Ohtsuki, "Advanced Interactive Layout Design System for Printed Wiring Boards", in *Hardware and Software Concepts in VLSI*, G. Rabat (editor), Van Nostrand Reinhold, New York, 1983, 495-523.

[Nea85] J. G. Neal, "A Knowledge Based Approach to Natural Language Understanding", 85-06, State University of New York at Buffalo, May 1985.

[NeS88] J. G. Neal and S. C. Shapiro, "Intelligent Multi-Media Interface Technology", *Architectures for Intelligent Interfaces: Elements and Prototypes*, Monterey Workshop, March 1988.

[NeK86a] R. Neches and T. S. Kaczmarek, "Where should the intelligence in intelligent interfaces be placed", *Proceedings of the Fifth National Conference on Artficial Intelligence*, Philadelphia, PA, August 1986, 1151-1152.

[NeK86b] B. Neches and T. Kaczmarek, AAAI-86 Workshop on Intelligence in Interfaces, August 14, 1986.

[Pal85] P. W. Palumbo, "Two-dimensional heuristic augmented transition network parsing", *The 2nd AI Applications conference*, Dec. 1985.

[PaS81] M. A. Papalaskaris and L. Schubert, "Parts Inference: Closed and Semi-Closed Partitioning Graphs", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, 304-309.

[PeS87] S. L. Peters and S. C. Shapiro, "A Representation for Natural Category Systems", *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August 1987.

[Pin84] S. Pinker, "Visual Cognition: An Introduction", in *Visual Cognition*, S. Pinker (editor), Elsevier Science Publishers, Amsterdam, 1984, 1-64.

[PAC85] A. Poggio, J. J. G. L. Aceves, E. J. Craighill, D. Moran, L. Aguilar, D. Worthington and J. Hight, "CCWS: A Computer-Based, Multimedia Information System", *Computer 18*, 10 (Oct. 1985), 92-103.

[Qui68] M. R. Quillian, "Semantic Memory", in *Semantic Information Processing*, M. L. Minsky (editor), The MIT Press, 1968, 227-270.

[ReC81] R. Reiter and G. Criscuolo, "On Interacting Defaults", *IJCAI*, 1981.

[Ren71] B. A. Renton, *Electrical and Electronics Drafting*, Hayden Book Co., New York, 1971.

[Req80] A. A. G. Requicha, "Representation for Rigid Solids: Theory, Methods, and Systems", *Computing Surveys 12*, 4 (December 1980), 437-464.

[RPK85] J. K. Reynolds, J. B. Postel, A. R. Katz, G. G. Finn and A. L. DeSchon, "The DARPA experimental multi media mail system", *Computer*, Oct. 1985, 82-91.

[Riv87] I. Rival, "Picture Puzzling", *The Sciences 27*, 1 (January 1987), 40-46.

[Rob86] G. Robins, *The NIKL Manual*, USC/Information Sciences Institute, April 1986.

[Ros78] E. Rosch, "Principles of Categorization", in *Cognition and Categorization*, E. R. B. Lloyd (editor), Lawrence Erlbaum, 1978, 27-48.

[Sal72] G. Salton, "Dynamic Document Processing", *Communications of the ACM 15*, 7 (July, 1972), 658-668.

[Sal73] G. Salton, "Recent Studies in Automatic Text Analysis and Document Retrieval", *J. ACM 20*, 2 (April 1973), 258-278.

[Sam86] T. Samad, *A Natural Language Interface for Computer-Aided Design*, Kluwer Academic Publishers, Boston, 1986.

[ScA77] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Press, Hillsdale, N.J., 1977.

[ScE86] A. Schappo and E. A. Edmonds, "Support for Tentative Design: Incorporating the Screen Image as a Graphical Object in PROLOG", *Int. J. Man Machine Studies 24*, 6 (June 1986), 601-609.

[Sha71] S. C. Shapiro, "The MIND System: A Data Structure for Semantic Information Processing", R-837-PR, United States Air Force Project RAND, Santa Monica, CA, August 1971.

[ShW76] S. C. Shapiro and M. Wand, "The Relevance of Relevance", Report 46, Indiana University, 1976.

[Sha79a] S. C. Shapiro, "Using Non-Standard Connectives and Quantifiers for Representing Deduction Rules in a Semantic Network", *Presented in Tokyo at: Current Aspects of AI Research*, Aug. 27-28, 1979.

[Sha79b] S. C. Shapiro, "The SNePS Semantic Network Processing System", in *Associative Networks: The Representation and use of Knowledge by Computers*, N. V. Findler (editor), Academic Press, New York, 1979, 179-203.

[Sha82] S. C. Shapiro, "Generalized augmented transition network grammars for generation from semantic networks", *The American Journal of Computational Linguistics 8*, 1 (1982), 12-25.

[ShM82] S. C. Shapiro and A. S. Maida, "Intensional Concepts in Propositional Semantic Networks", *Cognitive Science 6* (1982).

[ShS83] S. C. Shapiro and T. SNePS Implementation Group, "SNePS User's Manual", SNeRG Bibliography #31, SUNY at Buffalo, Sept. 1983.

[ShR86] S. C. Shapiro and W. J. Rapaport, "SNePS Considered as a Fully Intensional Propositional Semantic Network", *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986, 278-283.

[ShG86a] S. C. Shapiro and J. Geller, "Artificial Intelligence and Automated Design", *1986 SUNY Buffalo Symposium on CAD: The Computability of Design*, Dec 6-7, 1986.

[ShG86b] S. C. Shapiro and J. Geller, "Knowledge Based Interfaces", in *AAAI 86 Workshop on Intelligence in Interfaces*, B. Neches and T. Kaczmarek (editor), August 14, 1986, 31-36.

[SST86] S. C. Shapiro, S. N. Srihari, M. Taie and J. Geller, "VMES: A Network Based Versatile Maintenance Expert System", *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, New York, April 1986, 925-936.

[Shi83] I. Shirakawa, "Routing High Density Printed Wiring Boards", in *Hardware and Software Concepts in VLSI*, G. Rabat (editor), Van Nostrand Reinhold, New York, 1983, 452-479.

[SmM81] E. E. Smith and D. L. Medin, *Categories and Concepts*, Harvard University Press, Cambridge MA, 1981.

[Smi83] R. G. Smith, "STROBE: Support for Structured Object Knowledge Representation", *IJCAI-83*, August 1983, 855-858.

[Smi85] B. C. Smith, "Prologue to Reflection and Semantics in a Procedural Language", in *Readings in Knowledge Representation*, R. Brachman and H. Levesque (editor), Morgan Kaufmann, 1985, 31-39.

[Son76] N. K. Sondheimer, "Spatial reference and natural-language machine control", *International Journal of Man Machine Studies 8*, 3 (1976), 329-336.

[SpW86] D. Sperber and D. Wilson, *Relevance: Communication and Cognition*, Harvard University Press, Cambridge, MA, 1986.

[SuT88] J. W. Sullivan and S. W. Tyler, *Architectures for Intelligent Interfaces: Elements and Prototypes*, ACM/SIGCHI, Monterey Workshop, March 1988.

[Tai87] M. R. Taie, "Representation of Device Knowledge For Versatile Fault Diagnosis", Tech. Rep. 87-07, Dept. of Computer Science, SUNY at Buffalo, May 1987.

[TGS87] M. R. Taie, J. Geller, S. N. Srihari and S. C. Shapiro, "Knowledge Based Modeling of Circuit Boards", *Proceedings of the Annual Reliability and Maintainability Symposium*, January 1987, 422-427.

[Wal87] D. Walters, "Selection of Image Primitives for General-Purpose Visual Processing", *Computer Vision, Graphics, and Image Processing 37* (1987), 261-298.

[Wal80] D. L. Waltz, "Understanding Scene Descriptions as Event Simulations", *ACL*, 1980, 7-11.

[WiR86] J. M. Wiebe and W. J. Rapaport, "Representing De Re and De Dicto Belief Reports in Discoure and Narrative", *Proceedings of the IEEE 74*, 10 (October 1986), 1405-1413.

[WiB83] Y. Wilks and J. Bien, "Beliefs, Points of View and Multiple Environments", *Cognitive Science 7* (1983), 95-119.

[Win72] T. Winograd, *Understanding Natural Language*, Academic Press, New York, 1972.

[WCH87] M. E. Winston, R. Chaffin and D. Herrmann, "A Taxonomy of Part-Whole Relations", *Cognitive Science 11*, 4 (1987), 417-444.

[Woo70] W. A. Woods, "Transition Network Grammars for Natural Language Analysis", *Comunications of the ACM 10* (1970), 591-606.

[Woo87] W. A. Woods, "Knowledge Representation: What's Important About It?", in *The Knowledge Frontier*, N. Cercone and G. McCalla (editor), Springer Verlag, New York, 1987, 44-79.

[YTK84] M. Yokota, R. Taniguchi and E. Kawaguchi, "Language-Picture Question-Answering Through Common Semantic Representation And its Application to the World of Weather Report", in *Natural Language Communication with Pictorial Information Systems*, L. Bolc (editor), 1984.

[ZGY81] F. Zdybel, N. R. Greenfeld, M. D. Yonke and J. Gibbons, "An Information Representation System", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, 978-984.