GRAFLOG: A Theory of Semantics for Graphics with Applications to Human-Computer Interaction and CAD Systems

LUIS ALBERTO PINEDA CORTES

PhD

UNIVERSITY OF EDINBURGH

1989

Declaration

I declare that this thesis is my own work,
and has been composed entirely by my own hand.

December 1989.

A mi papá, Daniel, por hablarme del cariño

A mi mamá, Virginia, por hablarme del trabajo y de la fé

A Nydia, por hablarme del amor

A Nydia, Lilián y Alex, por hablar tanto

Este trabajo está dedicado

con mucho cariño, amor y fé para mi amorilina.

Pués claro que sí!

Acknowledgements

I would like to thank in the first place Dr. Ewan Klein for the time and effort that he devoted to the research that culminates with this thesis. I appreciate his numerous personal tutorials on linguistics and semantics, and the continuous, sometimes painful, confrontation of his theoretical views with my rather pragmatic vision of computation. My hope is that this work reflects a modest synthesis of these two opposite views. I also acknowledge his continuous support, encouragement and the meaningful friendship that he has offered me.

I would also like to thank Dr. John Lee for the huge amount of time and effort that he devoted to this research, for having taught me how to write in English, for the continuous feedback that he has given me during the writing of this thesis, and for his friendship.

For me, this thesis represents the culmination of a large period of preparation and study. I would like to thank as well Gerardo León Lastra and Sergio Santanta who taught me the little I know about computer graphics and computational geometry at Instituto de Investigaciones Eléctricas, México.

There are several people who had also supported me along all these years. I would like to mention specially Ramesh Krishnamurti for his help and support on the early stages of this work. I also appreciate the encouragement and support of Aart Bijl. I would like to mention as well Henk Zeevat and Marc Moens at the Centre for Cognitive Science and Chris Tweed, Peter Szalapaj and Margaret McDougall at EdCAAD.

I acknowledge the economic support of the Mexican Council for Science and Technology (CONACYT), the Instituto de Investigaciones Eléctricas, México, and the Bank of México. I also acknowledge the support of the British ORS award in my last year of studies.

During these years I had the moral support of my family in México. For them, my success in this enterprise is something that has an important meaning. I have always been encouraged by my father, Daniel, and by my mother, Virginia. The support and faith of my brothers and sisters has also given me the strength for completing this thesis. I have had in my mind Daniel, Claudia, Arturo, Magda, Juan Carlos, Carmen, Alfredo, Chela, Lety, Benjamín, Gustavo, Ricardo and Lourdes all the time.

Finally, this thesis is due to the love, patience, sacrifices, and support of my wife, Nydia, and our children, Nydia, Lilián and Alejandra. To them, I dedicate this work.

Abstract

In this dissertation a theory of semantics for graphics for Human-Computer Interaction and CAD systems is presented. This theory allows the integration of interactive computer graphics and natural language processing facilities. Through this theory, the 'knowledge' of both natural language and graphical expressions is represented in an integrated fashion. The semantics of the representational language is explained as an algebraic system that has a dual algorithmic interpretation. In the algebraic domain, the meaning of drawings is explained in terms of a modal multi-sorted algebraic system. The algorithmic interpretation, on the other hand, is used for integrating the declarative semantics of the representational language with the algorithmic knowledge that is required for producing synthetic pictures in computer graphics.

This theory was developed in conjunction with a computer program called GRAFLOG. The program is an interactive drafting system that has been augmented with a natural language processing facility. The interactive concepts that emerge in this kind of interface as well as the architecture of the program are illustrated and discussed. The grammatical formalism for handling the natural language facility, and the computational linguistic tools needed for its implementation are also shown.

The theory and the program are applied to a simple design task: geometric reasoning for geometric modelling in the wire-frame drawing domain. For this purpose a representational language for expressing design knowledge in this domain is developed. The language is useful for defining the notions of design concept, design intention, and design intention satisfaction. The language is used not only for following up the consequences of the information that is expressed through graphics and natural language, but also for interpreting design intentions that are expressed by human-users, solving in this way design problems that occur in the course of an interactive design session. The representational system is used as well for producing a graphics and natural language explanation of the methods by which the system comes to the solution of design problems.

In the last chapter, a reflection on the relation of computer graphics and AI is presented.

LIST OF CONTENT

Chapter 1

Natural Language and Interactive Computer Graphics

This dissertation is about human-computer interaction. However, this work is not about programming methodologies, nor about geometrical and topological techniques for producing fancy pictures. This dissertation is developed on a notion that is familiar to many computer disciplines, as Artificial Intelligence (AI), but which is not usually made explicit in graphics systems. The idea is that an explicit notion of linguistic interpretation is useful for computer graphics. The linguistic interpretation, for the purposes of this work, is the meaning given to graphical representations by human-users. Architectural drawings, for instance, are made out of lines and polygons, but these geometrical entities are interpreted as walls and rooms. Lines and polygons are elements of drawings, but the things that they refer to are objects in the world. Drawings are representations of states of affairs in the world, and as such have a meaning. The purpose of this work is to advance a theory in which those meanings are characterised and used in the context of interactive computer graphics. I like to emphasise that graphical systems have or use an implicit notion of interpretation; however, this interpretation is kept in the mind of the human-user or embedded in the program's code. The only claim that I would like to make here is that to take this notion to the surface, in an explicit way, by means of an interactive natural language processing facility, is a productive idea.

There are at least three important reasons, I believe, for considering an explicit notion on linguistic interpretation in interactive computer graphics. The first is related to the communication process. We hear very often that graphics is a language. It is often said as well that graphics provides a better language than other communication systems. Graphics are easy to define, easy to learn and easy to use. Everybody 'knows' that 'a picture is better than a thousand words'. However, it is extremely difficult to tell what is the source of these intuitions about graphics, and even more, whether they are in fact 'true'. The fact of the matter is that graphical representations are understood through their interpretation. We see the geometry, shape and colour, as standing for some or another thing according to the current subject of discourse. A line can mean many things depending on the context in which it is drawn: a wall in an architectural drawing, the border between two countries in a political map, the number one, the letter *l* etc.

The meaning that we give to graphical symbols and relations in a drawing determines the 'graphical language' of which that drawing is an expression. Given a basic underlying geometrical substratum we can define different graphical languages simply by changing the linguistic interpretation of the current drawing. Then, different graphical languages can be defined in the same computer graphics environment, and can be used by human-users working in different application domains.

The second reason for making explicit the linguistic interpretation of graphics is that it provides an additional source of information that can be used for solving problems in a more efficient and flexible way than traditional drafting and CAD systems do. This is so because the notion of interpretation makes it feasible to enrich graphics systems with ideas and techniques developed in other computer disciplines. For instance, we can augment their functionality with representational schemes, planning systems, decision making systems, etc. that have been developed in AI. In this dissertation, the case of so-called design systems is considered. In traditional drafting systems, design questions are related to the external or substantial aspects of graphical representations. Graphic systems are used as tools through which design objects are expressed but their design functionality is restricted to performing analytical tasks. Design systems in which not only geometrical aspects of design objects but also their semantic interpretation are represented allow us to ask design questions related to the function and purpose of these objects, and computer tools can be used to perform not only analytic but also synthetic tasks.

The third reason is that when the notion of semantic interpretation of graphics is made explicit, a number of tacit assumptions and presuppositions that are usually taken for granted by programmers and users of graphics programs come to the surface and become subject of critical discussion.

Among those presuppositions is that drawings produced in the course of interactive sessions will receive the proper interpretation by human-users. Programmers define data structures which are visualised as drawings by means of the graphics hardware. Programmers think of those data structures and the algorithms acting upon them as representations. However, the question of whether the drawings themselves are isomorphic representations of the underlying data-structures is not always answered. Programmers take for granted that human users, interacting with the drawings themselves, will interpret and use them in the proper way, even though the 'proper' interpretation might have never been stated in an explicit way. If we believe that drawings are representations, we would like to know what kind of representation they are, and as such what sort of properties they exhibit. If the interpretation of a drawing is made explicit through natural language at the time graphical representations are defined in the interactive session, we can say on more solid grounds that the human and machine interpretations are alike.

The second assumption that I would like to mention here is that in computer graphics there is a traditional sharp demarcation line between what is considered the programmer's domain and the user's domain. Specialised programmers research the application domain and translate their findings into the application's code. This approach has several weaknesses; it forces human-users to use a predefined and restricted conception of their own problems. However, human-users have intuitions that have evolved through many years of experience. They face design problems in an integrated, holistic, fashion. They are not necessarily aware of the rules that they use in the design process. They are familiar with their working tools through which their works are given shape. Furthermore, their experience cannot be thought of as a set of fixed beliefs. Experience is not captured by a rigid representational scheme. Experience evolves through practice. The dynamic process of design produces new working environments and old programs become obsolete.

This process of researching an application domain can be thought of in terms of a process of differentiation and integration. Human-users perceive the environment as a whole but for answering the questions that they are asked during the so-called 'analysis phase of the design cycle' they are forced to atomise, to make explicit distinctions. Through this process, an explicit account of the knowledge of the domain to be modelled is produced and communicated from human-users to programmers. But once again, the information is conveyed through the continuum of natural language and graphical aids that the programmer uses for identifying the descriptive patterns in terms of which traditional programs are produced. We develop programs hoping that in this reduction-expansion process no knowledge is lost. I believe that the introduction of an explicit notion of interpretation can smooth the relationship between programmers and human users, because the interpretation can be thought of as the specification of the program, and in some cases, as the program itself.

There are some additional reflections that are prompted by this academic exercise. The definition of an explicit notion of linguistic interpretation of computer graphics poses some rather difficult questions related to the relationship between human knowledge, on the one hand, and computer representations, on the other. Human knowledge is always expressed and used in relation to a context, but computer structures are discrete representations of the world, and yet human-users interact with computers in a natural 'human-like' fashion.

The relation between holistic contexts and discrete distinctions is a very elusive notion. We might assume for analytic purposes that the ability to realise similarities, and then to establish categories and define rules, depends on a faculty by which we can extract and classify distinctions from the context in which they occur. Contexts are grasped in a holistic way; they come as a continuum. A graphical symbol, on the other hand, is a discrete entity represented within the memory of the computer, and the symbol's internal unit is defined beforehand. The human-user sees drawings in a holistic fashion, and he or she builds up the whole context from the discrete parts in the interpretation process. However, not everything arising in a graphical context can be considered context free. Consider, for instance, the architectural plane representing a house in Figure 1.1.

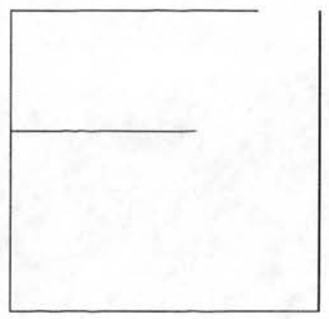

FIGURE 1.1

We can say that the drawing is made out of lines which are interpreted as walls. But what is relevant in this drawing is that the lines are there not only with the purpose of depicting the walls; they are also limiting the space that depicts, for instance, the rooms of the house. We can say that there are three rooms, or two rooms and a corridor, but there are no symbols denoting them. A region of the plane has the denotative function, and this spatial region is limited by the lines denoting the walls, but it is not fully determined by them. However, these rooms are graphical entities that have precise properties and stand in some relations to other graphical objects. For instance, rooms have areas, and the bedroom and the living-room are adjacent to each other, etc. We cannot afford to lose this information for modelling architectural design. A human interpreter is able to make these interpretations without any problem. We might just decide that a region of the space is a room, another a bedroom and another the living-room. The distinction has to have an internal unit, a differentiable pattern, in order to be considered as a discrete thing. But we cannot say that wholes are composed of the sum of the discrete parts. The union of the spatial regions depicting the rooms depicts the house. If I have a pair of scissors I can cut out the house, but I can also cut out the rooms, or the individual walls, etc. But for extracting all these different parts of the house I need many copies of the drawing: one for the whole house, one for the rooms, one for the walls, etc. However, for interpreting the picture, I can make all these extractions simultaneously. I can fix my attention on the distinction that I want to see without destroying the whole.

We might say that the human interpreter has some expectations about architectural drawings, and it is the coupling of these expectations with the current overt patterns of light that determines the final interpretation. The question is, how can we know, by means of which kind of rule, that certain arrays of lines are walls limiting the space which is interpreted as a room? We can advance many rules for concrete situations, but how can we guarantee that they will be of any use when the context changes? We do not know the answers to these questions in relation to the human visual perceptual process. However, an explicit notion of linguistic interpretation of computer graphics allows us to establish useful conventions for the definition and use of graphical symbols that emerge from a graphical context. Furthermore, without an explicit notion of interpretation these relations cannot be made explicit, because the same set of lines in a different context might mean different things, or nothing.

There is also a set of interesting questions related to the process of identification of graphical objects. Consider that graphical symbols can be defined and selected on the screen by means of a pick or a locator device without an apparent or problematic ambiguity. Symbols are selected from graphical menus and drawings are made out of those symbols. They can be updated by changing their position, their parametric description, or they can be translated, rotated or scaled in relation to some spatial definition. However, consider the identification of emerging graphical objects on the screen of a computer graphics device. Is it possible to point to the house without pointing to some room or some wall? How can we tell what object is being pointed out by an individual pointing action? Is the purpose of a pointing action to identify a geometrical object, or rather the thing which such a symbol stands for? If there is no notion of interpretation, there is no conceptual ambiguity in the pointing act. Objects are found by searching the geometrical information stored in the memory of the computer. There might be geometrical ambiguity, as when we point to the intersection of two lines, but it might be solved by implementational consideration as, for instance, by given a larger value of priority to the graphical segment in which the selected line is stored, but we are rather concerned with conceptual ambiguity. If there is an explicit notion of interpretation the ambiguity arises, and the pointing act has to be made in the context of the meaning of the drawing, and that context determines the entity that is being referred to. We can live with the old drafting and CAD systems, but we have to be aware that when they are used, the human-users are the ones making the meaningful work. I believe that the price of introducing an explicit notion of linguistic interpretation in traditional computer graphics systems is worth paying, because this knowledge can be used in the solution of problems far beyond the scope of traditional systems, and also because we will become more aware of the things we do when we are engaged in an interactive conversation with a computer system.

1.1. Computer Graphics and Natural Language Processing.

In this dissertation an interactive graphics and natural language program called GRAFLOG is presented (Pineda 1988a, 1988b, 1988c, 1988d, 1988e, 1989). The program has two major components: an 'intelligent' interface and a design modelling system.

The interface is a traditional graphical interactive environment that has been augmented with a natural language processing facility. These two functionalities are integrated, and the graphical and linguistic interactive dialogue proceeds with common discourse referents and context. The linguistic component of GRAFLOG can be regarded as an application of the theoretical studies and computational linguistic tools that were developed for a SPRIT project called ACORD (Klein 1987). In the graphics component, however, we depart from previous work and we emphasise not only the notion of graphical communication but also the representational aspect of drawings.

The design system is a geometric modelling system for the representation and transformation of 2-dimensional wire-frame diagrams. This kind of drawing is common in architecture and many other design domains. We first study the interfacing problem of how to define an arbitrary set of conceptual, geometrical and topological constraints that a drawing must satisfy in the course of a design task. For this, the rich expressive power of natural language is added to traditional graphical interaction facilities. Natural language is used for imposing a semantic interpretation, a conventional meaning, on the graphical symbols and relations making up a drawing. Then, we address the problem of modifying such a drawing when one or more properties of its constituent symbols are altered by a human designer in the course of a graphics interactive session. Geometric models and graphical design systems can be traced back to the origins of interactive computer graphics with Sutherland's Sketchpad program (Sutherland 1963). The novelty in this thesis is, perhaps, that these issues are addressed on a semantic basis. In addition to the geometrical and topological information that is needed for the production of a drawing, we modify a drawing in relation to its meaning.

The interrelationship between computer graphics and natural language has been subject of a relatively small number of studies. There are, however, a number of systems that have introduced key ideas for the subject. Here, we concentrate in the discussion of a small but representative sample of these programs. The first program that we consider is Winograd's SHRDLU program (Winograd 1972). It was one of the earliest natural language systems and its purpose was to show that the process of understanding natural language can be modelled through an integrated system of procedures that represent syntactic, semantic and pragmatic knowledge about a very constrained domain, in this case, the blocks world. Graphics in this system were brought about in a very indirect way, and there was little discussion of their relevance for the natural language understanding process itself. However, SHRDLU is relevant for our purpose because the objects that are represented in the system, as well as their attributes and relations, have a graphical realisation, and this context was displayed in a computer graphics device. This program showed the feasibility of integration of linguistic and graphical knowledge.

The first attempt to integrate natural language facilities in a graphics interactive environment was, to my knowledge, Brown and Chandrasekran's Picture Production system (Brown et al 1981, p. 176). These authors define a natural language and graphics system as follows:

A Natural Language and Graphics system is a computer programming system with both natural

> language input and output, and graphics input and output. It is intended that a complete NLG system should assist the user by behaving in an 'intelligent' fashion whenever possible, by 'understanding' natural language input, by having extensive knowledge about the objects to be displayed, by providing defaults and reasonable values if they are not provided by the user, by assisting the user with problem-solving capabilities when appropriate, by having the ability to answer questions, and by responding in both natural language and pictures.

In this system, some examples of the kind of natural language dialogue that might be relevant in computer graphics are presented, in particular questions and commands, and the role of knowledge representation and inference schemes in a graphical oriented dialogue is studied. In this work, of particular interest is the discussion of the kind of ambiguities that can occur if semantic distinctions are not properly accounted for. Consider that a natural language dialogue might be about the objects represented through the graphics or about the graphical symbols themselves. Consider as well that these entities occur in a conceptual world, but also within the screen of a computer device in which a number of abstract and physical spaces and transformations can be active, and the natural language dialogue can have interwined referents in these different contexts.

Brown's system has deictic capabilities by which the referents of natural language terms are pointed out in the graphical representation. However, the role of deixis as one of the predominant features of a graphics interactive environment was introduced in Bolt's *Put That There* program (Bolt 1980). In this system demonstrative words like *this* and *there* were coupled directly with conceptual objects represented through graphical symbols, and with a conceptual space in which the spatial referents can be identified through natural language deictic expressions. This feature was developed, however, to support the integration of graphical interaction and voice recognition facilities, and the semantic implications of the meaning of graphics whose interpretation is introduced through ostension and deixis were not fully followed up.

The development of integrated graphical and linguistic interfaces could not be expected independently of the progress made in the fields of theoretical and computational linguistics. None of the three systems mentioned above had a strong syntactic and semantic foundation for the natural language component. Even Winograd's system, impressive as it is, was developed on a poor syntactic and semantic basis. It success was due, in my view, to Winograd's conception of language. He thought of human language as a process in which the overt linguistic capability proceeds along with behaviour. Language takes place in a context and it is about the interaction of the individual who uses this capability and his environment. Ten years of research was necessary for to develope theoretical and computation environments in which the syntax and semantics for natural language processing could be understood on more solid grounds. Of particular relevance for this work are the development of unification based theories of grammar (Klein 1988a). in theoretical linguistics and the production of logic programming based computational tools for implementing and testing natural language grammars, such as the PATR formalism (Shieber et al 1983) It is in the context of these theories and techniques that the ACORD system (Klein 1987) was developed.

ACORD is a large natural language and graphics system for the creation, query and update of knowledge-bases. It has five major conceptual components: the Dialogue Manager, the Graphics System, the Parsers, the Knowledge Base, and the Text Generators. The theoretical emphasis of this system is, however, on the syntax and semantics of natural language, and the other components of the system have a subordinated role. Graphics, in particular, is used mainly for visualisation purposes and it does not play a significant role in the representational environment. Nevertheless, the system makes an explicit use of deictic expressions linking the graphical symbols with their corresponding linguistic identifiers, and there is a direct relationship between the graphics and the knowledge-base by which graphical events have an overt semantic interpretation. The ACORD system sets the context in which GRAFLOG was developed.

The starting point of GRAFLOG was that ostensive definitions and deictic expressions impose an interpretation, a conventional meaning, on the graphical symbols that are realised on the screen in a computer graphics environment. This notion is developed in Chapter 2 and it leads naturally to the notion of graphical language. The key ideas of this kind of interaction are introduced with the help of a simple example. In the second part of Chapter 2, the conceptual architecture of GRAFLOG is presented. Chapter 3 and 4 are devoted to explaining the linguistic component of the system. In Chapter 3, a brief introduction to modern theoretical and computational linguistic theories and techniques is presented. This chapter is included solely for the purpose of making this dissertation a self contained work. In Chapter 4, a theoretical and computational analysis of the kind of natural language dialogue that is supported in GRAFLOG is presented.

One of the main aims of this research was to understand the role of graphics as a representational environment. Graphics are commonly used for 'visualising' knowledge. However, the value of graphics depends not only on the amount of information that it conveys, but also on the nature of the inference processes that are used in its interpretation. In Chapter 5, an attempt to shed some light on the structure of graphics and graphical inferences is presented. Graphics are defined as an algebraic system in which the graphical symbols are the elements of the carrier, and geometrical and topological relations are the operations of this algebra. This structure is then embedded into a modal first order language in which not only the semantics of graphics, but also of natural language expressions can be represented in an integrated fashion. Finally, in this chapter, a notion of graphical inference is introduced.

In Chapters 6, 7 and 8 the theory presented in Chapter 5 is applied to modelling design problems in the 2-dimensional wire-frame design domain. In Chapter 6, a graphical and logical language for this design domain is introduced. We show its syntactic definition and semantic interpretation. In Chapter 7, we introduce a notion of design concept and design intention. We show how design concepts can be represented in a declarative fashion. Then the notion of design intention is introduced. In GRAFLOG, design intentions are manifested by standard graphical interaction. A graphical input event presupposes the intention of a human designer to produce a graphical representation of the design object, that is like the current drawing but in which one or more graphical properties are different. The job of the system is to infer such an intention, to deduce a plan for its satisfaction, and to apply such a plan to the current drawing to produce the new drawing that is intended by the human designer. In Chapter 8, we present such a planning function for our simple design domain. We also show that this is not a deterministic process, and that in general, design problems can have a large number of solutions. For that reason, the solution that is produced by the system must not only be constrained by the interpretation of a drawing: it must also be intuitive for the human designer. This additional requirement implies that the rule that produces a design object is constrained by an interpretation, but is not fully determined by it. Such a rule must also be a model of a design skill: the design behaviour itself, which is embedded in the interpretation process. These issues are very complex, and we leave the discussion on the presuppositions and consequences of our approach to computer graphics interaction and design, to the concluding chapter of this dissertation. For the moment, we come to the 'hard' material, and we start the presentation of GRAFLOG.

Chapter 2

GRAFLOG: A Graphical Language for Human-Computer Interaction

In this chapter, a notion of graphical language for human-computer interface is introduced. This notion is illustrated by means of a computer program called GRAFLOG (Pineda 1988a, 1988b, 1988c, 1988d, 1988e, 1989). The program was developed for the purpose of this dissertation, and it is a graphical and linguistic interpreter written in PROLOG (Clocksin 1981). Among the characteristics of the system is a facility for stating a linguistic interpretation for graphical symbols and relations. Such an interpretation is useful for expressing knowledge through graphical representations. The set of conventions by which drawings are interpreted defines, in an implicit way, a graphical system of communication: a graphical language. In GRAFLOG, the interpretation of drawings can be queried in both the linguistic and graphical domains.

In the current prototype implementation, the linguistic fragment covers a limited set of ostensive definitions and questions, and most of the linguistic interpretations of drawings are introduced through Prolog expressions. However, in the text of this dissertation, a theoretical analysis for implementing a rather strong linguistic component has been developed. In Section 2.1 a graphical and linguistic dialogue for stating the interpretation of drawings is shown, and a notion of graphical language is introduced. In Section 2.2, a general description of the system is presented.

2.1. A Graphical and Linguistic Dialogue.

In GRAFLOG we can type the ostensive definition

This is Luis

at the time a graphical symbol (i.e. a smiling face) is chosen from the graphical menu and placed somewhere in the screen. The drawing in Figure 2.1. is created by editing the graphical symbols at the time their corresponding linguistic interpretations are stated by ostension.

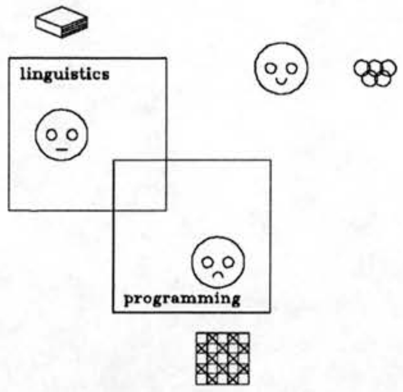

FIGURE 2.1

The 'serious' face stands for John, the 'sad' one stands for Pete and their interpretations are introduced by the expressions *This is John* and *This is Pete.* The rectangles refer to linguistics and programming --the subjects of study-- and their interpretations are introduced in similar fashion. The labels in the rectangles are shown for identification purposes, but these names are not required given that their interpretations are fixed by ostension at the time their referents are pointed out, and we --the users of the system-- can remember these interpretations. From the point of view of the program the rectangles are two independent symbols.

The graphical symbol depicting a book refers to a book --any as yet unspecified book-- the olympics games logo refers to a game --any game-- and their interpretations are stated by the expressions *This is a book* and *This is a game.* The chess board refers to chess --the game of chess-- and it can be introduced by typing the sentences *This is chess, it is a game.*

In Figure 2.2 the relationship between graphical and linguistic symbols is illustrated. The name *Luis* is a linguistic symbol, and the face is its corresponding graphical representation. The horizontal arrow represents a dimension of correlation or translation between the linguistic and graphical symbols, and both of them refer to the same individual who is the person in the world whose name is Luis.

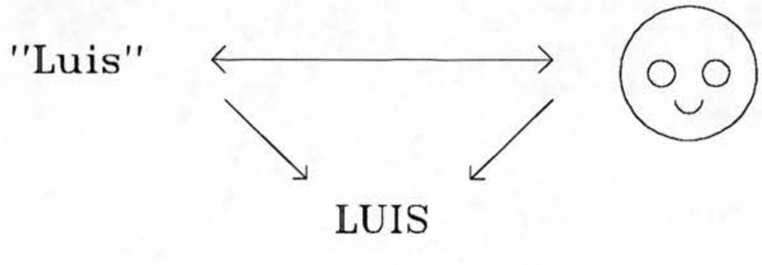

FIGURE 2.2

Other symbols denote some non-specified entity, as for instance the drawing of a book; such a symbol denotes an individual in the set of all books. The drawing of a book refers to that individual book, but not to all the members of the set of books, nor to the set itself.

When the graphical symbols are introduced their corresponding geometrical descriptions are asserted in the system. Each graphical symbol is specified in a geometrical data-base (*g_db*) in which the symbol's name, i.e. *luis*, is an index to the relevant geometrical information which is required for drawing the symbol on the screen. The type of the symbol is also recorded in *g_db*. The type of the symbol standing for Luis is *face*. Up to this point, the graphic interaction is a standard direct manipulation procedure. When the message has been stated, the representation has to be produced. The following relation is the entry for Luis in *g_db*

g_db(luis, face, PARAMETERS).

where the variable *PARAMETERS* contains the description of the particular symbol instance.

The linguistic knowledge conveyed by the ostensive definitions is asserted in the system's natural language knowledge base *NLKB*. These expressions are of the form *this is np*. The word *this* is a demonstrative pronoun whose function is to support the physical pointing gesture; the verb *is* has the function, in these expressions, of establishing a relation between whatever is being demonstrated by the word *this* and the individual of whom a name or a description is given. The variable *np* stands for a noun phrase through which a description of an individual is given. The *np*'s in expressions of the form *This is np* can be proper nouns as in *This is Luis*, or descriptions that refer to individuals by common nouns as in *this is a book.*

Specific individuals that are introduced by a proper noun are known by the system by that name. On the other hand, if an individual is only referred to by a common noun, or by a description given in terms of common nouns, GRAFLOG produces a name to refer specifically to this individual in the linguistic domain. This name is an internal identifier used by the system and it is never visible by the human user. This identifier is also used as an index to *g_db* where the graphical representation of the individual referred to by that name is stored. Here, a notational convention for this Section is introduced. Linguistic knowledge is represented by means of expressions of the first order logical language (FOL). Graphical information, on the other hand, is expressed as PROLOG relations. For instance, when the expression *this is a book* is typed in at the time the corresponding symbol is selected from the graphical menu, the system produces the internal name *book_1* and asserts the fact

book(book_1).

in *NLKB* and makes the entry

g_db(book_1, book, PARAMETERS).

in *g_db*. That is, there is an individual which is known to the system by the name *book_1* and which happens to have the property of being a book.

Now, we have to consider the question of what is the linguistic knowledge that has been communicated by means of these expressions. Of the individuals referred to by means of proper nouns nothing has being said. They have been demonstrated, but no property of any of them has been mentioned. On the other hand, descriptions assert a property that some specific or some non-specific individual has. Then, the current linguistic knowledge asserted in *NLKB* is,

book(book_1).
game(game_1).
game(chess).

Now, we can select the graphical symbol standing for Luis at the time the expression *this is happy* is typed in. This ostensive expression asserts a property that the individual referred to by this symbol has: the property of being happy. The word *happy* is an adjective that has the function of predicating a property of the individual whose name is *Luis*, and *NLKB* is updated with the fact,

happy(luis).

Note that this is the only instance in *NLKB* in which a reference to Luis is made, and before this expression was typed in, the system knew only about his name and his 'image'. Before the proposition that *Luis is happy* was asserted, the individual Luis could only be referred to from language for identification purposes, in virtue of grammatical knowledge. For example, we could ask the system *is this Luis?* at the time the graphical symbol was pointed out on the screen, and the system would produce the answer *yes*; and if another symbol --like one of the other faces-- were pointed out at the time the same question were asked the system would have answered *no*.

At this point we can assert some additional information about the individuals that are graphically depicted. We can, for instance, point to the chess board at the same time as we enter the sentence *Pete plays this game*. Through this expression, a relation between the individual depicted by the sad face --Pete-- and the abstract individual depicted by the chess-board --the game of chess-- is asserted in the representational system.

plays(pete, chess).

Now we can type the expression,

These are students

while the three faces are pointed out on the screen. In the same way, we can type the expression *These are subjects* at the time the symbols standing for programming and linguistics are selected on the screen. This expression is of the form *These are np*. The word *these* is a plural demonstrative pronoun, and *students* is a plural common noun. Note that these pointing actions are performed under the context set by a plural ostensive expression, and as a consequence, several similar pointing actions can be performed until that linguistic context is closed. Here no new graphical symbol is introduced, and the graphical representation remains the same. The purpose of these new declarative statements is to assert that the individuals graphically referred to by the faces are students, and the ones referred to by rectangles are subjects. The following facts are added in *NLKB*,

```
student(luis).
student(john).
student(pete).
subject(linguistics).
subject(programming).
```

Here, plural expressions are modelled as sets of singular statements. This is a conventional decision for implementational purposes.

At the current interactive state, a graphical representation with its corresponding linguistic interpretation has been introduced. This interpretation can be questioned for identification purposes. We can ask, for instance, *what is this?* in conjunction with a pointing act, and the system produces a linguistic description of the individual referred to in the graphical domain. If the smiling face in Figure 2.1 is pointed out at the time *what is this?* is typed in, the system will produce the answer *That is Luis*. Here, the answer is an expression of the form *That is np*, in which *np* is the name of the individual identified in the graphical domain. The word *that* is a demonstrative pronoun, like *this*, but implies a different relative spatial relation between the individual making the demonstration, on the one hand, and the object being demonstrated, on the other. The word *this* implies 'proximate', and the word *that* implies 'remote'. We take the convention that while the human-user stands in a 'proximate' spatial relation to the screen, the system stands in a 'remote' relation. This convention allows us to model a natural human-like dialogue with the egocentric reference centered in the human-user.

Now suppose that we point to the graphical symbol standing for John in Figure 2.1 (the serious face) at the time same question --*What is this?*-- is asked. Here, there is a referential ambiguity. In terms of the geometrical information --from the point of view of the graphical interpreter-- the referent could be either John or linguistics. The interpreter could just produce the names of all the individuals that are geometrically satisfied and the answer would be *John or programming*. However, the form of the expression is singular --*what is this?*-- and certainly this is an indication that we are looking for an individual and not for a collection of entities. In this example, there might be a very strong feeling for selecting john, but the problem for us is how can we know that. We can define some heuristics as, for instance, return the more specific entity in terms of the geometrical information, or we could tell to the human-user --as in many graphical editors-- point to a line unambiguously defining the symbol that you mean, but for the moment we reject these sorts of heuristics. The reason is that the referent is not determined by the drawing, by the physical substance of the representation, but in virtue of the meaning. We would like to know which one is the individual intended by the human-user asking the question. Furthermore, heuristics based in geometrical information would presuppose that the internal geometrical unit of the symbol is defined in advance, out of the graphical context. This is the case in the present example, but this assumption does not always hold. The assumption would not hold for drawings in which graphical entities emerge from the context established by some other overt symbols; as for instance, the rooms of a house in an architectural drawing 'emerge' from the lines representing walls. In such a context, a heuristic for selecting the more specific graphical symbols, the rooms, would prevent us from referring to the less specific --the house-- at all.

Some of these ambiguities can be handled by asking more informative questions. Users of GRAFLOG might type, for instance, *who is this?* or *which subject is this?* Given the same graphical reference --the symbol standing for John-- the former question is answered as *This is John* and the latter as *This is linguistics*. Here, the *wh*-words have the function of identifying the kind of entity demonstrated by the word *this*. The word *who* imposes a restriction on the description: it has to designate an animate entity, and for that reason the reference *linguistics* is ruled out. The fact that John is a human animate entity --the sort of entity that John is-- is stored in the lexicon. The fact that John is a student is a very contingent property of the individual John, and it is irrelevant for answering this question. In the second expression the constituent *which subject* introduces, in an explicit way, a property that the individual that is referred to has to have, and it is translated into the representational system as

$$?\exists x.\mathbf{subject}(x)$$

For satisfying this formula --for answering the question-- the fact

subject(linguistics).

has to be in *NLKB* as well.

Here, we open a small parenthesis. Although questions such as *who is this?* or *which subject is this?* provide grammatical clues for selecting the individuals that are pointed out on the screen, there might still remain more than one possible referent. Furthermore, even if there is no referential ambiguity there might be several alternative descriptions of the individual whose identification is required. If the referent has a proper name, then he or she can be referred to by that name, but another possibility is to produce a pronoun in the answer in order to make a more natural reference. On the other hand, if the individual referred to has no proper name, a description of such an entity has to be produced. But there might be as many descriptions of that individual as the number of properties or relations that it has or it stands in with other individuals in the representation. Besides, descriptions can be definite or indefinite according to what is considered relevant in the context of the conversation. Answering a question is, as can be seen, a rather complex process. The answers are produced not only by taking the conversational context and the discourse focus into account, but also by considering the intention or purpose for providing such an answer in relation to a problem to be solved. Here, there is no intention to provide a theory for such a process, and a very simple model for dealing with questions is shown.

Here, we come back to the interactive conversation with GRAFLOG. At this point we can introduce a more complex interpretation. For instance, we can type the following set of expressions,

If a game is to the right of Luis then he wins it.
If a book is above John then he reads it.
If a student is in a subject then he studies that subject.

These expressions are parsed and their semantic representation are

$$\forall x.\mathbf{game}(x) \wedge \mathbf{right\_of}(x, \mathbf{luis}) \rightarrow \mathbf{wins}(\mathbf{luis}, x).$$
$$\forall x.\mathbf{book}(x) \wedge \mathbf{above}(x, \mathbf{john}) \rightarrow \mathbf{reads}(\mathbf{john}, x).$$
$$\forall x,y.\mathbf{student}(x) \wedge \mathbf{subject}(y) \wedge \mathbf{in}(x, y) \rightarrow \mathbf{studies}(x, y).$$

and these expressions are asserted in *NLKB*.

The spatial prepositions *in, above, below, right of* have a very special status within the representational system. In the linguistic domain, they are represented as normal entries in the lexicon, and they are interpreted according to the rules of the grammar. In the graphical domain, on the other hand, they have a conventional geometrical interpretation. The linguistic knowledge is tied to the graphics through a set of basic geometrical predicates that have been given in advance to the system. The graphical interpreter takes these terms as primitive functions that are computed by standard computational geometry algorithms.

The conventions for interpreting geometrical relations are shown in Figure 2.3. For our purpose the geometrical conditions for the prepositions *right of, above* and *below* are satisfied if there is a line intersecting both graphical symbols oriented according to the corresponding relation. The preposition *in* denotes a relation of total inclusion between two graphical symbols.

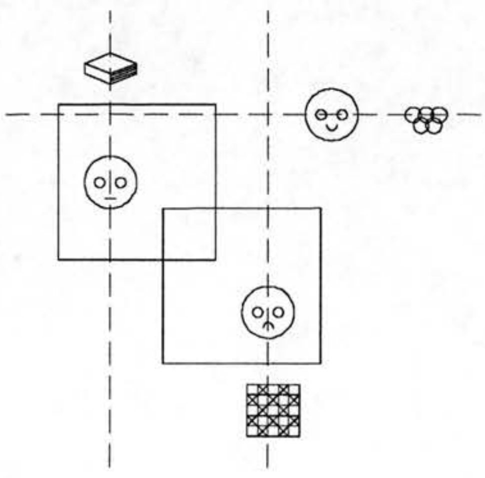

FIGURE 2.3

Once this more complex interpretation have been defined it can queried. For instance, looking at Figure 2.3 the user can ask,

Who studies linguistics?

The question is translated into its corresponding representation as

$?\exists x.studies(x, linguistics).$

and it can be satisfied in *NLKB* in terms of the rule,

$\forall x,y.student(x) \wedge subject(y) \wedge in(x, y) \rightarrow studies(x, y).$

The variable *y* is assigned to linguistics and *x* is assigned to John. The clause *in(x, y)* is satisfied by the geometrical processor in terms of the geometrical information stored in *g_kb*. Other interpretations can be queried in similar fashion.

Now we could edit the drawing by direct manipulation to change, for instance, the position of the olympics-logo, producing the graphical representation shown in Figure 2.4

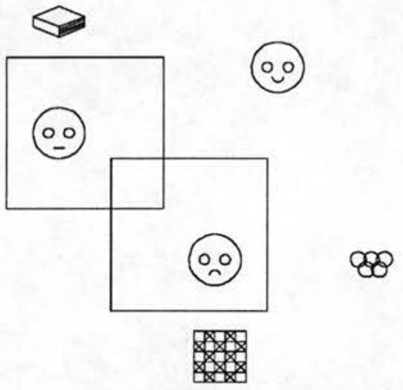

FIGURE 2.4

In this state we could repeat the question *Who wins a game?* and the system would answer *no one*. Note that the olympics-logo is to the right of the symbol standing for Pete, but under the current interpretation *Luis wins a game* is a meaningful graphical expression if the olympics-logo is to the right of the smiling face. This interpretive rule is specific to Luis. However, there might be the strong feeling that we could make an induction in terms of which a general rule, as the one for *studies*, is defined. Of course, making an induction from the knowledge of one particular case is a very weak inference; however, to my surprise, this example has been presented in a number of informal talks, and people have very often interpreted the drawings in Figure 2.4 as saying, in addition to the facts that have been explicitly asserted, that Pete wins a game. After all, we would like to use graphics as a language in which each drawing is an expression referring to some state of affairs, and which we can understand if we know the interpretation. That is, if we know the language. Although the study of conditions under which inductive rules are produced from graphical representations is a very interesting task, the aim in GRAFLOG is to make explicit the interpretation that human-users impose upon drawings, and we restrict ourselves to this more modest enterprise. We can type, for instance, the more general rule

If a game is to the right of a student then he wins it.

This rule would override the previous specific interpretation, and is represented as,

$$\forall \mathbf{x,y.game(x)} \wedge \mathbf{student(y)} \wedge \mathbf{right\_of(x, y)} \rightarrow \mathbf{wins(y, x).}$$

At the current state we can ask the same question --*who wins a game?*-- and the system would produce the answer *Pete.*

We can generalise not just one, but other interpretive rules that were originally defined in particular terms, and their interpretation would be,

$$\forall \mathbf{x.y.student(x)} \wedge \mathbf{book(y)} \wedge \mathbf{above(y, x)} \rightarrow \mathbf{reads(x, y).}$$
$$\forall \mathbf{x.student(x)} \wedge \mathbf{below(chess, x)} \rightarrow \mathbf{plays(x, chess).}$$

Note that the second rule refers to students in a general way, but it is specific to the game of chess.

Now, we can say that a graphical language has been defined. We can produce a set of expressions that can not only be understood by human-users but also interpreted by the system. We can produce, for instance, the drawing in Figure 2.5

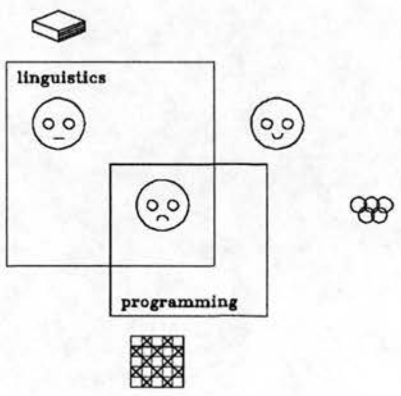

FIGURE 2.5

And it would be interpreted as saying,

John studies Linguistics.
John reads a book.
Pete plays chess
Pete wins a game.
Pete studies programming.
Pete studies linguistics.

Nothing prevent us to impose more complex interpretation upon the drawing. We can type, for instance, *If a student studies linguistics and programming then he is clever.*

This expression can be expressed in the representational system as

$$\forall x.\mathbf{student}(x) \wedge \mathbf{studies}(x, \mathbf{linguistics}) \wedge \mathbf{studies}(x, \mathbf{programming}) \rightarrow \mathbf{clever}(x).$$

Then we can ask,

Who is clever?

The question is translated as,

$$?\exists x.\mathbf{clever}(x).$$

and the answer would have to be *Pete*.

The expressive power of the graphics can be best appreciated if we modify the graphical representation for producing the drawing in Figure 2.6. Note that the symbol standing for a game is to the right of the symbols standing for both Luis and John. This graphical expression is telling, among other things *Luis wins a game* and also *John wins a game*. And we have to pay no price for expressing this additional knowledge. This point can be emphasised by noticing that it also expresses that Luis plays chess.

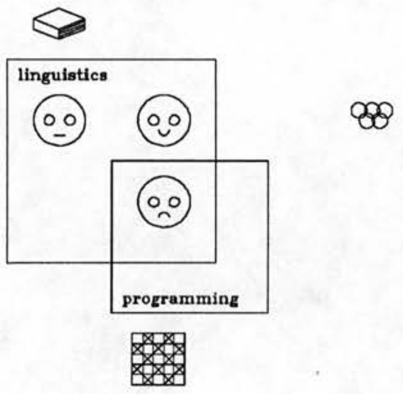

FIGURE 2.6

Under this interpretation, and with the current graphical symbols, an infinite number of drawings can be produced and understood, but many of them would express similar things. However, what is relevant for our purposes is that there is a set of drawings expressing nothing, a set of drawings expressing atomic propositions, and sets of drawings expressing the conjunction of two or more atomic propositions. Through the systematic interpretation of drawings, a number of different things can be said, and then we can convey factual knowledge by means of graphical communication. Furthermore, we can define new graphical symbols standing for specific individuals and this augments the knowledge expressible in the language. We can also generate duplicated instances of graphical symbols referring to the same individuals to avoid geometrical restrictions, and also change or augment the interpretation for defining, throughout the graphics interactive session, new graphical languages.

Before concluding this section, a comment on the interpretation of negation is worthwhile. If looking at Figure 2.6 --with the current interpretation-- the question *Does Luis read a book?* is asked, the answer produced by the system would be *no*. However, consider that the interpretative rule that has been used for answering this question is

(1) *If a book is above a student, then he reads it.*

Or has it? This rule does not assert or deny anything in relation to books which are not above students; nevertheless, the system assumes that Luis is not reading a book, and this is so because it is defined under the so-called closed-world assumption (Reiter 1985). This assumption can be paraphrased as "if you do not know something, assume that it is false". In our system, the assumption follows from the Prolog implementation, but it can be supported independently. A different way to present this feature is by saying that the system takes a default value for answering questions involving incomplete knowledge. The interesting thing about graphical representations is that there is a very strong intuition in regarding such defaults as correct, although there is no apparent reason for this. The closed-world assumption will be used throughout this dissertation, unless it is explicitly mentioned. This is an economic consideration, because this assumption is a very good heuristic, and as we have said, very intuitive in relation to graphics.

An interesting question is at what extend this assumption is needed here. If we change the interpretation and make explicit exclusion conditions, the graphical representations would express complete knowledge. For instance, instead of the current interpretation rule we could state,

(2) *If a book is above a student, then he reads it else he does not*

and the answer to questions relating books with students would follow as a consequence of the interpretation, although the graphics would not be changed. However, in such a scheme we could not express incomplete knowledge; we would have to know whether the propositions asserted through graphics are true or false, and this might turn out to be quite rigid. We could also express incomplete knowledge by stating (1), and complete knowledge by stating (2); however, in such an scheme, the inferences performed by the system would be the same as the ones produced by relying only in the closed-world assumption, but with some additional computational effort. Furthermore, we would not be able to know if the system answers *no* because it knows, or just because it does not know. Another possibility is to set the system to answer *no* if a rule like (2) is used in an inference process, and to answer *I don't know* if the answer comes from the closed world assumption. However, in such a scheme the system might keep answering *I don't know* most of the time, and that should not be a feature of an user-friendly system. Negation is a complex issue; it is not clear what do we mean when we say *no*.

2.2. The Conceptual Model of GRAFLOG.

The conceptual model of the system is thought of in terms of a conversation between two main processes, the human-user *HU* and the system itself, in which the former has the conversational initiative. In Figure 2.7 this model is illustrated. Here, the human-user is considered an active part of the system, but of course is not part of the program.

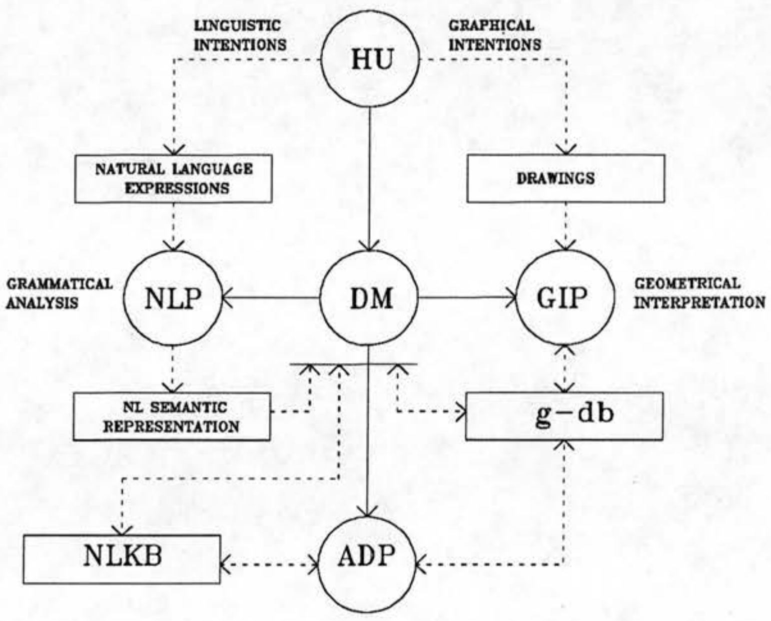

FIGURE 2.7

The system is organised in terms of four independent processes: the dialogue manager *DM*, the natural language parser NLP, the graphics interpretation process *GIP* and the automatic design process *ADP*. In Figure 2.7, processes are illustrated by circles, and the control discipline is represented by continuous directed lines. The direction of the arrows stands for control hierarchy; for instance, the human-user *HU* 'controls' the dialogue manager, which in turns controls the other three processes. The controlling process initiates tasks by sending a message to one of its subordinated processes, and when the task is accomplished the control is returned to the original process. In order to achieve the interactive dialogue, there is a fixed protocol for the main control cycle, as shown in Figure 2.8

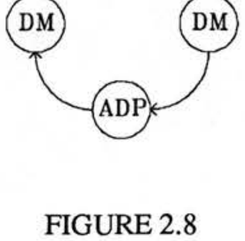

FIGURE 2.8

The blocks in Figure 2.7 stand for representational structures, and the dotted lines for information flow. Human users express linguistic intentions by typing in natural language expressions or by making drawings on the screen. Linguistic and graphical intentions are expressed as messages from the human-user to the dialogue manager, which in turns sends a message to *NLP* or *GIP* respectively. The linguistic expressions are analysed by the parser *NLP* and a semantic representation for every natural language expression is produced. The natural language semantic representation is a very rich structure which contains not only the logical form of the linguistic expressions, but also discourse information for maintaining the current conversational focus, and for solving anaphoric references. The natural language knowledge-base *NLKB*, on the other hand, contains only a representation of the logical facts and relations asserted through the linguistic dialogue, and for our purpose a simple Prolog data-base is used. The process that is responsible for making the decision of which facts and relations are stored in *NLKB* is the dialogue manager *DM*, as will be explained below in this Chapter. *DM* is also responsible for answering questions, and there is a simple theorem-prover defined within this process.

The graphical interpretation process *GIP* is responsible for interpreting the graphical input. When it receives a message from *DM* it has to prompt the human user for the definition of one of more graphical symbols. This process is also responsible for updating the geometrical data-base *g_db*. The automatic design process *ADP* is the user's application, and it also runs under the supervision of *DM*. The need of such a process is not shown in the dialogue of Section 2.1. However, it is the system's component responsible for solving problems. The application that is developed in this dissertation is a design interpreter for geometric modelling. The system's task is to understand and satisfy design intentions in the 2-dimensional wire-frame geometric modelling domain. Although the task chosen is very specific, it can be useful for architectural design and some related application domains, and the solution presented is very general, as will be shown in Chapters 6, 7 and 8.

The grammatical formalism in which the linguistic component of the system is defined is Unification Categorial Grammar (Klein 1988b, Zeevat 1986a) and the parsing strategy and procedures are taken from the UCG interpreter developed for the ESPRIT ACORD project (Calder et al 1986). The output of the parsing process is an expression of a language called Indexed Language (*InL*) (Zeevat 1986b). However, two features within *InL* for handling the particular requirements of the graphical interaction in GRAFLOG have been defined. The first is the inclusion of a special predicate, namely *deictic* for dealing with the deictic --or pointing-- use of the demonstrative pronouns *this* and *these*. The deictic predicate is part of the semantics of these words. The second is the addition of a special entry for the verb *to be* within the lexicon. The verb *to be* is traditionally analysed in terms of two functions, namely, to establish a relation of identity between two entities, as in *Pete is the student who plays chess* or to make an attribution as in *Luis is happy*. This 'third' *is* works in conjunction with the demonstrative pronouns --when they are used in the deictic mode-- and means *translates the graphical representation of an individual into a linguistic symbol.* For instance, when an ostensive definition is stated as in *This is Luis*, the individual referred to by the word *this* is identified in the graphical domain --by means of the deixis. The word *Luis* is a symbol of the linguistic system which denotes the same individual. Then, the word *is* stands for a 'translation relation' between two symbols which are realised in different mediums: graphical and linguistic. In the lexicon, the semantics of the ostensive *is* is defined in terms of the relation *translate*. Here, there is no philosophical or linguistic claim that such a kind of *is* 'exists' in human language; this is just the way the relation between graphical and linguistic symbols is established in GRAFLOG.

In order to show how the dialogue manager works we come back to our original example. At the begining of the conversation the system is set to a 'wait for input' state. It waits for an intention expressed by the human-user. When the user types the sentence *This is Luis* the string is taken by *DM* and it is packed in a message to *NLP*. The string is parsed and an expression of *InL* is sent back to *DM*. The *InL* expression is,

[s][deictic(x), pres(s), translate(s, x, luis)].

The form of this expression is *[index][body]* where the index stands for the topic of the sentence, or in other words the sort of thing that the sentence is about. The 'body' contains a list of atomic expressions, and at this stage we can think of it as the conjunction of formulas contained within the list. The brackets denote the scope of the index. Every atomic expression of *InL* has as index its first parameter; for instance, the index of *translate* is *s*. However, we take the notational convention that these indices are omitted unless they are explicitly required. The sentence *This is Luis* is about an the present state *s* in which an individual is pointed out on the screen, and whose name is *Luis*. The predicate *pre* comes from the meaning of the verb *to be* and it is a semantic marker indicating that the state *s* occurs at the present tense. The predicate *deictic* comes from the meaning of the demonstrative *this*. The argument of the deictic predicate, the variable *x*, stands for such an individual and its value comes from the graphical domain. The predicate *translate*, on the other hand, comes from the meaning of the verb *is* when it is used in an ostensive mode.

At this stage, the dialogue manager interprets the *InL* message in order to find whether there is any deictic reference, as it is the case for the current expression. Now, a message has to be sent to *GIP*. In the body of the message the deictic and the translation information is included. When the message is received, the graphical interpreter assembles and executes a graphics procedure with the information contained in the body of the message. In this case the procedure is of the following form,

deictic(x).

When this procedure is executed, *GIP* prompts the user for the definition of a graphical symbol.

The user has to select a symbol type from the graphical menu and then he or she has to provide its parametric description by standard graphics interaction. In our example, the user selects a symbol of type *face*. The variable *x* becomes the reference to the data-structure in which the geometrical representation of the graphical symbol is stored. Then, *GIP* takes the linguistic name of the symbol form the translation relation, *translate(s, x, luis)*, and assembles the corresponding entry in *g_db*, in which the linguistic name of the symbol is also the index to the data-base. This entry is of the form,

g_db(luis, face, PARAMETERS).

When the graphical interactive task has been completed, the control is returned to *DM*. Before finishing its current task, *DM* has to decide whether there is any linguistic information that has to be stored in *NLKB*. In the current message, there is none. The definition by ostension of John and Pete is processed along similar lines.

The next expression stated in the dialogue in Section 2.1 is *This is a book.* The *InL* interpretation of this sentence is

[s][deictic(x), book(y), pres(s), translate(s, x, y)].

These *InL* expression contains graphical and linguistic information. The graphical information is processed as before, with one exception: the name of the individual referred to in the graphical domain is not known. What has been given is a indefinite description of such an individual. Once the geometrical description of the graphical symbol is provided, *GIP* consults the translation relation in order to find the name of the individual for making the corresponding entry in *g_db*. However, in the current translate relation, there is no such name. Its place in the relation is filled with a variable. Then *GIP* produces an arbitrary identifier, as for instance *book_1*, with the purpose to refer to such an individual, updates this value in the translation relation, and updates *g_db* with the following entry,

g_db(book_1, book, PARAMETERS).

Now, a non-null message has to be sent back to *DM*. The body of such a message contains the original translation relation, but with the new value. The body of the message is of the form,

translate(s, x, book_1).

where the variable *y* has been instantiated with the value *book_1*. Now, *DM* has two expressions at hand, and the variable name provided from the graphical domain can be passed to the anonymous individual in the linguistic domain. The variable *y* in *book(y)* in the current *InL* expression is instantiated to yield the linguistic fact *book(book_1)*. This fact represents useful linguistic knowledge and it is updated in *NLKB*. This is the first fact asserted in *NLKB* in the current dialogue.

The system is able to handle some discourse information. For solving anaphoric references, *InL* expressions have an associated input and output list. The input list contains a set of reference markers that can be used as possible antecedents for anaphoric pronouns in the current sentence. The output list contains the markers that the current linguistic context provides for the following sentence in the dialogue. This technique is called *threading* (Johnson et al 1988) and it is taken from the UCG interpreter of the ACORD system (Calder et al 1986). The *DM* is responsible for solving anaphora by taking information from the input list.

The next expression that was typed in the the dialogue of Section 2.1 is *This is chess it is a game*. These two sentences constitute a simple discourse in which one anaphoric reference has been made. Note that the words *is* of the first and second clauses are different. The first *is* has the function of establishing the trans-substantial relation between the graphical and linguistic symbols standing for chess; the second *is*, on the other hand, has the function of establishing an identity relation within the linguistic system. The first clause introduces the individual whose name is *chess* by ostension, and is processed as was shown above. The second clause, on the other hand, contains the pronoun *it* whose antecedent was introduced in the previous clause. The *InL* translation of *it is a game* is,

[s][pro(neuter(x)), game(y), pres(s), identify(s, x, y)].

where the term *pro(neuter(x))* indicates that the individual *x* was introduced by the neuter pronoun *it*, and the relation *identify* is part of the meaning of the verb *to be* in this context. The reference for the pronoun *it* is, in the present example, chess, and it is found by *DM*. There is no graphical command involved in this expression, and the only thing that has to be done by *DM* at this stage, is to unify the symbol *chess* with the variable *y*, to produce the fact *game(chess)*. This is possible because the things referred to by these two symbols stand in an identity relation as is denoted by the relation *identify*. This fact is asserted in *NLKB*.

The next expression is *This is happy*. It was typed in at the time the symbol standing for Luis was pointed out and the following *InL* expression is produced,

[s][deictic(x), happy(y), pres(s), identify(s, x, y)].

Next, a message with the identification and deictic information is sent to *GIP*. The graphics interpretation process allows the selection of a symbol which is already represented on the screen, and it is identified according to geometrical information. The task of *GIP* is to identify such a symbol, and to pass this information back to *DM*. The message that is sent from *GIP* to *DM* is,

identify(s, x, luis).

where *y* has been instantiated with the constant *luis*. With this information, *DM* is able to produce the fact *happy(luis)*, which is also asserted in *NLKB*.

In the expression *Pete plays this game* the word *this* might function as a determiner but also as a demonstrative. If the referent of *this game* has an antecedent that can be found in terms of the discourse, the word *this* is only a determiner. This would be the case if the current discourse were,

Chess is a game. Pete plays this game

Here the referent of the expression *this game* would be chess. However, if there is no such discourse information, the word *this* plays both roles, and a pointing gesture supporting the expression is required. In any case, the task for the dialogue manager is specified by the *InL* expression. In the lexicon, there are four entries for the word *this*. In the 'pure' demonstrative mode *this* is a noun-phrase which has the deictic predicate specified in its semantics, as in *This is John*. In the determiner-demonstrative mode, *this* is a quantifier which has in its semantics the deictic predicate, as in the present example. The other two *this* have no deictic component. The *np* form is anaphoric, and the determiner form is just an alternative lexical realisation of the word *the*. The *InL* deictic translation of *Pete plays this game* is

[s][[y][deictic(y), game(y)], [s][pres(s), play(s, Pete, y)]].

Here, the graphical command is processed in the manner shown above. The message returned from the *GIP* is *deictic(chess)*. Note that in this example, neither the translate nor the identify predicates are involved. When *DM* receives the message, the fact *game(chess)* and the relation *play(pete, chess)* are produced and asserted in *NLKB*.

The next expression in the dialogue was *These are students*. Suppose that this expression is typed in before the rectangles standing for subjects had been introduced in Figure 2.1 and there is no graphical ambiguity. This is a plural ostensive definition. A expression such as this can be used either to define a set of graphical symbols referring to a set of anonymous individuals --in the same sense that a book was introduced above-- or it can be used to express a property that a set of individuals whose graphical representation is already there have. Simple plural nouns such as *students* and complex or modified plural nouns, such as *clever students*, refer to sets of individuals sharing some common properties. The *InL* representation of the current sentence is

[s][deictic(X), student(Y), pres(s), identify(s, X, Y)].

This expression is very similar to the singular analogous one; however, note the capital letters in the index and argument variable names. These variables stand here not for an individual, but rather for a set of individuals, and they will be instantiated in the graphical processor with a list containing the names of all individuals referred to by such an expression. This expression is processed by *DM* and a message is sent to *GIP*. The graphical processor executes a procedure of the form *deictic(plural(X))* --where the term *plural* indicates the variable type-- and the user is prompted for a sequence of graphical inputs. In our example, the three faces on the screen are selected and, according to our expectations, the variable *X* should take the value *X := [luis, john, pete]*. When the graphical interaction is completed, *GIP* sends a message back to *DM* of the following form,

deictic([luis, john, pete]).

This message is interpreted by *DM* as saying that each of the individuals referred to in the set have the property asserted in the current *InL* expression. The facts *student(luis), student(john)* and *student(pete)* are produced and asserted in *NLKB*, as was shown for the corresponding singular expressions. If the plural ostensive definition were instead *These are clever students*, the corresponding *InL* expression would have been,

[s][deictic(X), clever(Y), student(Y), pres(s), identify(s, X, Y)].

This expression is processed similarly, but the adjective implies that an additional property of each individual referred to by the expression, as *clever(pete)*, would have to be asserted in the natural language knowledge-base as well.

Now, suppose that the plural expression *These are students* is typed in after the rectangles standing for the subjects were introduced as shown in Figure 2.1. Here, there is a geometrical ambiguity. When John and Pete are selected, the rectangles representing the subjects are possible referents as well. According to the convention that graphical symbols are selected in terms of their meaning, there is no way that *GIP* can deal with the ambiguity. The message that the graphical processor would send to *DM* in this ambiguous case is,

deictic([luis, [john, linguistics], [pete, programming]]).

Here, *DM* has to make a descision on what symbols are the intended referents of the pointing acts. This ambiguities are represented by sub-lists within the plural list. The problem here is that there seems to be no general way of making an *apriori* decision for solving this sort of ambiguity. The decision cannot be made on the basis of other properties that the individuals referred to might have, because there might be none known to the system when the current ostension is made. In fact, the purpose of the ostension is to attribute a property to some of these individuals. Here, the human-user is to prompt for confirmation when the ambiguous referents are processed. For instance, when the sub-list *[pete, programming]* is analysed, *DM* takes one by one the symbols in the list and sends a special message to *GIP* with that referent, and the user is prompted by a linguistic expression such as *this one?* supported by a graphical feedback highlighting the corresponding graphical symbol. Once the ambiguity is solved, the system proceeds as was shown for the unambiguous case. Another possibility for dealing with this sort of ambiguity is to give priority to the symbols according to the type; for instance, we could say that symbols of type face have a larger priority value than symbols of type rectangle. An interesting question for human perception is how we can make this sort of discrimination when an object is identified in the visual field.

Interpretative rules like *If a student is in a subject, then he studies that subject* are assertions that have to be added in *NLKB*. When the *InL* representation of a sentence like this is received by *DM*, the grammatical information is removed and its simplified form, as a Prolog clause, is stored in *NLKB*. The *InL* representation of the current sentence is,

[s''][[s][student(x), subject(y), in(x, y), pres(s)]

=> [s'][pro(male(x)), subject(y), pres(s'), study(s', x, y)]].

This formula is of the form *[index][A => B]*, where *A* **and** *B* are in turn *InL* formulas. The symbol => stands for implication. Note that within this formula, no deictic predicate is included and no reference to the graphics has to be made. The predicate *in* has a geometrical interpretation, which is considered by *DM* when a question has to be answered. The grammatical information that can be removed here includes the facts that *s* and *s'* are present states, and that *x* is referred to by a personal pronoun *he.* Redundant information, such as the clause *subject(y)* in the consequent clause, can be filtered out as well. This expression can be represented in Prolog by the clause,

study(X, Y) :- student(X), subject(Y), in(X, Y).

In order to make such a simplification, the interpreter of *DM* has knowledge about the grammar of *InL* and also about the Prolog syntax. It is also provided with some simple heuristics.

The natural language component of this system can be replaced by a simpler language. The interpretation of graphical symbols and relations can be stated directly by means of Prolog expressions --or by expressions of some suitable representational language. Such a strategy was pursued in developing the first prototype of GRAFLOG, in which with the exception of a small set of natural language deictic expressions which are understood by means of a regular-expression analyser, the linguistic dialogue was performed in Prolog. However, using natural language is interesting at least in two very important respects. The first is related to computer technology: it is a matter of human-computer interface. If we want to take computers to people, we have to make programs that understand how human beings do things in the most natural way. Even Prolog, simple as it might be, requires to be learned, and has its idiosyncrasy. Furthermore, a complex application is likely to require a level of representation that Prolog is unable to provide by itself; in particular when it is required to reason about time and causality. This sort of reasoning will be needed when the system is applied to design tasks, as will be shown in Chapters 7 and 8. The second point is a matter of scientific interest. Modern linguistic theories require to be tested by computational devices, and the larger the linguistic fragment that they are able to cover, the better the understanding of human language that is accounted for by these theories.

Now, we come to the handling of questions. There are two kind of question defined in the system: yes-no questions, and wh-questions. The most basic question that can be asked is one of the form *Is this x?* where *x* stands for the name or description of an individual. This question has to be supported by an explicit ostension, and has the purpose of identifying the individual that is pointed out. The answer to such question is either *yes* or *no.* When the user types *Is this Luis* at the time the smiling face is pointed out in Figure 2.1, the answer produced by the system is *yes.* The way this question is processed is as follows. First *DM* takes the input string and it is sent to *NLP*. The sentence is parsed and its corresponding *InL* representation is sent back to *DM*. The message taken by *DM* has a tag identifying the expression as a question. The *InL* representation of *Is this Luis?* is,

[s][deictic(x), pres(s), identify(s, x, luis)].

A graphical reference is required and *GIP* sends a message to *DM* of the following form

deictic(luis).

because the only symbol that is identified by the graphical pointing device is the smiling face. For answering this question, *DM* only has to check whether the identification relation holds, after the graphical reference has been provided.

An example of a *wh*-question is *Who is this?*. If the smiling face is selected at the time this question is asked, the answer provided by the system would have to be *Luis*. The *InL* representation of this sentence is,

[s][deictic(x), who(y), pres(s), identify(s, x, y)].

This question is answered by looking for the value of the variable *y* such that its translation *x* comes from the graphical domain. When the graphical processor is queried, it returns the answer *deictic(luis)*, which is also passed to the variable *y* by means of the identification relation. The answer to the question is produced with the help of the predefined pattern *that is x*, Before the answer is produced, the index sort of the variable *y* is checked in order to verify whether *luis* is a human being or an animate entity. Every index variable of *InL* has an ontological type, and the relations between index variables are codified in a lattice structure, as will be explained in Chapter 3 and 4. It is in this structure where the type of Luis, as a human male individual is stored.

Now, suppose that the same question --*Who is this?*-- is asked when the sad face is pointed out in Figure 2.1. The form of the *InL* expressions is as was shown above, but there is an ambiguity in the graphical domain. In fact, there are two symbols which are a plausible graphical referent. *GIP* cannot decide which is the graphical symbol that has been pointed out, and in the message answering the deictic request, it has to return the list of all possible candidates. The message that *GIP* sends back to *DM* contains such a list, and it is of the following form,

deictic([pete, programming]).

The task for solving the ambiguity resides in *DM*. Here, the answer comes from looking into the ontological types of the candidate referents; Pete is a human being and programming is not. Then, the ambiguity is solved in terms of grammatical knowledge --*who*-- and the answer would have to be *That is Pete*.

Note that for finding the answers to the questions *Is this Luis?* and *Who is this?* the dialogue manager only has to check whether the identification relation in the corresponding *InL* expression holds, and that the ontological types of the individuals referred to by the question agree with the grammatical form of the questions. Note that there is no need to know any of the properties or relations of these individuals stored in *NLKB*. These questions are answered by considering only grammatical knowledge.

Now suppose that at the time the question *What is this?* is typed in, we point to the serious face in Figure 2.1. Here, there is graphical and linguistic ambiguity, and it cannot be solved by the general knowledge contained in *NLKB*. There are two alternative referents, namely linguistics and John. This question is interesting in the sense that the use of the word *what* would suggest that the human-user asking the questions knows that the referent is a non-human entity, but in the graphical domain, on the other hand, there is maybe a strong feeling that the referent should be the face. It is very likely that this sort of questions are never asked. Anyway, the ambiguity can be solved by asking, for instance, *Which subject is this?*. The *InL* representation of this sentence is,

[s][deictic(x), which(y), subject(y), pres(s), identify(s, x, y)].

The graphical interpretation of this question is as was shown before. The message that *GIP* sends to *DM* is

deictic([john, linguistics]).

and the referent coming from the graphical domain has to have the property of being a subject, as specified in the *InL* expression. To answer this question there is the need to prove the formula *subject(x)* in *NLKB*. The theorem is proved by *DM* itself. Here, the fact *subject(linguistics)* is found, and *linguistics* is the only possible assignment for the variable *x* at the current state of *NLKB* which satisfies the formula.

The last kind of question that has to be considered in this Section involves not only the identification of graphical referents but also the interpretation of graphical relations. If the question *Who studies linguistics* is asked --when the graphical representation is as shown in Figure 2.1-- the answer would have to be *John*. The *InL* representation of this question is,

[s][who(x), pres(s), study(s, x, linguistics)].

Here, the problem is to find a suitable satisfaction of the study relation in *NLKB*. This can be done in terms of the formula,

study(x, y) :- student(x), subject(y), in(x, y).

The only special consideration for performing this proof is that there is a predicate, namely *in*, that has an interpretation in the graphical domain. In the process of this proof the assignments *x := john* and *y := linguistics* are found, and the relation *in(john, linguistics)* has to be satisfied. The theorem prover knows that *in* is one of the system's relations that have a geometrical interpretation, and a message is sent to *GIP*. This message is interpreted as an imperative command, and the graphical processor executes a geometrical algorithm for testing the inclusion condition between the graphical symbols referred to in the message. When the computation is performed, *GIP* sends a message back to the theorem prover in which the result of the computation is acknowledged. The message for this case is of the form,

in(john, linguistics) := true.

In this way, the proof is partially performed by means of the graphical processor. The expressions *right_of, above, below* and *in* are geometrical predicates, and they are considered to be true if they satisfy some specified geometrical condition, as was shown in Section 2.1. Note that the process of geometrical interpretation is always directed by the theorem prover, and in the last resort by the natural language. This is in accordance with the principle that the interpretation of graphics takes place always in the context of a meaningful intention.

Here, we conclude the description of GRAFLOG. The main components of the system have been shown, and the way they are interrelated has been explained. However, before closing this Section it is worth noticing that answering a question is not just a matter of satisfying a formula. This is so because questions are always made with the intention of achieving some goal. The questions that have been shown in this dialogue have the purpose of identifying something, but they are, in a sense, very passive. The system has no knowledge of the reason why such questions are made in the first place, and it behaves as a simple data-base system. However, we can do better. As Winograd has pointed out, a computer program cannot deal reasonably with language unless it can also understand the subject that it is discussing (Winograd 1972). In our present dialogue, such a subject matter has not yet been introduced. There is a system component that has not been explained, namely, the automatic design interpreter. The purpose of this process is to make design inferences and to solve design problems. In order to do so, it has to be able to infer design intentions expressed by human-users engaged in design tasks, to understand the interpretation of the current drawing, and also the internal structure of graphical representations. However, before this system component is introduced we have to prepare the ground. In Chapter 3 and 4 the linguistic component of the system is explained. In Chapter 3 a brief introduction to modern trends in theoretical and computational linguistics is presented. In Chapter 4, the particular grammatical formalism used in this dissertation is introduced, and a grammar for covering the dialogue developed in this Chapter is presented in detail. In Chapter 5 and 6 an analysis of the internal structure of graphical representations is developed, and a graphical language for dealing with our design task is presented. In Chapters 7 and 8 to the application process is developed. In Chapter 7 the notions of design concept and design intention are presented, and the way these notions are represented and used in the present scheme is described. In Chapter 8 a function for the satisfaction of design intentions in the domain of 2-dimensional wire-frame diagrams is presented. The relation between the problem solving task, the representational system, and the graphical and linguistic dialogue is also explained. Finally, some concluding remarks are presented in Chapter 9.

Chapter 3

A brief introduction to Computational Linguistics

In this Chapter a brief introduction to modern trends in theoretical and computational linguistics is presented. In Section 3.1, the notion of syntactic structure is introduced. In this Section Categorial Grammar and the so-called unification based grammar frameworks are highly emphasised. The notion of semantic structure is introduced in Section 3.2. The relation between syntax and semantics is discussed, and the so-called rule-to-rule hypothesis is presented. In Section 3.3, an introduction to the computational linguistic tools useful for computer implementations of unification based grammar formalisms is presented. A simple example for introducing the main theoretical and implementational concepts of Unification Categorial Grammar is also developed. In Section 3.4, the role of discourse information and anaphoric resolution is presented. Finally, in Section 3.5 a discussion of other linguistic phenomena, such as subject-verb inversion in question, *wh*-terms movement and lexical operations are discussed, and their treatment within unification based grammatical formalism is sketched.

3.1. Modern Linguistic Theory and Categorial Grammar.

Theoretical linguistics is not a recent field of research. Linguistic studies can be traced back to the beginnings of civilisation. An excellent introduction to the subject is given by Lyons (Lyons 1968). Current linguistic research has been characterised by a formal --mathematical-- approach to account for linguistic phenomena. Modern theories of grammar are systematic --ruled based-- descriptions of natural languages. These theories provide also a computational framework through which human language can be modelled. An overview of current theoretical developments is provided by Klein (Klein 1988a). Probably, the best known modern theory of linguistic structure is Chomsky's Transformational Grammar. According to Chomsky's theory there is in language a basic level of syntactic description, the so-called deep-structures, that are produced by context-free rules. However, there is a second layer of rules for producing the overt linguistic patterns --the surface structure-- and the transformations from deep-to-surface structure are accomplished by means of context-sensitive rules (tree-to-tree transformations). Although this theory provides a formal account of many linguistic phenomena, it does not appeal to the computational linguistic community, mainly because to compute tree-to-tree transformations is computationally very expensive.

Chomsky's claims have been challenged and grammatical formalisms in which the structure of natural language is modelled in terms of context-free rules have been advanced. An important antecedent for these formalisms is Categorial Grammar. Among these theories we can mention functional unification grammar, generalised phrase structure grammar (GPSG), head-driven phrase structure grammar, lexical-functional grammar, the so-called PATR-II framework and unification categorial grammar (UCG). These theories are commonly known as **unification based formalisms** (Klein 1988a).

In Categorial Grammar, there are syntactic categories and simple patterns of combination. Given two expressions α and β, if we can analyse α as a functor from *B* type expressions to *A* type expressions, and if we can analyse β as a *B* type expression, then α and β can combine to make an *A* type expression. The category of α is often denoted as *A/B*. Such a combination rule is denoted as

A/B B -> A

According to this rule the argument term is to the right of the functor term. This is indicated by the direction of the 'slash' sign in the definition of the functor term. The opposite situation in which the argument term is to the left is represented as,

B A\B -> A

We can characterise some of the sentences in the dialogue of Chapter 2. The words *Luis, Pete, progamming, etc.* would be basic terms of category *np*, and the transitive verbs *studies, reads*, etc. would be of category *(S\np)/np*; the type of this category indicates that transitive verbs are functors that take first a *np* to its right to produce an expression of type *S\np* which in turns takes a *np* by its left to produce a sentence. The syntactic analysis of the expression *Luis studies linguistics* would be performed according to the following manipulation: the word *studies* of category *(S\np)/np* takes --to its right-- the word *linguistics* of category *np* to produce the expression *studies linguistics* of category *S\np*. This new expression takes in turn an *np* on the left --*Luis*-- to produce the expression *Luis studies linguistics* of category *S*. The result of this procedure is a well formed sentence.

The analysis of simple sentences, as the one in the example, pose no problems for any grammatical formalism; however, there are many linguistic constructions that are very difficult to capture, and some others that resist all attempted analysis. The important matter for us is that there are many highly developed systems that are able to cover the most common patterns of language, and those patterns can be modelled with current computational technology. The question of to what extent some or another grammatical theory captures the structure of human language needs not to bother us. It is an important question, but it does not have to be raised here.

3.2. The Notion of Semantic Structure.

The set of expressions of a language is sometimes referred to as the plane of expression and their meanings belong to the plane of content (Lyons 1968). The units of the plane of expression are symbols and complex structures, as sentences or drawings. The things or state of affairs denoted by symbols, and the truth value of sentences belong to the plane of content.

The plane of expression of language is specified by stating the combination rules --rules of composition-- from which acceptable and meaningful expressions are produced. The same principle is used for studying relationships at the level of the plane of content. According to Klein (Klein 1988a) the modern formal --mathematically rigorous-- theory for representing the meaning of natural language expressions is due to Montague, who established a firm bridge between the concerns of linguists and logicians. An introduction to Montague's system is provided by Dowty (Dowty et al 1981).

Modern semantic studies are based on Frege's principle of compositionality (Frege 1952) which says that the meaning of a complex expression is a function of the meaning of its parts and the mode of grammatical combination. Sentences have as a reference their truth value. The mathematical formalisation of this principle and the definition of truth in relation to a model was presented by Tarski (Tarski 1938). According to this notion, expressions of a formal language refer to a mathematical world composed by an explicitly defined set of objects, which have some properties and stand in certain relations to each other. Such a world is called **a model**. A model is defined by stating not only the set of objects referred to by the language, but also an interpretation function which relates every name of the language to some object in the set, and every predicate of the language to a set of objects in the set. For this reason, semantic theories defined on these lines are known as model-theoretic semantics. The notion of truth of a formula is a notion of satisfaction. Truth is defined as the satisfaction of an expression by a model. If every object in the set satisfies the expression, such a formula is said to be true, otherwise it is said to be false. For instance, sentence (1)

(1) *every student is happy*

in the state of affairs introduced through GRAFLOG in Section 2.1 would be false in that model. The set of students is {Luis, John, Pete}, and the set of individuals who happens to be happy is {Luis}. Sentence (1) means that if there is an individual that is in the set of students he is also in the set of happy individuals; however, there are two individuals, namely John and Pete who are in one set but not in the other, and sentence (1) is false in relation to such a model. However, we could increase the knowledge by saying that *John and Pete are happy* and this new statement would change the model: the set of happy individuals would be

{Luis, John, Pete} and the new model would satisfy the expression. A model is a representation of the world referred to by language: a semantic representation.

The important matter for us is that this notion implies that truth is defined as a notion of correlation between the world and the language. The assumption here is that the individuals of the world can be abstracted as set-theoretical objects: that we can talk about the set of individuals constituting the world. This notion implies that the world and the language have to be known in an independent way, and for this reason this notion of truth is not circular. According to this theory, understanding a sentence means being able to establish an association between the linguistic expression --the message-- and the world. The job of a theory of semantics is to establish the correlation between world and language in an explicit manner. This view is also known as the correspondence theory of truth.

3.2.1. Montague Semantics and the rule-to-rule hypothesis.

Formal languages have a simple syntactic definition, and their expressions refer to well specified mathematical objects. For that reason the definition of truth in relation to a model is a satisfactory notion. However, natural language has neither the former nor the latter characteristic. So, is it possible to claim that the semantics of natural languages can be given in the lines of semantic theories for formal languages? According to Montague and his followers that is in fact the case: "I reject the contention that an important theoretical difference exists between formal and natural languages" (Dowty 1981, p.1). Fortunately, we do not have to take a position on this issue. The enterprise of making graphical and linguistic interfaces for human-computer interaction is a modelling task that can be supported, as was mentioned, by considering that current results in linguistic theory are encouraging.

Here, we come to the link between formal semantics and linguistics provided by Montague. He made the claim that the semantic analysis of natural language expressions can be produced in parallel to its syntactic analysis. According to this, the syntactic rules that determine the syntactic structure of a sentence should correspond one-to-one with the semantic rules determining the meaning of a sentence as a function of its parts. This systematic pairing of syntactic and semantic rules has been called the rule-to-rule hypothesis, and is adopted in a variety of grammatical frameworks and most versions of categorial grammar (Klein 1988a).

One of the reasons for the popularity of categorial grammar among modern computational linguists is the way that the so-called rule-to-rule hypothesis is developed and computed. In categorial grammar, an expression of type *A/B* denotes a function *f* whose domain is a set of expressions of type *B* and has as range a set of expressions of type **A**. If an expression of type *A/B* denotes *f* and an expression of type *B* denotes *b* then an expression of type *A* denotes *f(b)*.

The denotation, or meaning, of the symbols *Luis, Pete, John* and *programming* are the individuals in the world that happen to have these names. In categorial grammar, the meaning of verbs, such as *studies, plays,* etc, is modelled as a function. The verb *studies*, for instance, denotes the function *x studies y*, or in relational terms, *studies(x, y)*. The category of the verb determines not only the function but also the order in which it is computed. According to the grammatical rules, the word *studies* of category *(S\np)/np* denotes the function *studies(x, y)* and combines with the word *linguistics* of category *np* to produce the expression *studies linguistics* of category *S\np*. This expression denotes the function *studies(x, linguistics)*, and it can be combined in turn with the word *Luis* of category *np* to produce the sentence *Luis studies linguistics* and whose meaning is the relation *studies(luis, linguistics)*.

3.3. Computational Linguistics and the PATR-II Formalism.

Modern developments in linguistic theory have been strongly influenced by current computational technology. Computers are useful to linguists for performing complex grammatical analysis, for verifying the correctness of a grammar, for consistency checking and for preventing clerical errors. For that reason, computational tools for writing and testing grammatical formalisms have been developed. A particular tool in which several grammars can be tested is the PATR-II formalism (Shieber et al 1983). A Prolog implementation of the PATR tool called PIMPLE has also been developed (Calder et al 1986). In this Section an introduction to the PATR formalism is presented.

3.3.1. Directed Acyclic Graphs.

The declarative part of the syntactic rules and lexical entries of many grammar theories can be specified as sets of attribute-value pairs. Such a specification can be easily described in terms of directed acyclic graphs (*dag's*). Some examples of these graphs are presented in Figure 3.1.

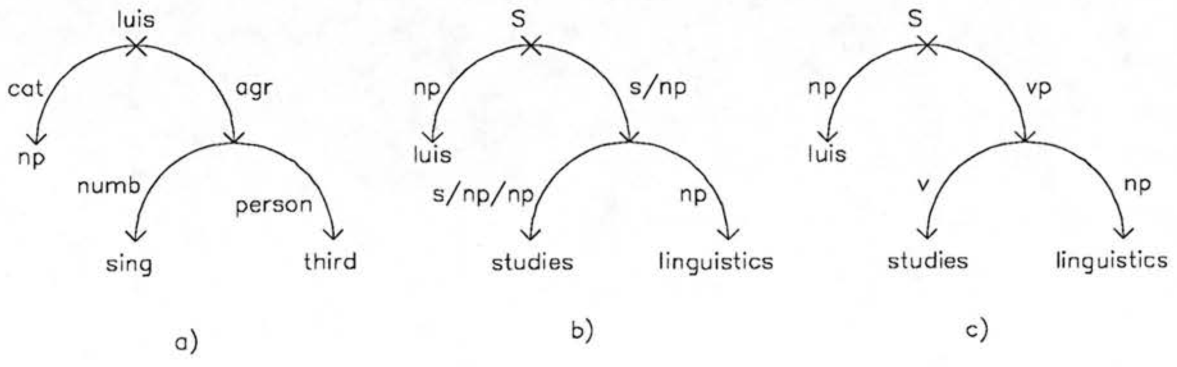

FIGURE 3.1

In Figure 3.1.a the lexical structure of the term *Luis* is shown. The node standing for Luis has two attributes: its grammatical category, a noun-phrase, and its agreement which in turn is a complex structure with two attributes: number and person, and these have as atomic values *sing* and *third*. The simple way to refer to an element in these structures is by describing the sequence of attributes --the path-- that must be followed in order to find that element. For instance, the value *sing* can be referred to as **luis:agr:numb = sing**. In Figure 3.1.b, the categorial derivation of the sentence *Luis studies linguistics* is shown. The same analysis for this sentence is produced by a simple context-free grammar with two rules, namely **S $\rightarrow$ np vp** and **vp $\rightarrow$ v np**, as shown in Figure 3.1.c. In fact, the close relationship between context-free and categorial grammars was noticed in 1958 by Bar-Hillel, and soon after they were proved to be equivalent in weak generative capacity by Gaifman, as reported by Pollard (Pollard 1985).

Note that in the three graphs, the label of an edge which ends in a node can be thought of as an abstraction of the whole structure leaving that node, in the same sense that pointers are abstractions of complex data-structures in traditional programming languages. Note as well that the incoming edge of a node is the 'attribute' which has as a value the structure which hangs on that node. The top-level of a syntactic structure is the sentence, and the top-level of a lexical entry is the word itself. The interesting thing here is that both levels of grammatical description can be captured within the same representational structure.

Within this formalism, the basic objects in the system are the lexical entries, and expressions of non-lexical grammatical categories are objects that are built along the syntactic analysis. In this sense, every term or expression in the grammar --either basic or complex-- is an 'object' of the system. For context-free grammars, the construction of the objects is guided by the rules when a sentence is parsed. In the case of categorial grammar, the objects are built in conjunction with the categorial combination. One of the advantages of using categorial grammars is that the categorial combinations that can be permitted in the construction of complex objects can stated by combinatorial schemes which are stored in the lexicon. The construction of complex objects is illustrated in Figure 3.2. The lexical items are represented in the bottom level of the Figure.

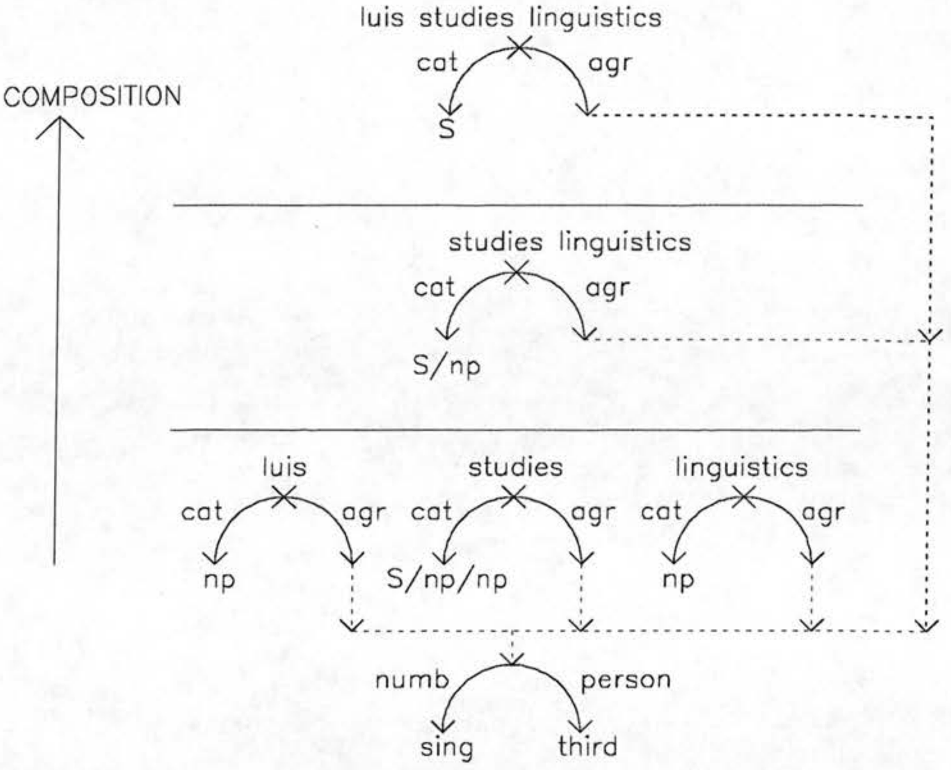

FIGURE 3.2

The agreement attribute in Figure 3.2 is common to the tree lexical terms and only one instance of such structure --which is referred to by all the others-- has to be represented. In PATR-II, modular structures that are part of several terms and are factorised out are called templates. Templates are referred to by means of a name. In the *dag's* presented below in the chapter the name for the third person singular agreement attribute is *3rdsing* or *3rds.*

As was mentioned, when two terms are combined in categorial grammar, one has to be taken as a functor and the other as its argument; in this example, the functor is the verb *studies* and its category is specified in terms of the combinations that are allowed for such a word within the system. Here, the verb takes an *np* by its right to produce a term of category *S/np* which in turns takes another *np* by its left side to produce a sentence. The direction of the 'slash' in the last term has been inverted. As can be noticed, the order in which the *dag's* have been drawn in Figure 3.2 does not necessarily reflect the order in which the lexical items have to combined. The order information has to be explicitly asserted. This point will be clarified latter.

The combinatorial process is illustrated by the two upper levels in Figure 3.2. In the second level, the expression *studies linguistics* is composed. This is a new object in the system, but it has its 'roots' in the lexicon. The complex object is a 'copy' of the functor term which is modified in two ways; first, the name of this structure is composed by taking together the names of the basic terms, and secondly, its category is made by taking the category of the functor term, and 'stripping' from it the category of the argument term. The representation of the final object is shown in the upper level. It is constructed as was shown for the second level object: its phonology is made by combining the names of its component parts, and its category is made by taking the category of the functor term --which in this case is *studies linguistics*-- and stripping from it the category of the argument term. The category of this last object is *S* and it cannot be further combined.

The category of agreement was propagated from the bottom to the upper levels. This can be thought of as the specification of a constraint on the combinatorial combination. From the bottom to middle level, the attribute of agreement is 'percolated up' or inherited upwards from the agreement of the verb --which is directly specified in the lexicon-- to the verb-phrase object. From the middle to the top objects, the attribute of agreement is a constraint on the possible combinations: the agreement of both objects involved most be the same in order that the categorial combination is permitted. This constraint reflects the grammatical fact that the subject of the sentence must agree with the verb.

In traditional grammatical analysis, the fact that the subject of a particular verb --such as *studies*-- must be a human entity is recorded as a selectional restriction related to the so-called sub-categorisation frame of that verb. This is a semantic fact, and it imposes a restriction on the kind of things that can perform the action denoted by the verb. For instance, the sentence *Linguistics studies Luis* must be ruled out by the grammar --at least in the actual world-- despite the fact that both *Luis* and *linguistics* are proper nouns. In categorial based formalims, such a restriction is made explicit in the lexicon, and the lexical entry for *studies* can be modified as shown in Figure 3.3. For the moment, let us take for granted that the sub-categorisation information is related to the right *np* --the second to be combined-- that is specified in the main category of the verb.

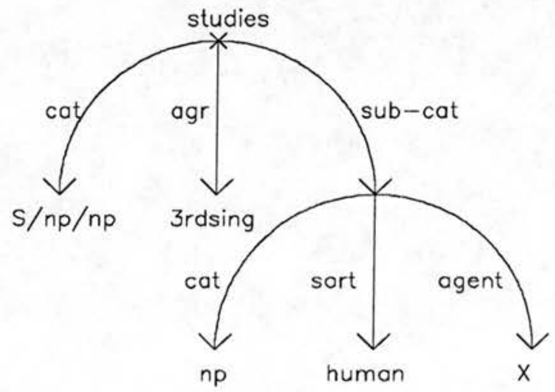

FIGURE 3.3

The notion of selectional restriction makes very thin the boundary between syntactic and semantic domains in a grammar. Furthermore, for our enterprise an explicit representation of the semantics of the lexical items is required, and for such a purpose another feature structure can be used. As has been mentioned, the semantics of a verb is a function with two arguments, and the lexical entry for *studies* can be augmented for capturing this information. In Figure 3.4 the new entry for *studies* is shown.

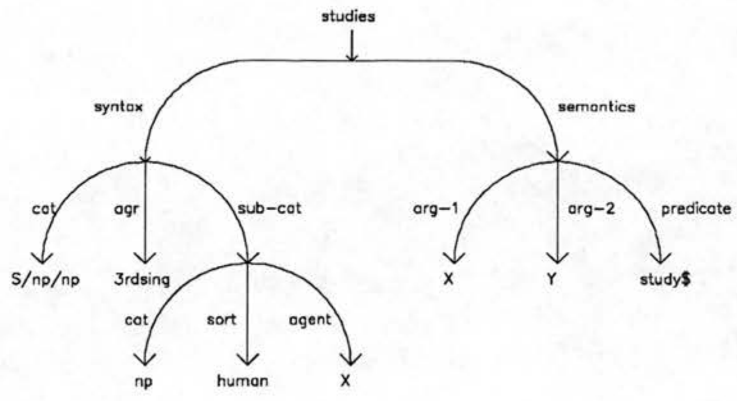

FIGURE 3.4

As can be seen, the semantics can be represented as an additional attribute in the feature structure. The label *semantics* has as a value the *dag* consisting of the predicate *study$* and two variables *X* and *Y* which have as labels *arg_1* and *arg_2* respectively. The semantics of the verb is then the relational function *study$(X, Y)*. Now, note that the agent term in the sub-categorisation information is the same as the first argument of the semantics: these two variables denote the same individual. We can capture this fact by modifying slightly the current *dag*, as shown in Figure 3.5.

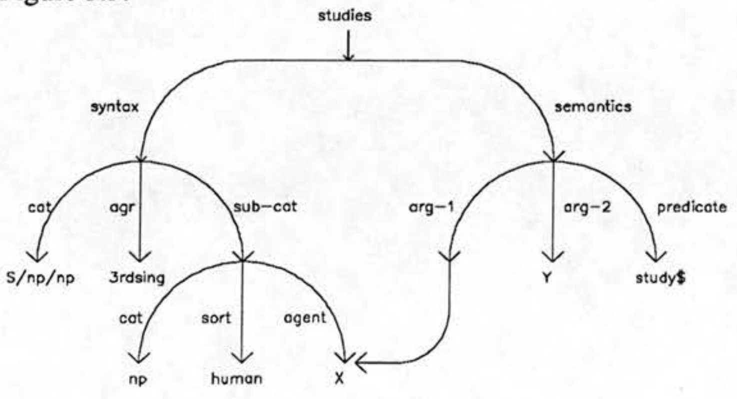

FIGURE 3.5

The variable *X* can be referred to by two alternative paths in the *dag*. These are **studies:syntax:sub-cat:agent = X** and **studies:semantics:arg_1 = X.** Note that the structures represented from Figures 3.1 to 3.4 could be thought of as simple hierarchies. However, the need for sharing information between the syntax and semantics makes a complex feature structure --*dag's*-- a more suitable representational structure. Within the PATR-II framework, a structural element that can be reached by more than one path is referred to as **reentrant**. In the definition of a *dag* completely cyclic graphs are not accepted. In general, a complex feature structure is recursively defined as,

(1) a variable, in which case the *dag* will be called 'uninstantiated' or 'unspecified'.

(2) a constant, such as *sing* or *third.*

(3) a set of attribute-value pairs, each of whose values is itself a complex feature structure.

3.3.2. Unification.

The construction of complex objects is performed by combining basic structures. For this process a basic operation is defined. Such an operation is **unification** and it can be thought of as the union set-operation between graphs. Two *dag's* which are identical unify in the same *dag*. Two *dag's* which are mutually exclusive (in the sense that every attribute-value pair appearing in one *dag* does not appear in the other) unify in a *dag* which has all the attribute-value pairs of each of the unified graphs. *Dag's* which partially overlap unify in a *dag* which contains the attribute-value pairs that appear in one or the other unified *dag's*. If the value of the same attribute of two different structures is a different constant, these structures cannot be unified, and unification between them fails.

Unification can be thought of in more abstract terms. In general, two descriptions x and y can be unified if there is an object z that fits both descriptions. Unification is widely used in many computer disciplines and a general overview of the subject is given by Knight (Knight 1989). As an example, consider the terms $f(x, y)$ and $f(g(y, a), h(a))$. They are said to be unifiable since replacing x by $g(h(a), a)$ and y by $h(a)$ will yield both terms as $f(g(h(a), a), h(a))$. In general two terms s and t are unifiable if there is a substitution σ such that $\sigma(s) = \sigma(t)$. In such cases σ is called a *unifier* of s and t. A substitution can be thought of as a list of variable bindings; the unifier for the example is $\sigma = \{x := g(h(a), a), y := h(a)\}$. The most general unifer (*MGU*) is defined as the simplest such substitution.

3.3.3. Definition of a Simple Grammar.

In the PATR-II formalism, the definition of a grammar consists, on the one hand, of the specification of the set of rules from which well formed expressions are produced, and on the other, of the specification of the constraints that have to be satisfied in for the application of the rules. The construction of complex objects by this dual specification has been partially illustrated in Figure 3.2. The expression *studies linguistics*, for instance, was produced by the application of a combination rule, and the constraint that the category of agreement of the verb should be percolated up to the verb phrase was also specified. However, such an example is too specific, and it is not clear when some or another rule has to be applied, and what are the constraints that have to be satisfied in order that the application of such a rule is permitted. One way to generalise this process is by making explicit the rules guiding the interpreter making the composition, and by specifying in the *dag's* themselves not only the particular combinatorial schemes permitted for a term, but also the constraints that must be satisfied by such operations. Next, an example for illustrating such a procedure is developed. The following scheme is presented only for didactical purposes, and no particular grammatical formalism or notation is implied. However, the example has been designed with the aim of introducing some of the more relevant theoretical and computational notions that are defined in *UCG*.

First, the representations of the lexical entries for the words *luis, studies* and *linguistics* have been augmented. These words, as well as all the words known by the system, are stored in the lexicon. This is indicated by the attribute *lexical item id.* The lexical entry for *studies* has, besides the syntax and semantics attributes, three new main paths: *phonology*, *order* and *operation*, as shown in Figure 3.6.

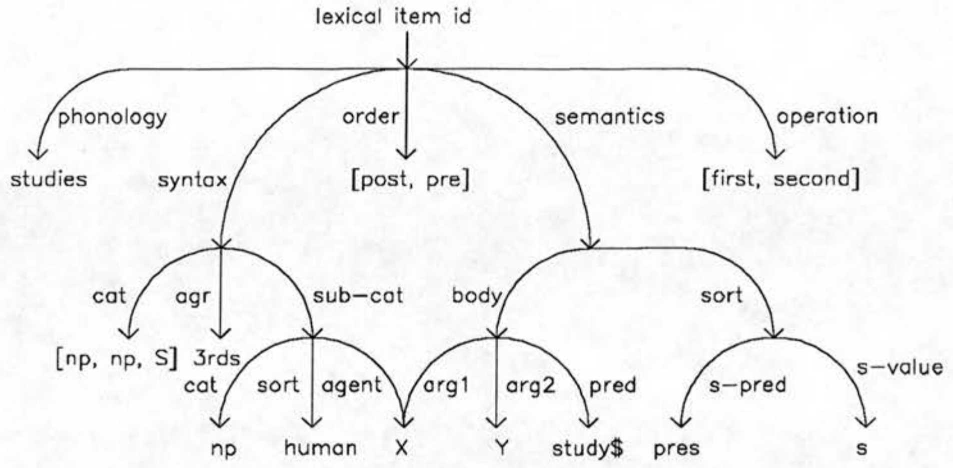

FIGURE 3.6

The phonology attribute has as value the string with which this word is realised. The other two attributes have a list as a value. The value of 'order list' is *[post, pre]* and the value of the 'operation list' is *[first, second]*. Note that the value of the path **syntax:cat** is also a list: *[np, np, S]*. Let us say that the header of each of these three lists is known as **the active part** of the object. The intuition behind this definition is that the 'active part' of the object provides the information required by the functor term for the current combinatorial process. It should be clear that the active part of the path **syntax:cat** has to match the same path of an argument term in order for the combination to be permitted. The order information specifies the position of the corresponding argument term in relation to its functor. In the operation list the set of additional constraints that have to be satisfied to allow the current combination are stored. The terms *first* and *second* identify the lists of unifications that have to succeed in order that the corresponding combinations of the term *studies* are permitted. The term *first* is the list

first = [functor:semantics:body:arg_2 = argument:semantics:pred].

and *second* is the list

second = [functor:semantics:body:arg_1 = argument:semantics:pred,

functor:syntax:sub_cat:sort = argument:semantics:sort,
functor:syntax:agr = argument:syntax:agr].

Note that every element of these lists has the form $\alpha = \beta$, where α is a *dag* in the functor term, and β is a *dag* in the argument term. Note as well that these specifications might have similar form for many linguistic terms, and they can be defined as templates within the system. However, such an economical factor is not considered in the present example. The 'semantics' attribute of *studies* has also been augmented. A new attribute indicating the kind of thing denoted by this verb has been asserted: **semantics:sort**. In the same way that names --like *Luis*-- denote a human individual, the verb has as a sort a temporal state *s* which is also a present --pres-- state. The semantics for *studies* is then that there is a present state *pres(s)* which is the state of *X* studying *Y*.

The *dag's* for *Luis* and *linguistics* have been augmented with the phonology and semantics arguments as shown in Figure 3.7.

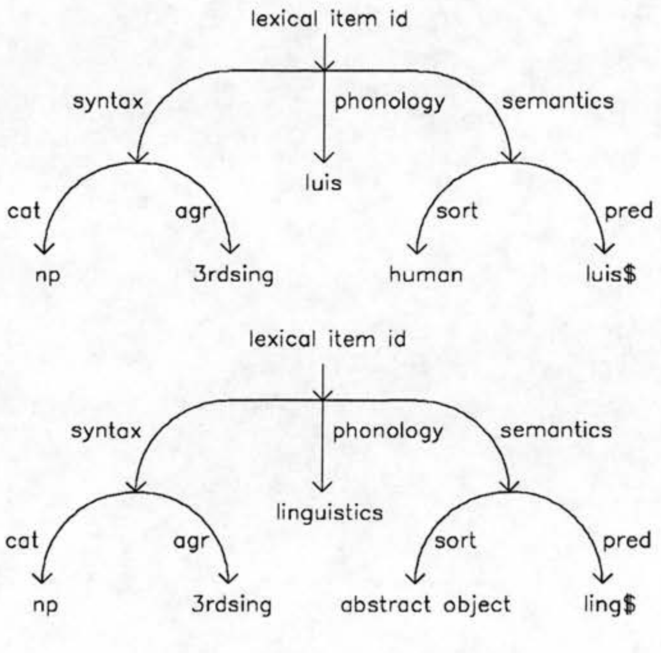

FIGURE 3.7

Now, we can define a very simple interpreter for combining these terms for building up a complex objects. The first step in the production of an object is its instantiation. For this process, a functor term has to be found in the lexicon, and a 'copy' of this object labelled with the attribute *functor* is made. This object would be, for instance, a copy of the *dag* in Figure 3.6, but with the term *lexical item id* replaced by *functor*. When the functor term has been selected, the interpreter has to find an argument term --originally in the lexicon-- and replace its identifier, *lexical item id*, by the term *argument.*

Now, let us define the function *head* by which the interpreter is able to know the active part of a list in the functor object. This function has as an argument a path specification. The value of such a path must be a list, and the function *head* returns the first element of such a list. Let us define the procedure *pop* as well. This procedure is like *head*, but it removes the first element of the list that is referred to. We also define the procedure *unify* which takes a list of unification specifications, and applies a unification procedure to each element in such a list. We define the function combination operation of the interpreter as,

```
combine(functor, argument):
        %Category check rule
        head(functor:syntax:cat) = argument:syntax:cat.

        %Phonology order rule
        IF head(functor:order) = post THEN
                functor:phonology := functor:phonology + argument:phonology.
        IF head(functor:order) = pre THEN
                functor:phonology := argument:phonology + functor:phonology.

        %Combinatorial restriction rule
        unify(head(functor:operation)).

        %stripping rule:
        pop(functor:syntax:cat).
        pop(functor:phonology).
        pop(functor:operation).
```

where the symbol '=' denotes that the value of the paths α and β is the same. The symbol ':=' denotes a value assignment on a *dag* path, and the symbol '+' denotes the concatenation of two strings.

Through this procedure, the phrase *studies linguistics* is composed by the application of the procedure *combine* to the *dag* standing for the current functor object, and taking the *dag* standing for *linguistics* as the argument object. The resulting object is shown in Figure 3.8. Note that the active information has been 'stripped' from the functor object. Now, this new object is looking for an *np* whose phonology will preceed the current phonology, and which has to be in agreement with the current object.

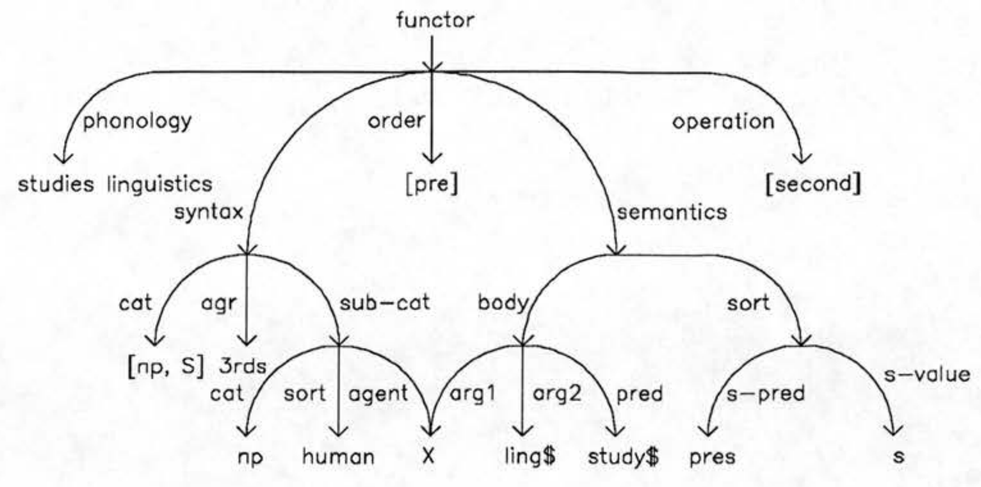

FIGURE 3.8

This new object is combined in turn with the *dag* standing for *Luis*. The output of the second combination is the *dag* shown in Figure 3.9

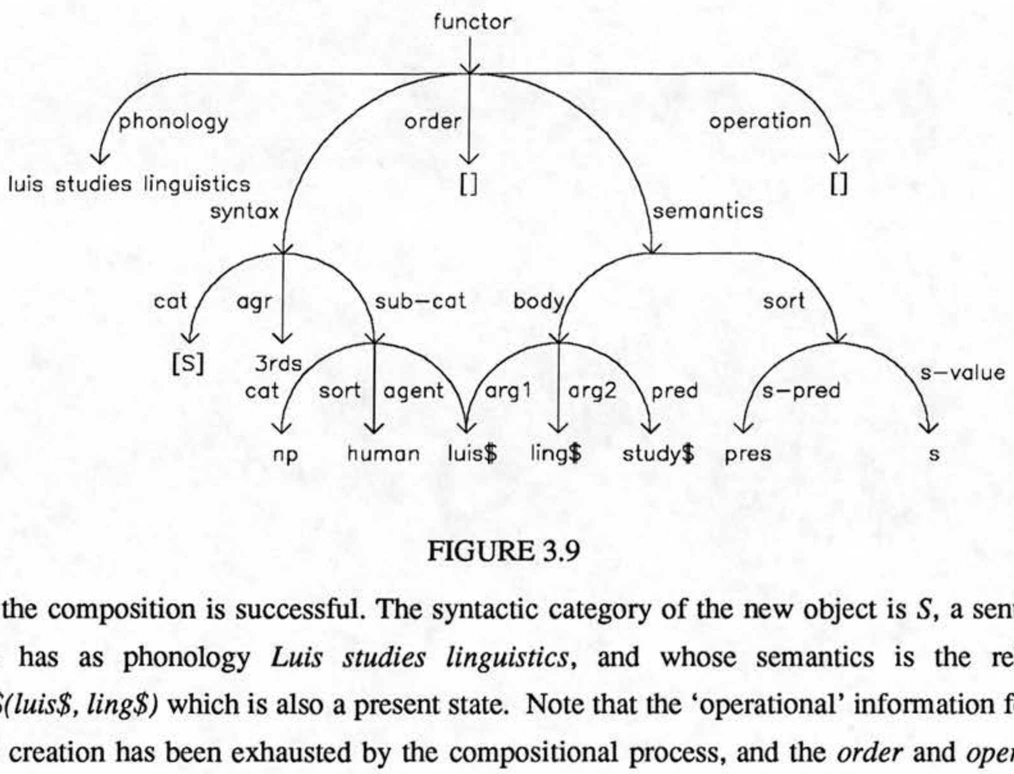

FIGURE 3.9

Now, the composition is successful. The syntactic category of the new object is *S*, a sentence, which has as phonology *Luis studies linguistics*, and whose semantics is the relation *study$(luis$, ling$)* which is also a present state. Note that the 'operational' information for the object creation has been exhausted by the compositional process, and the *order* and *operation* paths have an empty list as a value.

Here, it is worth emphasising some of the main points of this scheme. Firstly, it is lexically oriented. The descriptions of the lexical entries specify not only the grammatical information related to an entry, but also the information that is used in its own interpretation process. This operational information contains the specification of the permitted combinations, the lexical order in realisation and the constraints and value assignments that have to be made in each combinatorial step. A direct consequence of this approach is that the interpreter itself is very simple and general, and it can also be specified and implemented in a highly declarative fashion. A second important consideration is that the information acquisition process is monotonic. This means that the grammatical information gained along the interpretation process is never lost. This contrasts with the fact that the operational information is lost along the same process.

The fact that every object within the system --lexical, at an intermediate level of composition or a complete sentence-- is represented by a well defined *dag* has at least two important repercussions that have not yet been mentioned. The first is that sentences can be linked together to pass discourse information among them. This information can be used to compute the references of anaphoric pronouns. The second, is that 'unary' operations can be applied to a *dag* to change its characteristics in the compositional process. An introduction to these two problems is given in the rest of the Chapter.

3.4. Representation of Discourse information.

Consider the following piece of discourse,

Luis studies linguistics. He likes it.

The grammatical analysis of the first sentence can be performed in compositional terms as was shown above. We would like to do the same for the second, but there is a problem. In order to determine its semantic representation, the referents for the pronouns *he* and *it* must be found. In this simple discourse, it is clear that these referents should be *Luis*, and *linguistics*. However, a theory for making such an identification must be made explicit. In such a context, the words *he* and *it* are *anaphoric pronouns*,[1] and they act as a kind of 'variable' in the natural language expressions. On the lines of the grammatical theory and the computational techniques that have been sketched, the semantics of *He likes it* would be something like the function *likes(x, y)* where x and y are variables denoting the individuals referred to by the corresponding pronouns. However, in order to collect this information in the knowledge base we would like to make the substitutions x := *luis* and y := *linguistics* and then to assert the relation *likes(luis, linguistics)* in it. Within theoretical and computational linguistics the study of these problems is the matter of discourse theory.

Current theories of discourse representation are highly influenced by Kamp's theory of truth and semantic representation (Kamp 1981). The aims of this theory are threefold: to give a general account of conditionals, of the meaning of indefinite descriptions, and of pronominal anaphora. Kamp's theory has been further developed in conjunction with unification based grammar

[1] If the pronouns are supported by physical pointing gestures, these pronouns would be deictic, but for the moment we exclude this complication. The deictic use of pronouns will be analysed in Chapter 4.

formalisms (Zeevat 1986b). Following Johnson and Klein and others (Johnson et al 1988) the meaning of a linguistic expression α is a relation between the context that precedes and follows α. The meaning of α has --on this view-- the general form,

Preceding-context |α| following-context.

In a simple model, the discourse context is considered just a set of individual names or reference markers. In Figure 3.10 an illustration of such a context is shown.

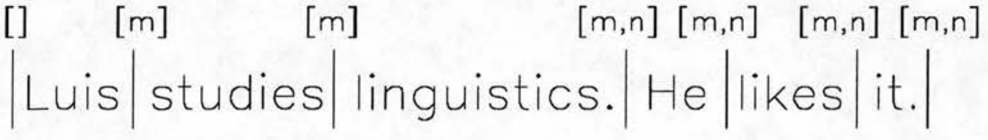

FIGURE 3.10

The bars in Figure 3.10 represent moments of time. The context preceeding the term *Luis* is the empty set, and the context following it is the set with the marker *m*, which stands for a masculine object, and it is also a reference to the term *Luis*. The context preceding *linguistics* is the set *{m}*. This term adds the discourse marker *n*, and its right-context is the set *{m, n}* where *n* stands for a neuter object. In this model, the discourse context is determined in terms of a set of equations relating the context which preceeds a lexical item with the context that follows it; for instance, the context for *linguistics* would be,

{} | linguistics | {n}

and the relation between lexical and discourse context can be established in terms of the union set operation between contexts as

$C \mid \textit{linguistics} \mid C \cup \{n\}$

The solution of an anaphoric reference can be found by solving an equation relating the pronoun sort with its left-context. For instance, the pronoun *he* refers to a masculine-human subject and it can be matched with the antecedent *Luis* by means of the discourse marker, but not with *linguistics*. In the same way, *linguistics* can only be matched with the pronoun *it* in this example. The meaning of the anaphoric *he* is the relation,

$C \mid \textit{he} \mid C$ **iff** $m \in C$

Of course, for more complex pieces of discourse, there might be several solutions for these equations, and the discourse is ambiguous.

This simple model of discourse can be integrated with the syntactic and semantic information by means of a *dag* representation. A mechanism for passing information between constituents in unification grammar under the name of **gap threading** was introduced by Pereira, and further developed by Karttunen (Karttunen 1986). This mechanism is illustrated in Figure 3.11

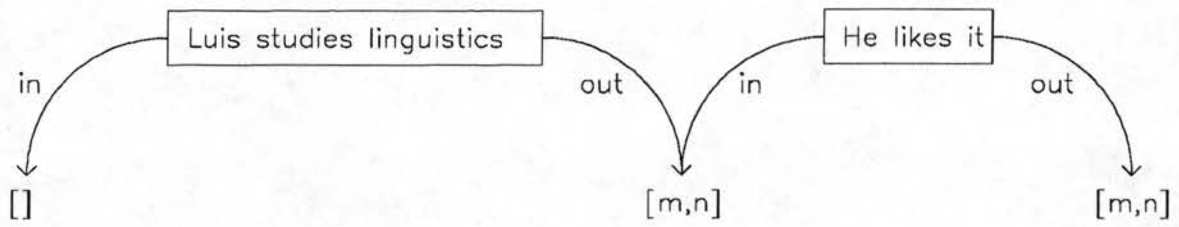

FIGURE 3.11

Here, an additional sub-*dag* has to be defined in every object. In this example, the reference for the sentence *Luis studies linguistics* is s_1 and the reference for *He likes it* is s_2. The new *dag's* for these objects are:

s_1:thread:in = [].
s_1:thread:out = [m, n].
s_2:thread:in = [m, n].
s_2:thread:out = [m, n].

Intuitively, this technique requires that a sequence of objects can be threaded if for any two adjacent objects s_i and s_{i+1},

s_i:thread:out = s_{i+1}:thread:in.

In this simple model of discourse context, the procedure for solving anaphoric references has to look for the possible antecedents of a pronoun in the **thread:in** list for that object. The definition of such a procedure is then fairly straightforward. This model, however, is not fully satisfactory.

The naive model of discourse is not able to account for the anaphoric properties of universally quantified *np's*. The data which shows this problem is fairly well known (Johnson et al 1988). In general, a universal term can normally enter into an anaphoric relation with pronouns that are in its scope. Consider expressions (2) to (5) in which the scope of universal terms is in bold.

(2) *A student studies linguistics. He is happy.*

(3) *Every student* **studies linguistics.** *He is happy.*

(4) *If a student* **is in a subject, he studies it.**

(5) *Every student* **who is in a subject studies it.**

In (2) there is no universally quantified term and the pronoun *he* has *a student* as its antecedent. In expression (3), the anaphoric binding is not allowed because the pronoun *he* is outside the scope of the quantified term *every student.* Expression (4) illustrates a relation between a universal term --*if a student*-- an indefinite term --*a subject*-- and the pronouns *he* and *it.* Here, the universal has wider scope than the indefinite, and then the anaphoric link between *a subject* and *it* is permitted. But this link would not be permitted if the scope of the indefinite were larger than the scope of the universal. Expression (5) is similar to (4) and is included in order to emphasise the the universal force of an indefinite term preceded by the conditional *if.*

Sentences like (4) and (5) are known as 'donkey sentences' and they have been widely studied. Kamp's discourse theory gives a natural account of the complex relations that they exhibit.

Here, there is no intention to discuss the complex issues involved in the analysis of sentences such as (4) and (5); however, this example can be used to motivate some of the mechanisms involved in a more satisfactory model of discourse. In the naive model, the context is a simple set, and in the resolution process, the antecedent of a pronoun can be any marker in the set; however, we can partition such a set in terms of a notion of subordination. Intuitively, the definition of a subordinated discourse context is triggered somehow by the expressions that introduce --or restrict-- quantification. These expressions are, in general, noun-phrases, and they act as 'pivots' that open the scope of the corresponding quantifier. With this new device, a reference marker can be the antecedent of a pronoun if it is 'accessible' by such a pronoun. Although the mechanisms that govern these relations are fairly complex, a fairly general account of them has been given (Johnson et al 1988). For our purpose, however, an intuitive understanding of such relations will be enough. Consider the contrast between Figure 12.a and Figure 12.b.

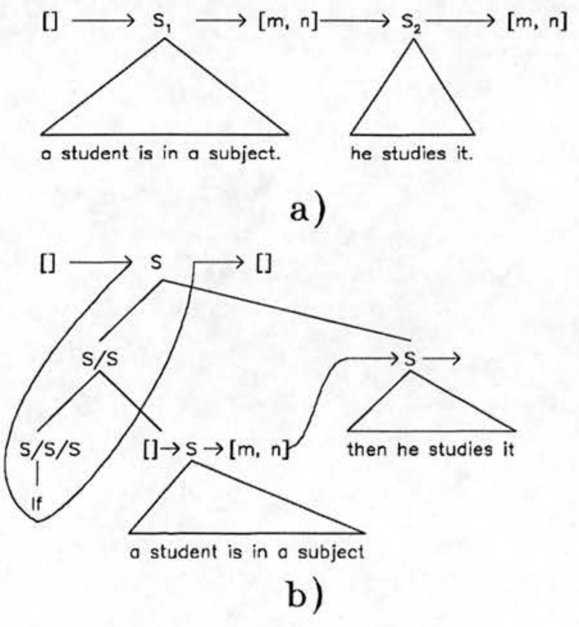

FIGURE 3.12

In the sentence of Figure 12.a there is no universal term involved, and the discourse context has no subordinated structures. The sentence *He studies it* will pass to whatever comes to its right the current discourse context *{m, n}*. However, if the first sentence *a student is in a subject* occurs in the context set by the conditional *if*, the term *a student* has universal force and nothing within its scope will be inherited to the piece of discourse which will follow the whole conditional expression. The main discourse context is passed from its input to its output list as soon as the universal is detected, and a subordinated discourse context will be open as shown in Figure 12.b. The references for the pronouns in the consequent clause will have access only to the discourse markers introduced by the *np's* of the antecedent clause.

Before closing this Section, a word in relation to the computational considerations is worthwhile. In this model, and according to Johnson and Klein, the bottom-up compositional process given for the grammatical analysis of these sentences is naturally integrated with the top-down information flow that is associated with anaphoric information.

3.5. Unary Rules, Gaps and Word Order Inversion.

In this Section some other very common linguistic phenomena such as plural terms, word order inversion and constituent movement are presented, and the way they are handled in unification based formalisms is illustrated.

3.5.1. Unary Rules.

In general, if d_i is a grammatical object --represented by a *dag*-- it can be transformed by a unary operator θ into a new object d_j if d_i satisfies a set of conditions required by θ --this is if d_i is of certain specific form.

This facility is useful in many circumstances. For instance, for the definition of the different forms of verbs in terms of a single base entry. Suppose that the base item --stored in the lexicon-- for the verb *studies* is *study.* We can produce the forms *studies, studying, studied, to study* and *is studied by* by the application of rules θ_i, θ_j, θ_k, θ_l, θ_m respectively. In each case, the output *dag* would contain not only phonological information, but also the syntactic and semantic attributes needed for combining the new objects in the set of contexts in which they can occur. Furthermore, these rules would apply not only to this particular verb, but also to all verbs of similar form.

Here, the use of unary rules will be illustrated with an example taken for the linguistic fragment presented in Chapter 2. Next, how a plural common noun can be produced from the corresponding singular form is shown. Consider sentences (6) to (9).

(6) *This is a student.*

(7) *These are students.*

(8) *These are clever students.*

(9) *Clever students study linguistics.*

Sentence (6) is in singular form and its structure can be produced according to the scheme presented above in Section 3.3. The term *a student* is a constituent of category *np* and the form of this sentence is *np (S\np/np) np*. Expression (7) is the plural form of (6). This sentence could be produced by a derivation similar to the one for (6) but with a plural agreement between the subject and the verb-phrase. The plural object *students* could be stored directly in the lexicon with all its phonological, syntactic and semantic characteristics. However, under such a scheme the modification of *students* by an adjective like *clever* would not be produced by the grammatical system. The problem is that *clever students* is a plural *np*, as well as *nice clever students* or *very nice clever students* are. The fact that terms of category *noun* can be modified in the process to be converted to an *np* suggests that the whole constituent must be combined first and then it can be transformed into a plural term by a single transformation. The purpose of this Section is to illustrate such a unary rule.

The definitions for lexical entries for the words *clever* and *student* are shown in Figure 3.13.

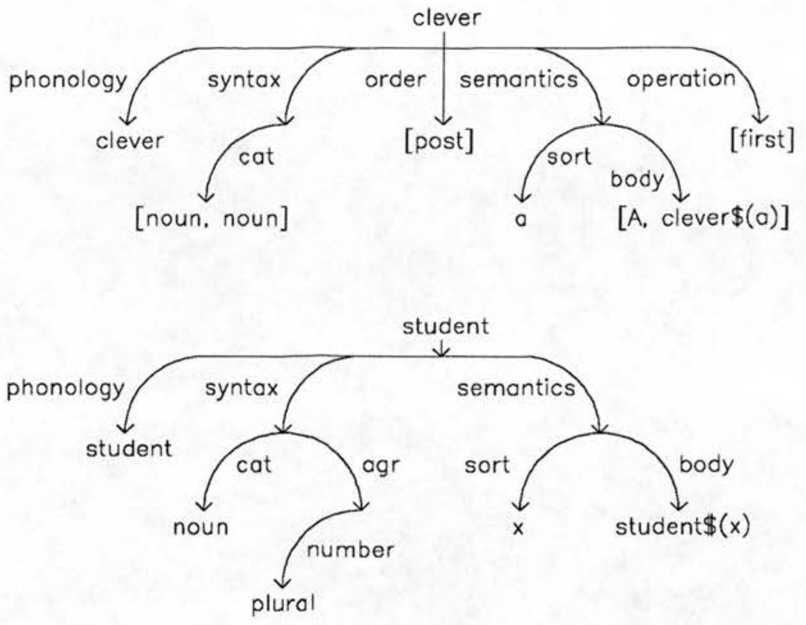

FIGURE 3.13

The adjective *clever* is a term of category *noun/noun*, and it can combine with a term of category *noun* by its right side --the order list has *post* as a value-- to yield a modified noun. The semantics of this term has been represented by a *dag* in which the sort attribute has as a value the variable *a* which is yet uninstantiated. The body attribute of the semantics has as a value the complex structure *[A, clever$(a)]*. We can think of this structure as the conjunction of terms within the list, in which *A* is a variable standing for a formula. The value of *A* will be acquired in the combination process. The index *a* is an unsorted variable. The term *first* in the operation list contains the list of unifications that have to be applied to the *dag* when it is combined with some argument. The value of *first* is the list:

[functor:semantics:sort = argument:semantics:sort,
head(functor:semantics:body) := argument:semantics:body].

Note that we assign a value to the head of the list representing the semantics of the functor term.

This is a new operation that can be easily defined with the interpreter presented in Section 3.3.

The definition of common noun *student* is straightforward. Note, however, that the attribute **student:syntax:agr:number = plural** has been defined. In this system, an independent entry for the singular form of each noun has to be defined. An alternative strategy would be to produce both forms from a base lexical entry by means of a unuary rule.

Now, we can combine the terms *clever* and *student* by means of the function *combine* as shown in Section 3.3. The result of *combine(clever, student)* is the *dag* shown in Figure 3.14. The category of this object is *noun*, with phonology *clever student* and semantics *[x][student$(x), clever$(x)]*. The variables of the functor term have been instantiated by values taken from the argument term. In particular, note that the sort value *a* has been replaced by the value *x*. Here, both are variables, but as will be shown in Chapter 4, they denote entities with different ontological properties. While *a* stands for an individual of whom we know nothing, *x* stands for an individual that, although its specific identity has not been defined, it is known to be a human being.

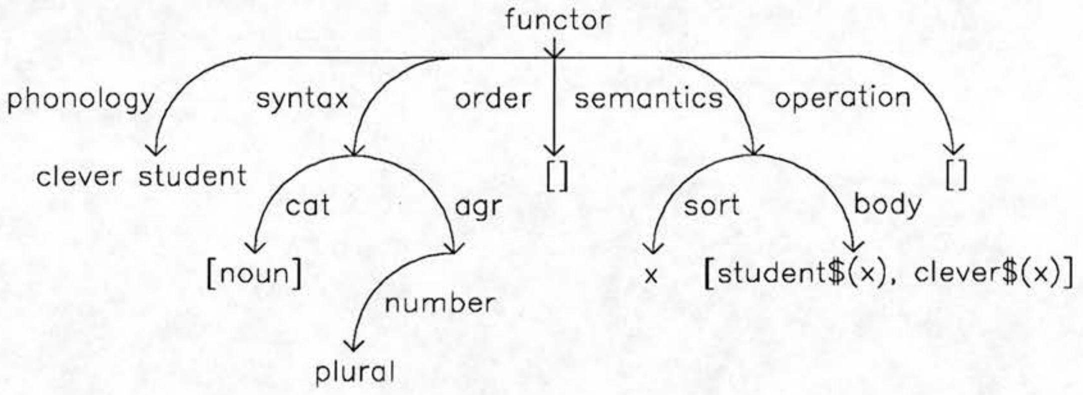

FIGURE 3.14

The object represented by the *dag* in Figure 3.14 cannot be taken as a functor for an additional combination. Its operational information has been exhausted. But, it could be taken as the argument of another adjective, like *nice* whose lexical definition is like *clever*, and the noun *nice clever student* could be produced. However, when the whole constituent is produced, it has to be combined with a verb to produce a term of category *S\np* and for that the term of category *noun* --simple or modified-- has to be converted into a plural *np*. This is achieved by the unary rule that we are concerned with in this Section.

The rule for forming plural *np* from terms of category *noun* is defined as,

IF

head(term:syntax:cat) = noun *AND* **term:syntax:agr:number = plural**

THEN

term:phonology := term:phonology + *s*.

term:syntax:cat := *np*

term:semantics:sort := *plural(X)*

where *term* is the object to be modified. The result of applying this operation to our current *dag* would yield the *dag* shown in Figure 3.15.

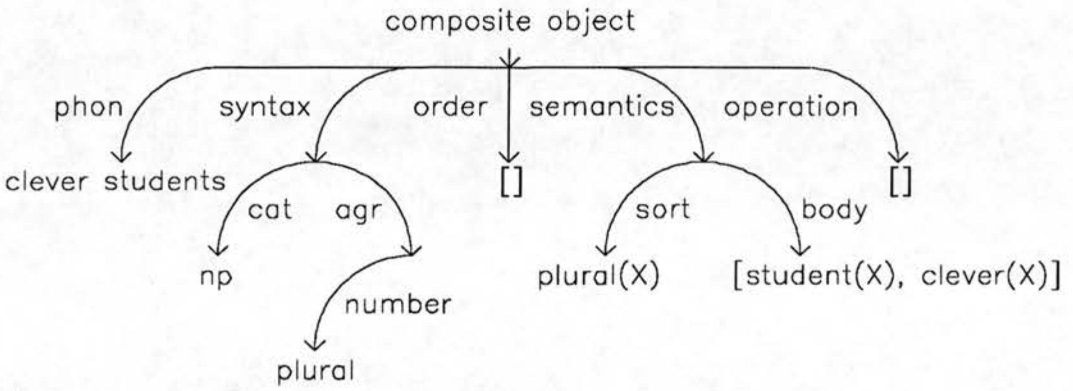

FIGURE 3.15

Note that the attribute *agr* of this object would not be needed for the production of the sentence *These are clever students* because the object term does not have to agree with the verb; however, if the term *clever students* is taken as the subject of a sentence, as in *Clever students study linguistics*, the agreement information would be required. Similarly, the expression *Clever students studies linguistics* is ruled out by this grammar.

3.5.2. Subject-verb Inversion in Questions and Unbounded Dependencies.

Now, consider the sentences,

(10) *This is Luis.*

(11) *Is this Luis?*

(12) *Who is this?*

Sentences (10), (11) and (12) are somehow related. In fact, if an individual is introduced by means of sentence (10), sentence (11) is a yes/no question by which the identity of such an individual is confirmed, and sentence (12) is a *wh*-question by which such an identity is required. A very general fact of English is that a declarative sentence is related to its corresponding question by inverting the positions of the subject and verb, as shown by the contrast between (10) and (11). Note that in both expressions the subject position is occupied by *this* and the object position by *Luis*. Now, compare expressions (11) and (12); these are similar questions, but realised in slightly different manner. However, in (12) the word *this* seems to be taking the object position, and the word *who* seems to be taking the subject position. In the corresponding declarative expression, the subject is the agent of the action denoted by the verb, and the object is the patient of such an action, and these roles should remain the same in both declarative and interrogative forms. Expression (12) can be better understood as the 'surface realisation' of the expression *is this who?* --contrasting with *is this Luis?*-- in which the particle *who* is an *np* that has been shifted to the beginning of the sentence. In general, for dealing with questions we have to consider the order inversion between the subject and the verb. Additionally, for *wh*-questions, the shift of the *wh*-term has to be taken into account. The inversion between subject and verb in questions can be modelled by means of one unary rule, such as the one presented for modelling plural terms. However, the second phenomena has an additional degree of complexity because the *Wh*-term can occur in many different positions.

The following procedure is a simplified version of Pereira's treatment of unbounded dependencies, as reported by Calder (Calder et al 1986). For dealing with this problem the notion of threading presented in the previous Section is useful. The *dag's* structures of every lexical item can be used. In Section 3.4, the value of the paths **thread:in** and **thread:out** was defined as the discourse context. We can modify slightly such a definition, and declare that the value of these paths is a list of two pointers: the first has as a value the discourse context as shown before, and the second points to a list, called *gap list*, in which constituents that are not found in their usual position --unbounded-- are provisionally placed. When they are required by the interpreter, they are taken back from the *gap list* and combined according to the grammatical rules of the system.

When the word *who* in *Who is this?* is processed, it is placed in its own **thread:out** list --which has the same value as the **thread:in** of the next term in the string. The verb *is* has to be modified by applying the unary rule for subject verb inversion in questions. Then, the term *who* is taken back from the input list, and it is combined with *is* yielding an expression with underlying form *is who*. This last expression is in turn combined with the word *this* to produce an object with the underlying form *this is who* which has the desired semantics.

Here, this introduction to theoretical and computational linguistics is concluded. It is worth emphasising that this very simple grammatical system has been presented with the aim to introduce some linguistic notions that are greatly generalised in *UCG*. It should not be taken as a formal specification but rather as an intuitive motivation of the subject, as many details have not been worked out. However, we can summarise the main points of the scheme. The most important feature of the system is that the main source of information is specified in the lexicon. Once the lexical schemes, the combinatorial rules and the unary rules have been defined, the definition of the grammar is completed by specifying the lexical items. The next most important feature is that every grammatical constituent is an object, and it can be represented by means of a feature structure in the system; then lexical and composite objects can be combined in building up the sentences and the discourse representation. It is also important to mention that the compositional process takes place in a bottom-up process, and it is naturally integrated with the top-down left-to-right discourse information flow. Also, the construction of the semantic representation is a monotonic process, given that the information gained in the compositional process is never lost. Now, we are ready for the formal description of *UCG*. Such a description and the derivation process for the linguistic fragment of Chapter 2, including a discussion of the deictic use of demonstratives and some suggestions for dealing with deictic pronouns, is the subject of Chapter 4.

Chapter 4

Deixis in Unification Categorial Grammar

In this Chapter the grammatical analysis of the linguistic fragment introduced in Chapter 2 is presented. Section 4.1 is a standard presentation of Unification Categorial Grammar. In Section 4.2 the lexical entries for our linguistic fragment are shown, and the derivation for some of the most relevant expressions of the dialogue of Chapter 2 are illustrated. Finally, in Section 4.3 the model for the deictic use of pronouns and demonstratives within the framework provided by *UCG* is presented. This model has been developed for the particular aims of this research.

4.1. Unification Categorial Grammar.

Unification Categorial Grammar (*UCG*) was originally introduced in 1986 by Zeevat, Klein and Calder (Zeevat et al 1986a). *UCG* is a version of categorial grammar that on the theoretical side enforces principle of composition, and relates phonology, syntax and semantics as tightly as possible. In addition, *UCG* is defined to handle intra and inter-sentential anaphora. For that purpose, several insights of Kamp's theory of truth and semantic representation (*DRT*) are incorporated (Kamp 1981).

On the practical side, it has been motivated by the desire to develop a theory which could be parsed efficiently with current computational technology. For the implementation of *UCG* several ideas of the PATR-II unification-based formalism (Shieber et al 1983) have been used.

The basic object in *UCG* is called a *sign*, and it has the following four major attributes,

(1) **i.** The phonology of the object (its orthography in this context).
ii. The syntactic category of the object
iii. The semantics of the object.
iv. The order information for the object combination.

In *UCG* there are three basic or primitive categories: nouns (*noun*), noun phrases (*np*) and sentences (*Sent*). Categories are defined as follows:

(2) **i.** Any primitive category is a category.

ii. If *A* is a category, and *B* is a sign, then *A/B* is a category.

The sign *B* in a category of the form *A/B* is called *the active part* of the sign in which *A/B* occurs.

Here, a notational convention is introduced: if *W, C, S, O* are variables standing for the phonology, grammatical category, semantics and order of the sign *E* respectively, then *E* can be specified either in a sequence notation as:

W:C:S:O

or in a column notation as

W
C
S
O

For instance, the sign for *studies* is

(3) **studies**
(Sent/w_1:np:x:pre)/w_2:np:y:post
[s][PRES(s), [s]STUDY(s, x, y)]
O

The category of the sign in (3) is of the form *A/B*, where *A* is *(Sent/w_1:np:x:pre)* and *B* is *w_2:np:y:post.* The category of *B* is a primitive object, a sign specified in a sequence notation, and its attributes are phonology w_2, category *np*, semantics *y* and order *post*. It is also the active part of the sign for *studies*. The parenthesis indicate that the association for the combinatorial functor "/" is to the left. The category *A*, on the other hand, is a complex term of the form *(Sent/w_1:np:x:pre)*. This category is in turn of the form A_i/B_j, where A_i is a sentence --of category *Sent*-- and B_j is a basic object of category *np*.

The semantics of the sign is an expression of a representational language called *InL* (Zeevat 1986b). which will be introduced below in this Chapter. Intuitively, the semantics of *studies* is a state *s*, which is occurring at the present time *--PRES(s)--* and it is the state of *x* studying *y*. The specification of a sign can be stated in terms not only of constants, but also of variables, like the order *O*. This allows that incomplete information which will be specified in the combinatorial process is stated in the sign's description. In general, if some or all attributes of a sign *E* are left unspecified but other attributes are specified or cross-identified, then such a sign can be represented as

E(W:C:S:O)

Signs are combined by a rule of functional application, and the combination of two signs yields a sign which has all specified information of the combining constituents. Functional application in *UCG* splits into two separate operations: instantiation and stripping. These operations are defined in (4) and (5) as follows:

(4) S_3 is the *instantiation* of S_1 with respect to S_2 if it results from S_1 by **unifying** its active part with S_2.

(5) Given a sign S_1 with category *A/B*, the result of stripping S_1 is the sign S_2 just like S_1 except that its phonology is the concatenation of S_1's and *B*'s phonology, and its category is stripped down to *A*.

The rule of functional application is as in (6),

(6) Let S_1 and S_2 be well-formed signs. Then stripping the instantiation of S_1 with respect to S_2 results in a well-formed sign.

In *UCG* only adjacent constituents can combine grammatically and the order information is specified by the constants *pre* and *post*. The interpretation of the order information is used in the stripping part of the functional application process. The constants *post* and *pre* are the values that the order variable of the **argument term** takes after the instantiation process. The constant *post* says "if I am an argument in functional application, my functor's phonology follows my own phonology". The constant *pre* says "if I am an argument in a functional application, my functor's phonology precedes my own phonology".

According to this, functional application is realised as two different rules.

(7) **RULE 1:** $(W_f + W_a){:}C{:}S \rightarrow W_f{:}C/E{:}S\ E(W_a{:}pre)$

RULE 2: $(W_a + W_f){:}C{:}S \rightarrow E(W_a{:}pre)\ W_f{:}C/E{:}S$

A simple way to understand these rules is by considering that in both of them the argument sign has been partially specified, and just the attributes of phonology W_a and order are stated. Whenever the active part of the functor sign --whose phonology is W_f -- is unified with the argument sign, the order position of the argument sign is filled with a value, which is either *pre* or *post*. In the stripping process, the rule to be applied is selected according to such a value; if it is *pre* then *RULE 1* is selected, otherwise *RULE 2* is selected. The stripping rule concatenates the phonologies of the functor and argument terms by means of the string combinator symbol "+". Finally, the active part of the functor sign --*E*-- is taken off, and the new object is produced. Note that the resulting signs are underspecified for the order attribute.

4.1.1. Indexed Language and the Semantics of UCG.

The semantic representation language used in *UCG* is called *InL* (for Indexed language). There are only two connectives for building complex formulas: an implication that at the same time introduces universal quantification, and a conjunction. An implication of the form

(8) $[A(x_1,...,x_n) \Rightarrow B(y_1,...,y_n)]$

is interpreted as

(9) $$\forall x_1,...,x_n[A(x_1,...,x_n) => \exists y_1,...,y_n B(y_1,...,y_n)]$$

where $x_1,...,x_n$ are all the variables in *A* outside the scope of any implication occurring in *A*, and $y_1,...,y_n$ are the analogous variables in *B*.

A formula as a whole has an existential interpretation; the formula

(10) $$A(x_1,...,x_n)$$

is interpreted as

(11) $$\exists x_1,...,x_n\ A(x_1,...,x_n)$$

Every formula of *InL* has a designated variable called its index. This variable has a special status, and it designates the ontological type denoted by the formula. By convention, the index variable is always the first argument in the argument list of a formula. Indices report the kind of entity which the formula is about: individuals, states, events, etc. The following expressions are about the kind of thing under the type heading:

(12)		**Expression**	**Type**
	i.	*Luis studied linguistics*	**event**
	ii.	*yesterday*	**an unspecified eventuality**
	iii.	*is in a subject*	**state**
	iv.	*coffee*	**quantity of mass**
	v.	*to the game*	**some entity with a direction**
	vi.	*studied*	**event**
	vii.	*does not*	**absence**

The translations of these expressions in *InL* are as follows:

(13) **i.** **[e] [PAST(e), [e]STUDY(e, LUIS, LINGUISTICS)]**

ii. **[a] [YESTERDAY(a), [a]A]**

iii. **[s] [SUBJECT(y), [IN(x, y), [s]PRES(s)]**

iv. **[m] COFFEE(m)**

v. **[a] [GAME(x), [a][TO(a, x), [a]A]]**

vi. **[e] [PAST(e), STUDY(e)]**

vii. **[s] [A => FALSE]**

In expressions 13**ii**, 13**v** and 13**vii** the variable *A* indicates that such formulas denote functions that have to be combined with some argument in order to yield either the semantics of a sentence, or in general, an expression that can be taken as an argument in a further combination. Variables in the semantics of a functor term are instantiated by the semantics of an argument term in a functional application process. For instance, the semantics of the expression *Yesterday Luis studied linguistics* can be produced by replacing the variable *A* in formula 13**ii** by formula 13**i**, yielding the formula,

(14) **[e] [YESTERDAY(e), [e] [PAST(e), [e]STUDY(e, LUIS, LINGUISTICS)]**

in which the eventuality reported by the word *yesterday* has been made explicit. Note that the variable *A* in 13**ii** has been unified with expression 13**i**, and the unsorted variable **a** has been further specified as an event **e**. Another related consideration is that in the process of functional application, quantifiers over variables of the functor term have always a larger scope than quantifiers over variables in the argument term. This is particularly important for determining the relations of subordination between discourse contexts when the reference of an anaphoric pronoun has to be found.

Formulas 13**i**, 13**ii**, 13**iv** and 13**vi** are either the semantics of a sentence or the semantics of an expression that has to take the role of the argument in a process of combination. The semantics of 13*vii* is more complex. It reports that the object which will be combined with the expression *does not* is false. The sentence *Luis did not study linguistics* reports the *absence* of an event, and within *InL* it would be treated as a special kind of eventuality. However, there is no need to discuss such deep complexities here, and a more detailed account of these issues is given in the original source (Zeevat et al 1986a).

Indices of formulas are semantic variables that are divided into sorts. A sort is a bundle of features associated with a particular variable of referential constant and, as has been illustrated, unification can be performed on sorts. Sort variables are related in a semi-lattice structure, as shown in Figure 4.1.

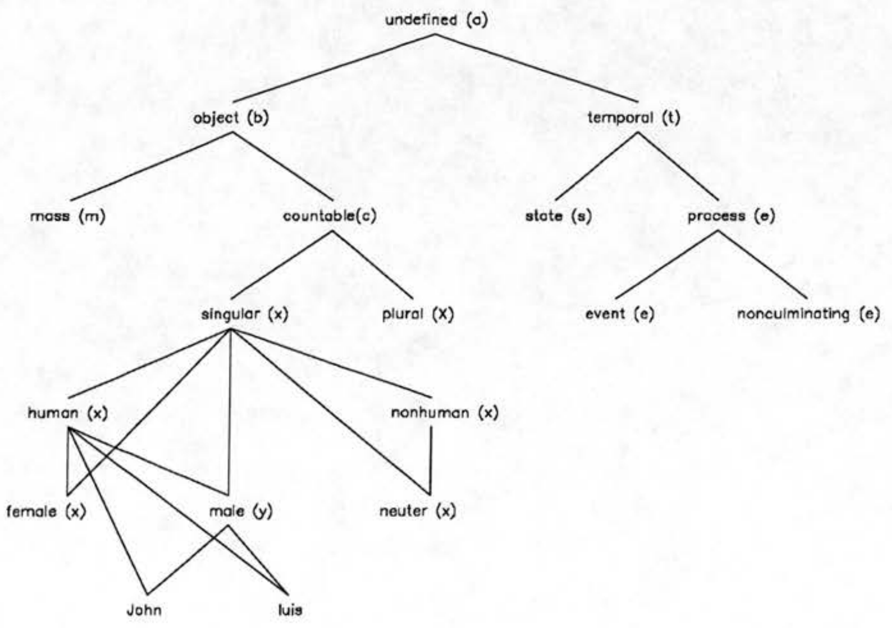

FIGURE 4.1

The index of a lexical entry is specified by a variable whose sort is as far down as possible in the structure in Figure 4.1. For instance, the index for a proper noun is its corresponding constant in the sortal hierarchy: its ontological type cannot be further specified. On the other hand, when the index of a lexical item can take any sortal value, as in the case of *yesterday*, the index is an unsorted variable on the top of the structure. In the process of functional application, the semantics of the composed expression will acquire an index of the greatest possible degree of specificity in the sortal structure. A detailed specification of the sortal structure for *UCG* is presented by Calder et al (Calder et al 1988). Conventionally, variable names are associated with particular sorts. For instance,

object variables	**$x, y, z, x_1, x_2, x_3,...$**
mass variables	**$m_1, m_2,...$**
event variables	**$e, e_1, e_2, e_3,...$**
state variables	**$s, t, s_1, s_2, s_3,...$**
unsorted variables	**$a, b, c, a_1, a_2, a_3,...$**

Alternatively, the sort type can be made explicit in the notation. For instance, the variable *x* might refer to a *male* or to a *female* object. For clarity, we can write *male(x)* or *female(x)* in the index position of a formula.

4.1.2. Type Raising and Quantification.

In categorial grammar and related grammatical formalisms there is certain degree of freedom in choosing which elements are functors and which are arguments in the combinatorial process. In the analysis sketched in Chapter 3, *np's* were considered arguments and verbs were considered the functors. In such a scheme, the meaning of a predicate term is a function that maps the individual denoted by the subject to the truth value of the sentence. For instance, the meaning of the sentence *Luis studies linguistics* is produced by applying the function *studies linguistics* to the argument *luis*. However, in many modern theories, a reversed scheme is used. This new approach was introduced by Montague in 1973 in the *PTQ* system *Proper Treatment of Quantification* (Dowty et al 1981). and it is adopted and generalised in *UCG*. Under this alternative scheme, it is the meaning of the subject term the one which is modelled as a function mapping a predicate into a sentence.

One important reason for taking *np* as functions is that the scope of quantifiers introduced by the subject of the sentence will be larger than the scope of the quantifiers introduced by the object term. In *UCG*, as was mentioned, functors have a larger quantification scope than their arguments. Consider sentence (15)

(15) *every student likes a subject*

Under the previous scheme sentence (15) would be analysed with the term *a subject* having a larger scope than the term *every student*. Such an analysis would imply that there is one subject --as *linguistics*-- which is liked by every student. However, a more intuitive reading for sentence

(15) is that for every student there is a subject that he likes; for instance, Luis might like linguistics, but John might like philosophy, and Pete might like programming. The second analysis can be produced if the subject *every student* is analysed as a function which takes the predicate *likes a subject* as its argument. Here, there is no space to get involved in the very deep issues posed by the analysis of quantified expressions, but it is worth mentioning that the matter gives the title for the best known Montague's paper: Proper Treatment of Quantification. In this system, the analysis of the subject as a function from predicates to sentences plays a fundamental role. Furthermore, the syntax and semantics of a natural language expression bear a closer resemblance to each other, according to the desiderata of the rule-to-rule hypothesis.

In the combinatorial scheme presented in Chapter 3, the category *A/B* has as its meaning the function *f* which combines with an expression of category *B* --whose meaning is the object *b*-- to produce an expression of category **A** and whose meaning is *f(b)*. According to this, in the new scheme the category for proper nouns and quantified terms would have to be of the following form,

Sent/category-of-predicate.

The category of a predicate --a *vp*-- is *Sent/np*: an expression that wants an *np* to become a sentence. Then, the category for proper nouns is **type raised** from *np* to *Sent/(Sent/np)*.

Now, we have to consider the impact for the semantics of such a change. In the old scheme, predicates were considered functions mapping individuals to truth values (*Sent/np*), and the meaning of a proper noun --or of a referring expression-- was the individual denoted by such a term. In the new scheme, the meaning of a type raised *np* is the set of properties that the individual denoted by that name has. This is equivalent to a function that for every property asserted of a particular individual returns the value *true* if such an individual has in fact that property, and *false* otherwise. Formally, this is a function from individuals to functions from properties to truth values, and it can be represented as the following Lambda calculus expression (Dowty et al 1981),

(16) **λxλP[P(x)]**

where *P* is a high-order variable standing for the set of properties that some individual *x* has. The meaning of the proper noun *Luis* would be the function,

(17) **λxλP[P(x)](luis)**

or by lambda conversion

(18) **λP[P(luis)]**

and it denotes the set of properties of Luis. We can apply this function to the predicate *walks* as its argument, as in

(19) **λPP(luis)(walks)**

yielding the predication

(20) **walks(luis)**

which would be true if the individual Luis is in the set of individuals who happen to be walking in the model.

Now, we can come to the first approximation for the definition of the lexical entries for proper nouns in *UCG*. Provisionally, the lexical entry for *luis* is defined as follows:

(21) **luis**
Sent/W:(Sent/luis:np:LUIS:O):S:O
S

This is a sign whose phonology is the name *Luis*. The category of this sign is of the form *A/B* where the *A* is a sentence, and *B* --the active part-- is the sign *W:(Sent/luis:np:LUIS:O):S:O* which itself contains a complex category of the form

(Sent/luis:np:LUIS:O)

Note that the semantics of the active part is the variable *S*, which is also the semantics of the sign in (21) as a whole. The easiest way to understand how such a sign works within the system is by showing a combination process. Consider the sign for the intransitive verb *walks*,

(22) **walks**
Sent/W:np:x:pre
[e][PRES(e), WALK(e, x)]

Now, consider the production of sentence (23)

(23) *Luis walks*

in which *Luis* is the functor term, and *walks* the argument term. In the instantiation process the active part of the functor is unified with the argument sign

(24)	*Active part of Luis*	*sign for walks*
	W	**walks**
	Sent/luis:np:LUIS:O	**Sent/W_1:np:x:pre**
	S	**[e][PRES(e), WALK(e, x)]**
	O	**O**

The most general unifier for these two signs is:

(25) **σ = {W := walks, luis := W_1, LUIS := x, O := pre, S := [e][PRES(e), WALK(e, x)]}**

In addition, the constants *Sent* and *np* in both functor and argument have to match to allow the unification of these two signs. After the instantiation process, functor and argument take the following values,

(26)	*Active part of Luis*	*sign for walks*
	W	**walks**
	Sent/luis:np:LUIS:pre	**Sent/luis:np:LUIS:pre**
	[e][PRES(e), WALK(e, LUIS)]	**[e][PRES(e), WALK(e, LUIS)]**

pre **pre**

and the whole sign for Luis is instantiated as,

(27) **luis**

Sent/walks:(Sent/luis:np:LUIS:pre):[e][PRES(e), WALK(e, LUIS)]:pre

[e][PRES(e), WALK(e, LUIS)]

pre

Now, for the stripping process we can look at the value of the order attribute of the argument sign in (26) which is *pre*. This value was originally stated in the active part of the sign *walks* in (22) and it was passed to the order of the whole sign for walks, via the order variable in the functor sign. According to this value, the functional application rule *RULE 1* in (7) has to be applied. The value *pre* says "If I am the argument in a functional application my functor's phonology precedes my own phonology". Then we know that the phonology of the final sign will be *luis + walks*. In the stripping process, the active sign of the functor term is taken off, and the object produced by this functional application process is,

(28) **luis walks**

Sent

[e][PRES(e), WALK(e, LUIS)]

The final object is underspecified for the order attribute, as stated in the functional application rule. Note that in the unification process, the values in the argument term instantiate the variables of the functor term, with the exception of the value of the constant *LUIS* which flows in the opposite direction. Contrast this with the evaluation of the corresponding lambda expression,

Meaning of *luis*:	**λP[P(luis)]**
Function and argument:	**λP[P(luis)](walks)**
Lambda conversion:	**walks(luis)**

The meaning of the lexical entry for *Luis* is a function from *Luis* to the properties that he has, and this function maps predicates --as *walks*-- into sentences as *Luis walks*.

There is still one consideration that has to be made for the definition of lexical entries for proper nouns. The definition of the entry for *Luis* in (21) is not fully satisfactory because it is only useful for the proper noun occurring in subject position. However, these terms can also occur in object position and a more flexible scheme is required. In *UCG* this flexibility is achieved by generalising the notion of type raising. The category of raised terms is not *Sent/(Sent/np)* but rather *C/(C/np)*, where *C* is a variable over complex categories and for that reason is referred to as a **polymorphic category**. It is easy to see that when a proper noun is in subject position, the variable *C* takes the value of *Sent*, and the derivation shown in the previous Section is produced. However, when that noun is in object position, such a variable receives a more complex value. In general, polymorphic variables act as a bridge that passes category values among complex expressions. The use of this variables can be illustrated by the derivation of the expression (29)

(29) *Luis studies linguistics*

in which the lexical entries for the proper nouns are specified in terms of type raised categories. The categories for these three words are as follows[1]

(30) **luis**
C/W:(C/np:LUIS:O):S:O
S

(31) **linguistics**
C/W:(C/np:LINGUISTICS:O):S:O
S

(32) **studies**
Sent/np:x:pre/np:y:post
[s][PRES(s), STUDY(s, x, y)]

The derivation is illustrated by the tree in Figure 4.2

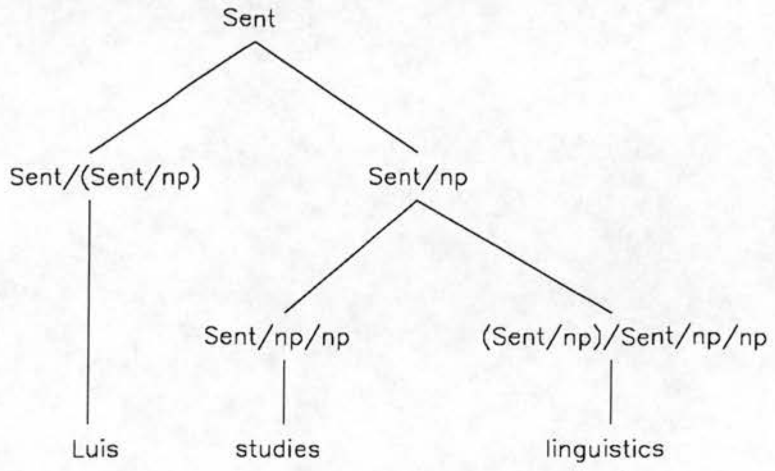

FIGURE 4.2

The first step in the derivation process is to compose the predicate term, by combining the signs (31) and (32). Note that the functor is the sign standing for the noun in (31). The active part of the functor and the argument sign are unified. These signs are,

(33)	*Active of linguistics*	*sign for studies*
	W	**studies**
	C/np:LINGUISTICS:O	**Sent/np:x:pre/np:y:post**

[1] The redundant phonology attributes are omitted in this description. As was shown, they do not take any significant role in the derivation process.

S	**[s][PRES(s), STUDY(s, x, y)]**
O	**O**

The most general unifier for these two signs is,

(34) **σ := {W := studies, C := Sent/np:x:pre, LINGUISTICS := y,
O := post, S := [s][PRES(s), STUDY(s, x, LINGUISTICS)]}**

The sign resulting from functional application is of the form,

(35) **studies linguistics**
C
S

where the values for the variables *C* and *S* are instantiated as shown in the substitution σ in (34). In the stripping part of the functional application process these values are substituted in the functor sign as a whole, and the following sign is produced.

(36) **studies linguistics**
Sent/np:x:pre
[s][PRES(s), STUDY(s, x, LINGUISTICS)]

The combination between subject and predicate is produced in an analogous manner. The subject is the functor, and the predicate the argument. The active part of *Luis* and the sign for the predicate *studies linguistics* are,

(37)	*Active of Luis*	*sign for studies linguistics*
	W	**studies linguistics**
	C/np:LUIS:O	**Sent/np:x:pre**
	S	**[s][PRES(s), STUDY(s, x, LINGUISTICS)]**
	O	**O**

The most general unifier for these two signs is,

(38) **σ := {W := studies linguistics, C := Sent, LUIS := x,
O := pre, S := [s][PRES(s), STUDY(s, LUIS, LINGUISTIC)]}**

Now, these values are substituted in the sign for *Luis* and after the stripping process, the object

(39) **Luis studies linguistics**
Sent
[s][PRES(s), STUDY(s, LUIS, LINGUISTICS)]

is produced.

With this example, the introduction to *UCG* is concluded. The intention in this section is to present a general view of the theory. It is worth recalling that the definition of a grammar segment consists mainly in the specification of the lexical entries. The expressions produced by the grammar will be those permitted by the functional application process as defined above. In a similar way, the grammatical system will rule out expressions which are blocked by failures in the unification process: when the active part of a sign cannot be unified with the argument sign. In the Section 4.2 the lexical entries for our linguistic fragment are shown, and the derivation for some of the most relevant signs of the dialogue of Chapter 2 are illustrated. Finally, in Section 4.3 the deictic use of pronouns and demonstratives within this framework is presented.

4.2. The Linguistic Fragment.

In this Section, the linguistic fragment from which the dialogue of Chapter 2 is produced is presented. As was mentioned, this is mainly a matter of describing the lexicon. The following rules recapitulate the syntactic and semantic description of a sign:

(40)	**sign**	**→ {phonology:category:semantics:order, E}**
	phonology	**→ {string, W}**
	category	**→ {Sent, np, noun, category/sign, C}**
	order	**→ {pre, post}**
	semantics	**→ {atom, [variable][semantics, semantics], [variable][semantics => semantics], S, [a]S}**
	atom	**→ predicate(arg*)**
	argument	**→ {variable, constant, semantics}**
	variable	**→ sort integer**

Now we come to the presentation of the linguistic fragment.

4.2.1. The Linguistic Dialogue in GRAFLOG.

In the previous Section, the definition of proper nouns received a great deal of attention. However, proper nouns are not the only terms whose category is a type raised *np* or *C/(C/np)*. Terms of similar grammatical category are pronouns such as *he, she* and *it*. Besides, expressions composed by an article and a common noun have a similar category. Examples of these expressions are *The student, a student, every student, the clever student, a clever student*, etc. Other words that have the same grammatical category are *wh*-terms as *who, what, which*, etc. All of these expressions have a thing in common: their function is to denote an entity or a set of entities. Furthermore, words like *the, a* and *every* introduce quantification over the terms that lie within their scope, and for that reason play a very important role in the representational system.

We start the exposition of the linguistic fragment by emphasising the analysis of referring expressions. Consider expressions (41) and (42).

(41) *He studies it.*

(42) *He studies that subject.*

Here, we assume that the pronouns are anaphoric and their antecedent must be found in the previous discourse representation. The alternative deictic use of these terms in which they are supported by an ostensive gesture is presented below in Section 4.3. The lexical entries for the pronouns *he* and *it* are as follows,

(43) **he**
C/W:(C/np:x:O):[a]S:O
[a][[x]PRO(male(x)), [a]S]

(44) **it**
C/W:(C/np:x:O):[a]S:O
[a][[x]PRO(neuter(x)), [a]S]

The syntactic category of these words is similar to the category of proper nouns. Compare, however, the variable *x* in (43) and (44) with the constant *LUIS* in the semantics of the proper noun in (30). The semantics of a pronoun is a function from an unspecified individual --denoted by a variable-- to the properties that it has. Such variables are not unrestricted. They belong to certain specific sort. The terms *male(x)* and *neuter(x)* in (43) and (44) indicate the sort of the corresponding individual in terms of the semi-lattice structure of Figure 4.1. The word *PRO* is just a semantic constant useful for the interpretation process; in particular, it is used by the procedure for finding out the antecedent of an anaphoric pronoun. It indicates that its argument is a *PRONOMINAL* variable. The index for the semantics is an unsorted variable, and it will be specified in the combinatorial process.

The word *that* in sentence (42) is a determiner. In this analysis, *that* is considered an alternative lexical realisation of the word *the.* This is a simplification of the facts, because the word *that* connotes a 'remote' relation between the speaker and the thing referred to by the constituent in which the determiner is included. However, we ignore this fact. The grammatical category of a determiner is specified as something that combines with a term of category *noun* to yield an *np.* In *UCG* such an *np* is replaced by the corresponding type raised category. The definition of its lexical entry is as follows,

(45) **that**
C/W:(C/np:b:O):[a]S:O/noun:[b]R:pre
[a][[b]R, S]

The word *that* is a functor term that will be combined with a noun of semantics *R* and index *b*.

The category for the word *subject* is *noun* and its definition is as follows.

(46) **subject**
noun
[x]SUBJECT(x)

Now, we can combine the determiner *that* in (45) with the noun *subject* in (46) to produce the sign in (47)

(47) **that subject**
C/W:(C/np:y:O):[a]S:O
[a][[y]SUBJECT(y), S]

The application of the functor *that subject* in (47) to the argument *studies* in (32) yields the following predicate sign (48)

(48) **studies that subject**
Sent/np:x:pre
[s][SUBJECT(y), [s][PRES(s), STUDY(s, x, y)]]

Finally, the application of the functor *he* in (43) to the argument *studies that subject* in (48) yields the sign (49)

(49) **he studies that subject**
Sent
[s][PRO(male(x)), [s][SUBJECT(y), [s][PRES(s), STUDY(s, x, y)]]]

The derivation tree for the expression sentence (41) is similar that the one shown in Figure 4.2, but the words *Luis* and *linguistics* are replaced by *he* and *it* respectively. The sign for sentence (41) is,

(50) **he studies it**
Sent
[s][[x]PRO(male(x)), [s][[y]PRO(neuter(y)), [s][PRES(s), STUDY(s, x, y)]]]

Now, we come to the analysis of the sentence

(51) *A student is in a subject.*

is important in two respects: it shows the use of an indefinite expression in both subject and object positions, and it shows the verb *to be* in conjunction with a prepositional phrase in which the word *in* denotes a spatial relation between the object and the subject terms. In Figure 4.3, the derivation tree for this expression is shown.

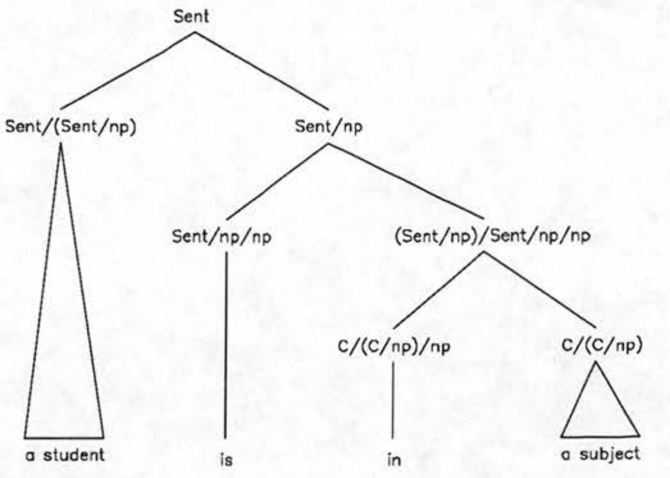

FIGURE 4.3

The definition of the article *a* is as follows,

(52) **a**
C/W:(C/np:b:O):[a]S:O/noun:[b]R:pre
[a][[b]R, S]

The sign for *a subject* is,

(53) **a subject**
C/W:(C/np:y:O):[a]S:O
[a][[y]SUBJECT(y), S]

and the sign for *a student* has a similar form.

The lexical entry for the preposition *in* is the most complex basic sign in this grammar. It is a functor that wants an *np* on its right to become another functor that wants a verb --in some unspecified location-- to become a predicate. Its lexical entry is,

(54) **in**
C/(C/np:x:O):[a]S/np:y:post
[a][IN(x,y), S]

This term combines with the sign in (53) to yield the sign in (55)

(55) **in a subject**
C/(C/np:x:O):[a]S
[a][SUBJECT(y), [IN(x,y), S]]

The definition of the lexical entry for the verb *is* is as follows,

(56) **is**
Sent/np:x:pre/np:y:post
[s]PRES(s)

Note that the semantics of the sign in (56) introduces a present state, but it does not introduce a relation between the individuals referred to by subject and predicate. The function of this *is* is mainly syntactic. Without this word, the expression would be a modified noun --*a student in a subject*-- and not a sentence.

The combination of (55) and (56) two signs yields the predicate (57)

(57) **is in a subject**
Sent/np:x:pre
[s][SUBJECT(y), [IN(x, y), PRES(s)]]

Finally, the sign for *a student* which is similar to (53) is applied to the predicate *is in a subject*, and the sign in (58) is produced,

(58) **a student is in a subject**
Sent
[s][STUDENT(x), [s][SUBJECT(y), [IN(x, y), PRES(s)]]]

Now, we come to the analysis of a rule through which the interpretation of a spatial relation --like *in*-- is introduced. Consider sentences (59), (60) and (61),

(59) *If a student is in a subject then he studies it.*
(60) *If a student is in a subject then he studies that subject.*
(61) *every student who is in a subject studies it.*

The words *if* and *then* are sentence modifiers. Their corresponding lexical entries are,

(62) **if**
Sent/Sent:S:pre/Sent/T:pre
[s][T => S]

(63) **then**
Sent/Sent:S:pre
S

The derivation tree for sentences (59) and (60), as well as their relation to the discourse context was shown in Figure 3.12. The sign for sentence (59) is as shown in (64)

(64) **If a student is in a subject then he studies it**
Sent
$[s_2][[s_1][\mathrm{STUDENT}(x_1), \mathrm{SUBJECT}(y_1), \mathrm{IN}(x_1, y_1), \mathrm{PRES}(s_1)]$
$\Rightarrow [s_0][\mathrm{PRO}(\mathrm{male}(x_0)), \mathrm{PRO}(\mathrm{neuter}(y_0)), \mathrm{PRES}(s_0), \mathrm{STUDY}(s_0, x_0, y_0)]]$

Note that the variables in the antecedent and consequent clauses of the rule are not yet bound. Strictly speaking they are different variables. However, the clauses indexed by the state s_1 and s_0 are properly embedded (Kamp 1981) in the whole structure indexed by s_2, and the anaphoric link between the variables in antecedent and consequent clauses is permitted. The referents for the anaphoric variables are found by the resolver procedure yielding the semantic expression,

(65) **$[s_2][[s_1][STUDENT(x), SUBJECT(y), IN(x, y), PRES(s_1)]$**
$\Rightarrow [s_0][PRES(s_0), STUDY(s_0, x, y)]]$

where the variables are properly bound. In this representation the indices for atomic expressions have been omitted, and some associative parentheses have been dropped. However, the order of the atomic formulas is preserved because according to the *InL* semantics, it reflects the scope of the quantified terms. The pronominal predicates *PRO* in the consequent clause can be dropped because their only purpose is to establish the sort of the individuals denoted in the *STUDY* predicate.

The representation for sentence (60) is produced on similar basis. However, the task of the resolver is slightly different because the anaphoric link is establish considering not only sortal information, but also a property of an individual denoted in both antecedent and consequent clauses: the fact that the individual denoted by the variable *y* is a subject.

The meaning of sentence (61) is very close to the meaning of sentences (59) and (60). The word *every* introduces an implication in which variables in the antecedent clause *a student who is in a subject* are universally quantified. The link between the pronoun *it* and the indefinite term *a subject* is permitted because both are properly embedded within the discourse context set by the word *every* according to Kamp's discourse model (Kamp 1981). The lexical entry for the word *every* is

(66) **every**
C/W:(C/np:b:O):[a]S:O/noun:[b]R:pre
[a][[b]R => S]

and it takes a noun --i.e. *student*-- to become the type raised *np* --i.e. *every student*-- which maps the predicate *who is in a subject studies it* into a sentence.

4.2.2. The semantic roles of the verb *to be.*

In the dialogue of Chapter 2, the word *is* was used in several expressions. However, the function of this word varies significantly according not only to the different syntactic constructions in which it appears, but also to the context in which the same expression is used. Here, a small section on the meaning of this word is worthwhile. Consider first the word *is* above in (51); its only semantic contribution is to introduce the reference to a present state *s*. Now, contrast the words *is* in (51) and (67).

(67) *Luis is happy.*

Expression (67) expresses that the individual denoted by the term in subject position has the property asserted in the predicate. The purpose of this second *is* is to make such an attribution. In (67), *happy* denotes the set *happy(y)* of happy individuals. The semantics of the *is* term includes the predicate *identify(s, x, y)* which is read as asserting that *x* is some *y* in the set *happy*. The sign for this 'identifying' *is* is,

(68) **is**

Sent/np:x:pre/np:y:post

[s][PRES(s), IDENTIFY(s, x, y)]

The sign for the predicative adjective *happy* is,

(69) **happy**

C/(C/np:y:O):[a]S

[a][HAPPY(y), S]

and the sign for the predicate *is happy* is

(70) **is happy**

Sent/np:x:pre

[s][HAPPY(y), PRES(s), IDENTIFY(s, x, y)]

The combination of this predicate with the proper noun *Luis* in (30) yields the sign,

(71) **Luis is happy**

Sent

[s][HAPPY(y), PRES(s), IDENTIFY(s, LUIS, y)]

The semantics of this last sign indicates that *Luis* is identified as one y, or that he is in the set of individuals who happens to be happy. The word *in* in (51), on the other hand, asserts an spatial relation, and not a set theoretical inclusion relation.

There are still other roles of the word *is*. One is to assert that the individual denoted by subject and predicate are the same. Consider (72),

(72) *Luis is luis*

Although this expression is tautological, it can be considered grammatical, and the semantics of the lexical entry for this *is* is defined as,

(73) **is**

Sent/np:x:pre/np:y:post

[s][PRES(s), EQUALITY(s, x, y)]

There is still another meaning for the word *is*: its use in ostensive definitions. Such a role is crucial for our enterprise. However, the analysis for the ostensive *is* will be delayed until the model for dealing with deixis and demonstration is introduced below in Section 4.3.

4.2.3. Modifiers.

Before concluding the exposition of the linguistic fragment, a word on noun modifiers is worthwhile. In *UCG*, modifiers --like the attributive adjective *happy*-- are easily represented. In the expression (74)

(74) *Luis is a happy student*

the adjective *happy* marks the individual Luis as an individual who happens to be not only within the set of students, but also within the set of happy individuals. The lexical entry for the word *is* in (74) is the attributive form in (70). Expression (74) marks Luis within the intersection of the sets of all the students and all the happy individuals. The definition for *happy* is

(75) **happy**
noun/noun:[a]A:pre
[a][HAPPY(a), A]

and it is similar for other attributive adjectives as well. The sign for the modified noun is, for instance,

(76) **happy student**
noun
[x][HAPPY(x), STUDENT(x)]

The combination of (76) and the functor *a* in (52) for producing the sign for the *np a happy student* is straightforward. With this example, the exposition of the linguistic component of GRAFLOG is concluded. The detailed explanation of other linguistic phenomena as for instance, word order inversion in questions, unbounded dependencies and plural terms are developed in the lines presented here and in Chapter 3. A formal account of those issues has been developed for the ACORD project (Calder et al 1988). Finally, it is worth mentioning that the full power of *UCG* has not been shown. In *UCG*, the categories *Sent* and *np* carry syntactic feature specifications. For instance, finite and non-finite are features of sentences, and grammatical case is distinguished for *np's* (Zeevat et al 1986a). The explicit use of this information imposes a further constraint on the set of expressions that are produced by the grammar. However, it is worth pointing out that this feature of *UCG* can easily be augmented in the linguistic fragment presented in this chapter. Now, we come to the relation between the language and the graphical representations.

4.3. Deixis, Demonstration and Ostension.

In this Section, the mechanism by which the relation between linguistic and graphical representations is established is presented. This mechanism is essentially a model for dealing with the phenomena of deixis. The intuitive ideas behind this notion were introduced in Chapter 2. In particular, some examples of the deictic use of demonstrative pronouns were discussed. Here, the same notion is extended to model the deictic use of pronouns. Consider the following piece of discourse,[2]

(77) *Luis studies linguistics. He likes that.*

This piece of discourse can be analysed without meta-linguistic consideration along the lines shown in Section 4.2. However, if at the time the pronouns *he* and *that* are uttered their referents are ostensively pointed out, the meaning of such terms can vary. Suppose that at the time *he* is uttered, the speaker points to John instead of Luis, and at the time *that* is uttered, the speaker points to John's car. Then the expression *he likes that* in (77) would mean *John likes his car* and not *Luis likes to study linguistics*. A model for dealing with indexical terms would allow us to capture such differences in meaning.

It is very intuitive that ostensive gestures supporting speech acts provide information that come from out side the linguistic system. According to Kamp (Kamp 1981, p. 283)

> and anaphoric pronouns select their referents from certain sets of antecedently available entities. The two pronoun uses differ with regard to the nature of these sets. In the case of a deictic pronoun the set contains entities that belong to the real world, whereas the selection set for an anaphoric pronoun is made up of constituents of the representation that has been constructed in response to antecedent discourse.

The model that has been developed here for dealing with these indexical expressions is based on providing an interface between the linguistic system and the world. Of course, 'the world' is related to the linguistic system by an intermediate representational structure: an analogical representation, or rather, a system of symbols realised in a non-linguistic medium.

The model for dealing with indexicals has the following three conceptual constituents:

(*a*) an indexical one-place predicate that is incorporated in the meaning of indexical words, namely *deictic(x)*, where x is a variable whose interpretation is given out side the linguistic system.

(*b*) A trans-substantial relation between graphical symbols and their linguistic names. This relation is established by the verb *is* when it is used in ostensive definitions.

(*c*) A procedure by which the reference of a graphical symbol is assigned as the interpretation of the indexical variable in some specific context.

In this Section points (*a*) and (*b*) of the model are formally presented. The procedure (*c*) has been presented in Section 2.2 and it is not further discussed.

The definition of the deictic pronouns *he* and *that* is as follows,

(78) **he**
C/W:(C/np:x:O):[a]S:O

[2] Here, the definition of *likes* is similar to the definition for *studies*.

[a][[x][PRO(male(x)), deictic(x)], [a]S]

(79) **that**

C/W:(C/np:x:O):[a]S:O

[a][[x]PRO(neuter(x)), deictic(x)], [a]S]

The derivation of the sentence

(80) *he likes that*

in which one or both pronouns are deictic would be produced in the lines shown above. In the case that both of the pronouns are deictic the sign for (80) is,

(81) **he likes that**

Sent

[s][[x][PRO(male(x)), deictic(x)],

[s][[y][PRO(neuter(y)), deictic(y)], [s][PRES(s), LIKE(s, x, y)]]]

Now consider sentence (82),

(82) *If a student is in a subject then he likes it*

If the pronoun *he* in (82) is supported by physical gesture, or by pointing act in computer graphic interaction, the sign for (82) is,

(83) **If a student is in a subject then he likes it**

Sent

[s_2][[s_1][STUDENT(x_1), SUBJECT(y_1), IN(x_1, y_1), PRES(s_1)]

=> [s_0][PRO(male(x_0)), deictic(x_0)],

[PRO(neuter(y_0)), PRES(s_0), LIKE(s_0, x_0, y_0)]]

and the reference for the variable x_0 would have to be taken from the graphical domain before the anaphoric resolver could be evoked. The variable y_0 stands for the pronoun *it*, and it is anaphoric.

4.3.1. Demonstrative Pronouns and Ostension.

Demonstrative pronouns, such as *this*, relate language and space in a very direct fashion. The deictic use of pronouns in expressions like (82) might not seem very natural in normal discourse; however, that is a contextual rather than a grammatical factor. Sentence (82) is the kind of expression that was used for introducing the interpretation of spatial relations in Chapter 2. Furthermore, if a graphical context is fully considered, the interpretation of the relation *in* introduced by (82) can be expressed in a fairly direct way by using demonstrative pronouns. Consider the discourse in (84)

(84) *If* $this_0$ *is in* $this_1$ *then* he_0 *likes* it_1

where the subscripts indicate the symbol that is pointed out on the screen at the time the demonstration is made. The two instances of the word *this* act as proper nouns with a deictic component, and the lexical entry for the demonstrative can be defined as the sign,

(85) **this**
C/W:(C/np:x:O):[a]S:O
[a][deictic(x), [a]S]

Now, contrast (84) and (86),

(86) *If this$_0$ student is in this$_1$ subject then he$_0$ likes it$_1$*

In (86) the word *this* is a determiner with a deictic component, and its definition is the sign,

(87) **this**
C/W:(C/np:b:O):[a]S:O/noun:[b]R:pre
[a][[b][deictic(b),R], [a]S]

The production of the signs for (84) and (86) is developed in the general framework provided by *UCG*. Here, it is worth recalling the values of deictic variables are taken from the graphical domain, in conjunction with explicit ostensions. The instantiation for such deictic variables is a task for the Dialogue Manager, on the lines shown in Chapter 2. It is also worth noticing that many linguistic expressions will accept both deictic and anaphoric interpretations, and the grammatical system would produce a sign for all such analyses. In fact, there are two additional interpretations for the word *this* that have no deictic interpretation: the anaphoric *this* and the determiner *this*. These words are respectively defined in (89) and (90),

(89) **this**
C/W:(C/np:a:O):S:O
S

(90) **this**
C/W:(C/np:b:O):[a]S:O/noun:[b]R:pre
[a][[b]R], [a]S]

In the resolution process, the distinction between a deictic and an anaphoric form will be determined by the constraints on the pronominal resolution, the general knowledge about the entities that are referred to by the linguistic expression, and most importantly, by the presence or absence of an overt ostensive act.

Now, we come to the analysis of the most basic use of a deictic word: definition by ostension. In the analysis presented above, the deictic words were always used to refer to an entity that is graphically represented; however, in such an analysis we took for granted that the association between the linguistic and graphical signs had already been established. As was shown in Chapter 2, the most basic operation in our system is to establish such an association. Consider the ostensive definition in (91)

(91) *This is Luis*

When this expression is uttered with the purpose of introducing Luis by the very first time --as in the example of Chapter 2 in Figure 2.1-- a graphical symbol is baptised with the linguistic name. Then, both graphical and linguistic symbols stand in a trans-substantial relation as was explained in Section 2.2. Here, the word *is* has the purpose of establishing such a relation. In GRAFLOG this relation is represented as *translate(s, x, y)* and it is part of the semantics of the ostensive *is* which is defined as,

(92) **is**

Sent/np:x:pre/np:y:post

[s][PRES(s), TRANSLATE(s, x, y)]

For the production of the ostensive definition *This is Luis* the 'proper noun version' of the demonstrative *this* in (86) is used, and the sign for the definition is

(93) **This is Luis**

Sent

[s][deictic(x), [s][PRES(s), TRANSLATE(s, x, LUIS)]

The variable x in (93) **must** be interpreted by taking a referent from the graphical representation. The purpose of the definition *This is Luis* is to establish an association between the analogical representation of the individual Luis and the linguistic symbol *Luis*. The linguistic symbol is the index of the analogical one. With this last analysis, this chapter on the linguistic fragment of GRAFLOG is concluded.

Here, a concluding remark on the the linguistic section of this dissertation is worthwhile. The linguistic fragment that has been presented is representative of the kind of dialogue that might be natural in computer graphics interaction; however, it comprehends a fairly small fragment of English, and for a practical system it has to be enhanced significantly. What is needed for developing graphical and linguistic human-computer interfaces is not a complete theoretical account of human language --maybe this is not a fully coherent or even meaningful ambition-- but rather to achieve a good compromise between linguistic generality and computational feasibility. One of the purposes of this work is to show that *UCG* --and the related grammatical formalisms-- provides a framework for this sort of enterprise. If this compromise is found, graphical and linguistic interfaces might turn out to be a good thing. Now, we can come to the second part of this dissertation: the structure of graphical representations.

Chapter 5

An Algebraic Structure for Graphics

Early this century, the linguist Ferdinand de Saussure advanced some fundamental assumptions about the nature of human language (Saussure 1974).

Saussure defined language as a system of signs. A sign is a unit consisting of a *signifiant* and a *signifié*. The *signifiant* is the external aspect of the sign, and it is realised in a physical medium such as light, sound, etc. A signifiant is a **distinction** that can be noticed by the physical senses: a pattern of colours, of sounds, of tastes, a tactile stimulus, etc. Combinations of these patterns build up messages or linguistic expressions. Messages refer to states of affairs in the world and they can be understood by a human interpreter if he or she has knowledge of the language, knowledge of the world, and is able to establish a correlation between these two. The *signifié*, the internal aspect of the sign, is its meaning.

Although the notion of system of signs was originally thought of in relation to natural language, Saussure advanced its generalisation to other communication systems (Saussure 1974, p. 16).

> Language is a system of signs that express ideas, and is therefore comparable to a system of writing, the alphabet of deaf-mutes, symbolic rites, polite formulas, military signals, etc. But it is the most important of all these systems.
>
> A science that studies the life of signs within society is conceivable... I shall call it *semiology*.

Along the lines of this semiological spirit, we define a graphical structure, a graphical language, as a system of graphical signs. A graphical sign would have an external aspect and a meaning as well. The graphical *signifiant* is realised in light and is a distinction extracted from the visual whole. Its meaning, the graphical *signifié* is the object in the world that the graphical sign stands for.

For Saussure, a crucial property of the linguistic sign is its arbitrary nature. That is, the pairing between the external and internal aspects of a sign is a matter of convention. The system of signs making up a natural language exists as a set of social conventions. It is supported by the linguistic acts of individual speakers, but cannot be reduced to them. In the definition of artificial graphical languages the relation between the external aspect of a graphical sign and its referent can be established by convention too. In GRAFLOG, the association between a graphical sign and its meaning is set by means of an ostensive definition, as was shown in Chapters 2 and 4. However, these conventional interpretations differ from natural language in as much as they can be changed in the course of a graphics interactive session. These conventions lack the social dimension, and we can profit from this flexibility. Now, we come to the definition of a simple graphical language.

5.1. A Graphical Language for the Blocks-World $\mathbf{L_g}$.

Suppose that at some particular time and place there is a table which supports some boxes and pyramids. Suppose, in addition, that this table and the blocks on it constitute a 'world' which can be described by a graphical language. Following tradition, we call it Blocks World (*BW*). For instance, the arrangement of lines shown in Figure 5.1.a. represents a situation in which the table (shown as a horizontal line) supports two boxes. The left-hand box in turn supports a pyramid. We regard the drawing as an expression which either denotes, or fails to denote, a configuration of objects in *BW*. In either case, it is a *meaningful* graphical expression, in the sense that it does represent a possible configuration of objects in the intended model. By contrast, the drawing in Figure 5.1.b is illegitimate --there is no possible configuration of objects in *BW* whereby a block is suspended some inches above the table top. Thus, relative to the intended interpretation, we regard certain graphical expressions as being *ill-formed*.

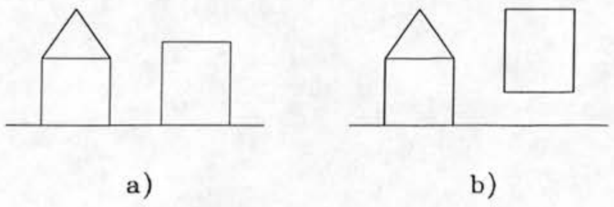

FIGURE 5.1

In fact, there are two sets of constraints on meaningful expressions in graphical languages of the kind we have in mind. These correspond to the notions of lexicon and syntax in linguistic theory. At the lexical level, we have to ensure that expressions belong to the right sort of 'vocabulary'. For example, the basic expressions of our graphical language --which we shall call $\mathbf{L_g}$-- consist of horizontal lines, rectangles and triangles. Other kind of graphical elements, such as circles, fail to belong to this lexicon, in the same way that *hogar* fails to be a member of the lexicon of English. At the syntactic level, the basic expressions have to be arranged in particular ways. For example, blocks have to abut either the horizontal line or other blocks.

A possible objection to our view of drawings as expressions of a language is the following: a language like English succeeds as a medium of communication because the knowledge of the conventions governing its use and interpretation is shared amongst a community of speakers.

On the other hand, there are no generally shared conventions governing the interpretation of drawings.

Although there are particular classes of drawings that are understood in terms of a set of agreed conventions, like architectural and engineering drawings, we can concede most of this point of view without necessarily abandoning our claim that it is useful to treat drawings as expressions of a graphical language. People do succeed in associating interpretative conventions with drawings in particular contexts of use, apparently by extrapolating general principles of interpretation from a small number of cases. For instance, suppose we are presented with the 'corpus' of drawings in Figure 5.2. It is possible to suppose that we would regard 5.1.a as belonging to the same family of representations. Moreover, whatever interpretation was assigned to 5.2 would presumably generalise to 5.1.a.

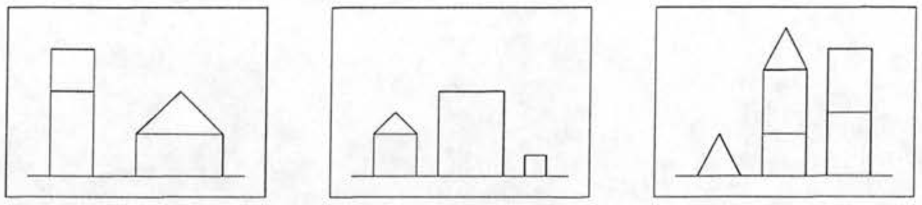

FIGURE 5.2

Now we proceed to induce from such a sample the rules from which a specific family of drawings is produced. Our aim here is not to model the induction itself, though such a model would be a part of a comprehensive theory of visual perception and inference, but just to specify the set of rules from which such drawings might be produced. There is also the requirement that such a set of rules would prevent the production of ill-formed drawings. We assume, for analytical purposes, that we look at the pictures in the sample by the same projection method: the same perspective and illumination conditions. This is of course an idealisation, but it can be considered for our particular purposes.

In relation to the semantics of this language, we can consider that if we were looking at a drawing of *BW* and were asked whether or not it is a 'name' for the the current state of affairs on the table --the world that is referred to-- we would have to compare drawing and world. If the message --the graphical representation-- corresponds to the world, the table and the blocks themselves, then the message denotes. If message and world are not alike, the drawing fails to denote that state of affairs. Ungrammatical or ill-formed drawings, on the other hand, do not refer to possible situations in the blocks world and they are ruled out on a syntactic basis. However, we do not want to say that ill-formed drawings, like Figure 5.1.b, are completely meaningless. If some of the contingent properties of the symbols constituting an ill-formed drawing had been different, like their spatial positions, the same of set of symbols would constitute a well-formed drawing. Furthermore, there is some sense in saying that we can measure how much an irregular drawing diverges from another drawing that is well-formed. Consider a linguistic analogue of this phenomena: the sentence *Being and Time has a cover blue* is not grammatical, but the sentence *Being and Time has a blue cover* is. In both of these sentences, each individual word belongs to the English lexicon, but one particular constituent of the ungrammatical sentence is ill-formed.

Now, let us define a graphical grammar from which the graphical expressions in Figure 5.2 can be produced. We have seen that there are three kinds of symbols: triangles, rectangles and lines. These symbols are placed in a 2-dimensional coordinate space. Every position in such a space is an ordered pair $<x, y>$ of point positions. We define a normalised coordinate system as one in which both x and y have a value in the real interval $[0, 1]$. There is just one line in each drawing and it represents the table. We consider its position horizontal at $y = 0$ in all well-formed drawings. In fact, the extreme points of this line are in positions $<0, 0>$ and $<1, 0>$. We describe rectangles in terms of a parametric position --the bottom left corner-- and the parameters of width and height as shown in Figure 5.3.a. Triangles are defined in terms of their parametric position and their base, as shown in Figure 5.3.b

We can also observe that there are two kinds of complex figures: those with a rectangle on the top, and those with a triangle on the top. A configuration with a rectangle on top will be classified as a rectangle as well. A configuration with a triangle on top will be classified as a term of category 'pyramid'. The parametric positions of complex rectangles and pyramids are shown in Figures 5.3.c and 5.3.d respectively. These configurations play the role of the traditional syntactic categories of natural language such as noun-phrases (*NP*) or verb-phrases (*VP*). The basic graphical symbols will belong to their corresponding grammatical categories in the same way that nouns and verbs in natural language grammars are classified as *NPs* and *VPs* respectively. Finally, we have to make sure that the grammar does not produce floating triangles or rectangles, or rectangles on top of triangles.

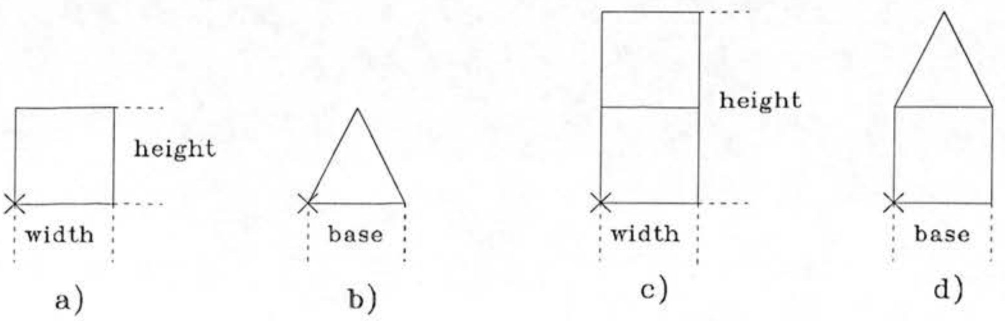

FIGURE 5.3

We also consider other spatial properties that relate a pair of graphical symbols. For instance, the distance between a block and the table is a real number that measures the height of a block in relation to the table. We also consider the distance between two blocks if the one below is a rectangle. We show this relation in Figure 5.4.

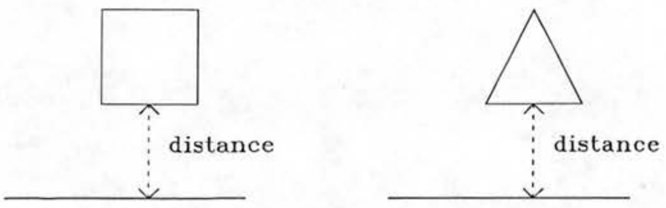

FIGURE 5.4

For the formalisation of the structure of graphics and its integration in the logical representation, the following strategy is adopted: first, an algebraic structure **G** for graphics in *BW* is developed. This structure is crucial for our enterprise because it has both a declarative and an algorithmic interpretation. This duality is at the heart of this theory, and the two interpretations of **G** are fully illustrated. Then, the structure **G** is embedded within the modal first order logical language $\mathbf{L_g}$ in which both linguistic and graphical expressions can be represented. Some examples of expressions that refer to graphical and abstract individuals, properties and relations are illustrated. The algebraic structure allows us to represent the meaning of well-formed graphical representations of discrete states of the graphics interactive session; however, the algebraic system must be extended for capturing the meaning of drawings when they undergo a process of change. For this purpose, modality is introduced in the representational system. Finally, a notion of graphical inference in relation to the blocks world domain is introduced. This notion embodies the strategy that is used for the solution of problems in the representational environment supported in GRAFLOG.

5.1.1. The structure G.

In the formulation of **G** we follow Goguen (Goguen et al 1978). We start with some general definitions. Let S be a set of sorts. An S-sorted signature Σ is a family $\Sigma_{w,\,s}$ of sets, for each $s \in S$ and $w \in S^*$ (where S^* is the set of all finite strings over S, including the empty string e). An operation symbol $F \in \Sigma_{w,\,s}$ is said to have **rank** w, s, **arity** w and **sort** s. If c is a symbol of rank e, s (i.e. where e is the empty string), then c is called a **constant** of sort s.

For example, we might take our set $S = \{integer\}$, and $\Sigma_{w,\,s}$ to be empty except for $\Sigma_{e,\,integer} = \{0\}$ and $\Sigma_{integer,\,integer} = \{succ\}$. That is, *succ* is to be interpreted as a unary operation (i.e. successor) which takes an argument n of sort *integer*, and yields a value $succ(n)$ which is also of sort *integer*. This system produces the set of natural numbers, that is, the carrier S.

Algebraic systems can be pictorically illustrated using the conventions in Figure 5.5. Circles represent carriers of sorts, and arcs represent the operations with their arguments and values.

For instance, the circle labelled *natural numbers* 'contains' all the objects of that sort. There are two inputs to this carrier: the constant *0* and the objects that are produced by the operation *succ*. This operation takes an element in the carrier as its argument and produces another element of the carrier as its output. The tail of the arrow stands for the operation's argument and it starts in the circle representing the carrier of the argument's sort. The head of the arrow stands for the operation's value and points to the carrier of the value sort. Operations with n arguments, where $n > 1$, are depicted by an arrow with n tails, with 'roots' in the circles standing for the carriers of the corresponding sorts.

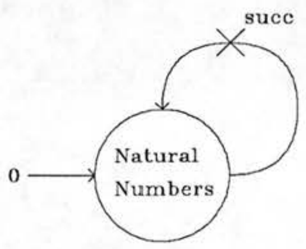

FIGURE 5.5

We now turn to the algebra **G** of graphical terms in *BW*. These terms stand for the graphics themselves, and we think of the terms of **G** as pictures drawn on the screen. In the language, we refer not only to graphical objects, but also to real numbers and ordered pairs of real numbers in order to represent the space itself. We include as well terms for naming the spatial properties and relations of graphical symbols, and for this purpose the sort of truth values *bool* is introduced.

Now, we define the structure **G** as follows,

(1) The set S_G of sorts = {*bool, real, real_pair, line, pyramid, rectangle, drawing*}.

Next, we introduce the individual constants:

(2) $\Sigma_{e,\, bool} = \{0, 1\}$.

(3) $\Sigma_{e,\, real}$ is a (non-countable) infinite set of numerals.

(4) $\Sigma_{e,\, real\_pair}$ is a (non-countable) infinite set of pairs of numerals.

(5) $\Sigma_{e,\, line}$ = {*table*} is the only line of the blocks-world.

(6) $\Sigma_{e,\, pyramid}$ is a (countable) set {*triangle-1, triangle-2,...*} of triangles.

(7) $\Sigma_{e,\, rectangle}$ is a (countable) set {*block-1, block-2,...*} of rectangles.

The operator symbols of graphical sorts are shown from (8) to (11) as follows and they constitute the core of the structure. Operators (12) to (19) are defined in order to make explicit the relation between graphical symbols and their spatial properties and relations.

(8) $\Sigma_{rectangle\ pyramid,\, pyramid}$ = {*wedge*}.

(9) $\Sigma_{rectangle\ rectangle,\ rectangle} = \{stack\}$.

(10) $\Sigma_{line\ pyramid,\ drawing} \cup \Sigma_{line\ rectangle,\ drawing} = \{cons\}$.

(11) $\Sigma_{drawing\ drawing,\ drawing} = \{union\}$.

(12) $\Sigma_{pyramid,\ real\_pair} \cup \Sigma_{rectangle,\ real\_pair} = \{position\}$.

(13) $\Sigma_{pyramid,\ real} = \{base\}$.

(14) $\Sigma_{rectangle,\ real} = \{width,\ height\}$.

(15) $\Sigma_{line\ pyramid,\ real} \cup \Sigma_{line\ rectangle,\ real} \cup \Sigma_{rectangle\ rectangle,\ real} \cup \Sigma_{rectangle\ pyramid,\ real} = \{distance\}$.

(16) $\Sigma_{drawing\ drawing,\ bool} = \{overlap\}$.

(17) $\Sigma_{real\_pair,\ real} = \{x\_coord,\ y\_coord\}$.

(18) $\Sigma_{real\ real,\ real} = \{+\}$.

(19) *Equality*: For every sort s in S_G, $\Sigma_{s\ s,\ bool} = \{=\}$

In Figure 5.6, operations (8) to (11) of **G** are pictorically illustrated for clarity. These operations play a fundamental role in graphical composition.

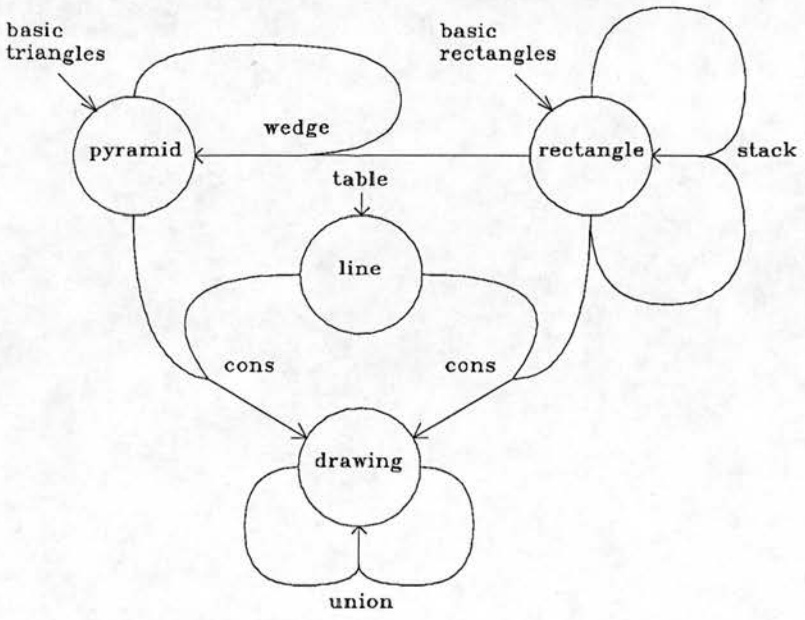

FIGURE 5.6

The diagrams corresponding to operations in (1) to (7) and in (12) to (19) can be drawn along the same lines.

Formation rules are expressed as follows:

(20) Every constant of sort s is a term of sort s.

(21) If $t_1,..., t_n$ are terms of sorts $s_1,..., s_n$, respectively, and f is an operation symbol of rank w, s, where $w = s_1,..., s_n$, then $f(t_1,..., t_n)$ is a term of sort s.

Suppose, for example, that α and β are terms of sorts *rectangle* and *pyramid* respectively. Since *wedge* has been declared to be a constant of arity *rectangle pyramid, pyramid*, we know that *wedge*(α, β) is a term of sort *pyramid.* Some further expressions of this language are,

(22) **stack(block-1, block-2).**

(23) **cons(table, stack(block-1, block-2)).**

The intended interpretation of (22) is a stack made out of two basic rectangles; (23) denotes a configuration in which the object denoted by (22) is supported by the table.

5.1.2. Ill-formed Drawings and the Closure of G.

Operations in standard multi-sorted algebraic systems denote total functions. That means that if the arguments of an operation term are of the appropriate sorts, the value of such an operation is an element of some appropriate sort as well. An operation of **G** that takes as arguments graphical symbols that have appropriate spatial properties, like their position and dimensions, has as a value a graphical object of the operation's sort. However, if the properties of the symbols that are the arguments of an operation have inappropriate values, the operation would not produce an object of its corresponding sort, despite the fact that the arguments themselves are of the right sort. This would be the case, for instance, if the operation *cons(table, right_block)*[1] is evaluated in the drawing in Figure 5.1.b.

One way out to this problem could be to say that operations in **G** are partial functions, and to develop our theory along those lines. However, in this thesis we follow and extend the approach of Goguen (Goguen et al 1978). for the specification of abstract data-types. We define an infinite number of ill-formed elements of every sort in every carrier in S_G. That is, besides the 'normal' elements that belong to the carrier of a sort *s*, there are an infinite number of objects, namely $e_{<s, 0>}$, $e_{<s, 1>}$, ..., $e_{<s, n>}$, which act as the value of operations that otherwise would be undefined. These elements correspond to the 'error element' introduced for the specification of abstract data-types (Goguen et al 1978). In Figure 5.1.b, the operation *cons(table, right_block)* = $e_{<drawing, i>}$, where $e_{<drawing, i>}$ is the *i* error element in the carrier of sort *drawing*. When an error occurs it is propagated to all operations that are related in a complex expression.

We now proceed to modify the specification of the operation terms of Σ so that the error elements of every sort are accounted for in the algebraic system. The modification consists in defining the value of every operation in Σ as follows: if the properties of the arguments of the operation terms have appropriate values, the operation value is a normal element of the

[1] Suppose that the name of the 'flying' block in Figure 5.1.b is *right_block*.

appropriate sort; otherwise, the operation's value is an error element of the operation's sort.

We consider the case for the operation *stack* in detail. Other operations of **G** have to be redefined along the same lines. We define the rank of the operation term *stack* as *bool rectangle rectangle, rectangle.* Note the addition of the first sort in the arity: the first argument of *stack* is a term of sort *bool.* At the syntactic level the operation has the following form

(24) $stack(\phi_{bool}, x_{rectangle}, y_{rectangle})$.

The value of the boolean argument is true if the spatial properties of the rectangles to be combined by *stack* are such that the operation is permitted. According to Figures 5.3.a and 5.3.c, the condition by which ϕ is true is defined in terms of other operator terms of **G** as follows,

```
(25)  IF     y_coord(position(x)) + height(x) = y_coord(position(y)) AND
             x_coord(position(x)) = x_coord(position(y)) AND
             width(x) = width(y)
      THEN
             φ = TRUE.
```

The value of *stack* itself is given by the following table:[2]

(26)

CASE	ϕ	x	y	VALUE
1	*true*	*r*	*r*	*r*
2	*false*	*r*	*r*	e_r
3	e_b	*any*	*any*	e_r

The terms *r* in (26) stand for normal members of sort *rectangle.* The terms e_b and e_r stand for the error element in the boolean sort and for any error element in the rectangle sort respectively. The terms *any* stand for a normal or an error element, since if the boolean error is an argument of the operation term, the error propagates whatever the other arguments are. The rows in the table illustrate the cases that have to be accounted for. Case 1 indicates the situation in which the combination of two rectangles is permitted. Case 2 illustrates the introduction of an error element e_r in the rectangle sort. In this case, the arguments of the operation are of the proper sort, but their geometrical properties block the combination. Case 3 illustrates the situation in which an error has been propagated. This case might occur in the evaluation of complex expressions of **G**. Consider, for instance, that the value of expression (27) relation to Figure 5.1.b[3] is $e_{<drawing, j>}$. This is so because the value of the second argument of *union* is

[2] If there is no ambiguity in the expression, we use the terms *true* and *false* instead of *1* and *0* for clarity.

[3] Suppose that the triangle and rectangles in the left structure of Figure 5.1.b are named *triangle-1* and *rectangle-1* respectively.

$e_{<drawing,\ i>}$.

(27) **union(cons(table, wedge(rectangle-1, triangle-1)), cons(table, right_block)).**

Note that the conditions in terms of which the value of the boolean argument of *stack* is known are defined in terms of other operations of *G*. In fact, we have to define the value of these operations when one or more of their arguments are the error elements of their correspondings sorts. The arity of all operators of **G** has to be redefined to include a boolean argument. With these considerations, the conditions for applying the *stack* operation are as follows,[4]

(28) **IF** y_coord(true, position(x)) + height(true, x) = y_coord(true, position(true, y)) AND
x_coord(true, position(true, x)) = x_coord(true, position(true, y)) AND
width(true, x) = width(true, y)

THEN

ϕ = true

Note that the first argument of every operation term in (28) is itself the boolean constant *true*.

The arity of every operator symbol in Σ of **G** has to be redefined on the same lines, and the conditions under which the boolean flag indicating that the operation is permitted have to be made explicit. However, these definitions are straightforward and they are assumed in the following sections. Furthermore, we adopt the convention that the first symbol in the arity of every operation is *bool*, and we omit this argument in the formulas below, unless it is explicitly required in the context.

5.1.3. Algorithmic interpretation of G.

As has been mentioned, expressions of **G** have a dual interpretation. The graphical terms of sort *line, pyramid* and *rectangle* have a dual substantial realisation in the graphical domain. In GRAFLOG, basic graphical symbols are introduced directly by ostension, and composite structures are produced by a geometrical construction procedure related to each operator in **G**.

Suppose that in the graphical menu of GRAFLOG three kinds of symbols are defined, namely *line, pyramid* and *rectangle*. Each individual symbol of any of these types is represented as a basic constant of the corresponding sort in **G**, and can be described on the screen in terms of a set of parameters that correspond to some of the operator terms in **G**. Of course, basic constants of sort *pyramid* are triangles. Suppose as well that the constant names for the graphical symbols are introduced by ostensive definitions in a graphics interactive session as was shown in Chapter 2. For instance, the expression *This is block-1* is typed at the same time that the shaded rectangle is defined in the graphical context of symbols in dotted lines on the screen, as shown

[4] The boolean conditions for the operators "+" and "=" are omitted for clarity.

in Figure 5.7. The name *block-1* indexes the graphical symbol. This indexing relation is illustrated in Figure 5.7 as well.

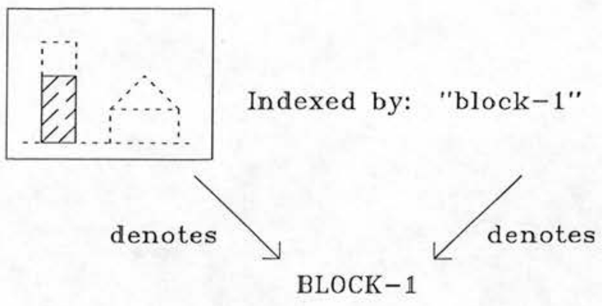

FIGURE 5.7

The intended interpretation of this picture is that the name *block-1* is the index of the shaded rectangle and both the name and the image denote the object *BLOCK-1* in the block world itself.

Now we introduce an algorithmic interpretation for operator symbols in **G**. Every operator symbol $\Sigma_{w,s}$ in **G** has an algorithmic interpretation. This is a typed geometric constructive procedure that given a set of graphical symbols of some appropriate sorts constructs an object of an appropriate sort. In the process of interpreting a *BW* drawing, every algorithm that is related to an operation symbol of **G** builds up the representation of a geometrical object of the operation's sort. This object is then asserted in the geometrical data-base of GRAFLOG.

For the moment we assume that these procedures exist, and they construct a graphical object if the set of graphical symbols to be combined are of the appropriate types and satisfy the predefined geometrical conditions that condition their corresponding operation in **G**.[5]

We can illustrate by means of an example how the operations of **G** work in the geometrical interpretation process. In Figure 5.8, a drawing made of five graphical symbols, with their corresponding indices in **G**, is illustrated.

[5] Computational geometry procedures for producing this kind of drawing are available (Weiler 1980, Pineda 1986).

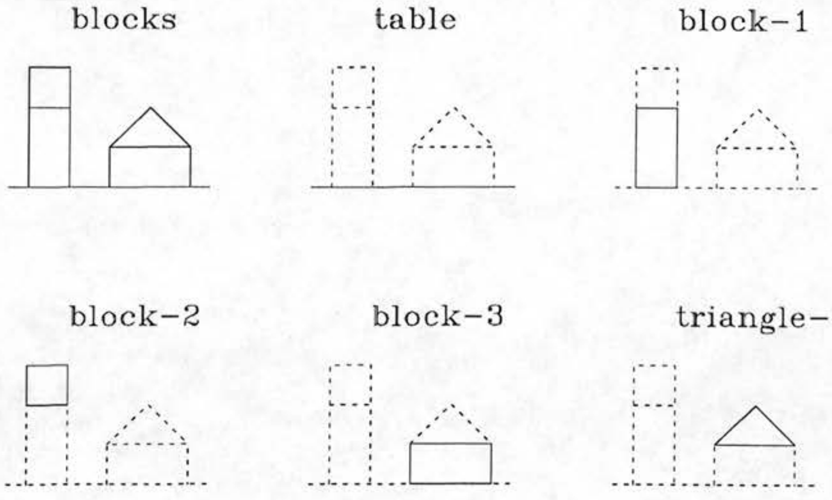

FIGURE 5.8

Note that the name *blocks* is the index of the whole structure, and it stands for the corresponding composite term. The way this identifier is defined is explained below.

Figures 5.9.a to 5.9.e illustrate how operators are applied to terms of appropriate sorts. The symbols drawn with solid lines in the left and middle pictures of each figure are the terms combined by the operation symbol. The graphical term produced by the operator is shown to the right. It is worth emphasising the nature of this construction process: once the basic symbols are created on the screen, the system has no notion of the relationships holding between them; but once the symbols are combined, a new symbol is produced in the representational system, and its geometrical description is asserted in the geometrical data-base. This new symbol is the geometrical union of its constituent parts. The expressions above the Figures 5.9.a to 5.9.e are terms of **G** and they are also 'indices' of the corresponding composite pictures.

(29) **stack(block-1, block-2).**

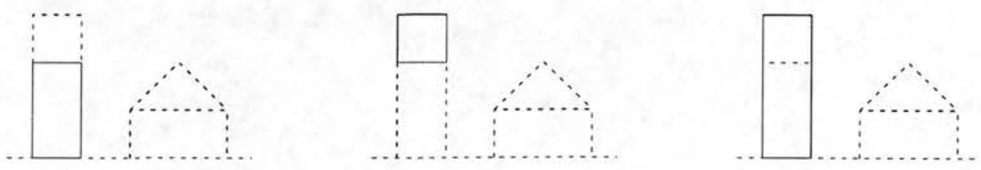

FIGURE 5.9.a

(30) **cons(table, stack(block-1, block-2).**

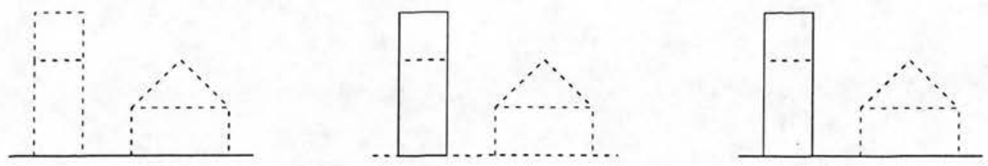

FIGURE 5.9.b

(31) **wedge(block-3, triangle-1).**

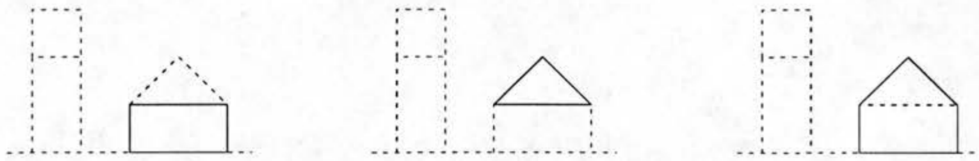

FIGURE 5.9.c

(32) **cons(table, wedge(block-3, triangle-1).**

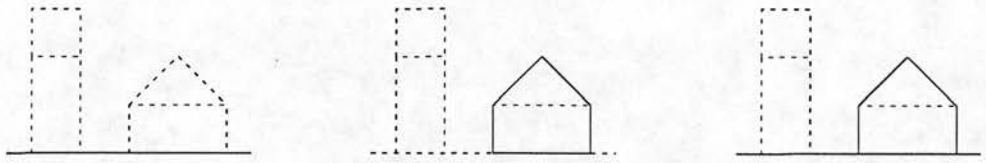

FIGURE 5.9.d

(33) **union(cons(table, stack(block-1, block-2), cons(table, wedge(block-3, triangle-1).**

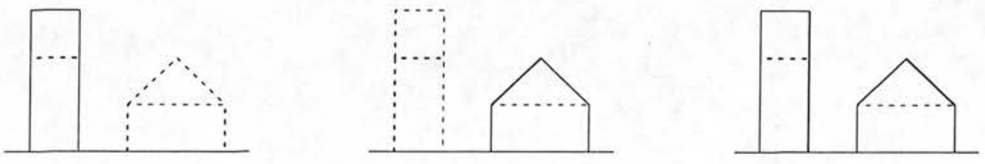

FIGURE 5.9.e

5.1.4. Relation between Algebraic and Algorithmic Interpretation of G.

An interesting question is how we can know which expression of **G** corresponds to a drawing that has been created by a standard graphics and linguistic interaction in GRAFLOG, and that is currently displayed on the screen. The interpretation of a basic symbol is stated directly by ostension and it is asserted in the representational structures of GRAFLOG, as was discussed in Chapter 2. However, the interpretation of complex graphical structures must be established by an inference process that acts upon the graphical input. This process corresponds with the parsing of natural language expressions in computational linguistics --or with the production of the object code of some source program by a compilation process-- and it could be regarded as 'a graphical parser'. The function of a graphical parser is to take a collection of graphical symbols on the screen and to produce a structured representation of the whole drawing. That is, an expression in **G** that corresponds to the drawing on the screen taken as a geometrical and topological unit. For instance, the graphical parser takes the set of basic graphical symbols in the graphical context in Figure 5.8 and produces the complex geometrical structure labelled *blocks* in the same figure. Each one of the basic graphical symbols has an entry in the geometrical data-base, that is indexed by the symbol's name, and whose sort is *line*, *triangle* or rectangle. The object *blocks*, on the other hand, indexed a complex graphical object which is geometrically and topologically defined in terms of the basic symbol in the geometrical data-base. The representation of such a structured graphical object on the screen and its corresponding expression in **G** in Figure 5.9.e are two isomorphic representations of the same object.

On the basis of the structure **G** there is a simple method for producing the interpretation of a graphic of sort *drawing* --a graphical sentence-- in the blocks-world. For the definition of such a method consider that the number of basic graphical symbols introduced by ostension in any drawing, at least in the drawing that we are interested in, is finite. For instance, the drawing in Figure 5.8 is built up from five symbols. Consider as well that there is a countably infinite set $EXP_\Gamma = \{exp_1, exp_2, ..., exp_n\}$ of expressions of sort *drawing* in **G**, where Γ is a finite set of graphical symbols --in the example $\Gamma = \{table, block\text{-}1, block\text{-}2, block\text{-}3, triangle\text{-}1\}$-- such that every symbol in Γ is referred to by every expression exp_j in EXP_Γ.

The interpretation of a complex drawing composed by the set Γ of basic graphical symbols is the simplest expression in the inductive closure of Γ and the operation symbols of **G** that can be constructed by the algorithms named by the operations of **G**. This is a 'syntactic' method, and it could be proved correct for a family of drawings. Note that this procedure terminates only for well-formed *BW* drawings.

We can also define a decision procedure by which graphical structures of sort *drawing* can be distinguished from arbitrary compositions made out of lines, triangles and rectangles that are not well-formed drawings. It can be noticed that in every term of sort *drawing* in **G**, all blocks are above the line representing the table, and all blocks are also supported in an appropriate manner.[6] The decision procedure for **G** consists in an algorithm that tests that all blocks satisfy such conditions. Furthermore, if a set of symbols define an object of sort *drawing*, the expression of **G** that denotes such a graphical object can be found by a search process that takes the geometrical description of the basic symbols as its input. The expression of **G** reflects the geometrical method by which it itself is found. This method is a constructive procedure: the proof that an arbitrary set of symbols constitute a term of sort *drawing* in *BW* is given by its geometrical construction method. Its interpretation in **G** is a term of sort *drawing* that corresponds with the construction method itself.

In the PROLOG implementation of GRAFLOG, if the graphical parsing procedure for **G** takes the basic symbols in Figure 5.8 as its input, it produces expression (34) as its output. In GRAFLOG, the functors *union, cons, wedge* and *stack* implement the corresponding operations of **G**, and *blocks* is an arbitrary identifier provided by the system.

(34) **blocks := union(cons(table, stack(block_1, block_2),**
cons(table, wedge(block_3, triangle_1).

In the graphical domain, the object that is named by (34) is constructed and its representation is asserted in the geometrical data-base of GRAFLOG. For the moment we assume that the graphical parser methods exist. However, *BW* has no practical interest and neither the syntactic parsing nor the decision procedure are spelled out in detail. But the corresponding procedures for the language for design presented in Chapter 6 are fully discussed.

5.2. The Graphical and Logical Language $\mathbf{L}_g$.

For many purposes, it is useful to have a more expressive way of presenting graphical information than that provided by the algebra **G**, and consequently we will extend the signature *S* so that it allows us to define a (many-sorted) modal first order language $\mathbf{L}_g$. In other words, we embed the structure **G** within a modal logical language. This will allow us to represent graphical and logical knowledge in an integrated fashion.

The set of sorts S_g of $\mathbf{L}_g$ is obtained by adding in a new sort: $S_g = S_G \cup \{individuals\}$. The sort of individuals is a very general set and all objects in other sorts are also members of the sort of individuals of the world. Intuitively, there are individuals referred to by means of the language $\mathbf{L}_g$ which are not graphical structures. These individuals are referred to by natural language expressions, and they might be physical or abstract entities.

The modal first order language $\mathbf{L}_g$ is defined as follows:

[6] See Figure 5.3.

(35) All operator symbols of **G** are also in $\mathbf{L}_g$.

(36) $\Sigma_{bool\ bool,\ bool} = \{\wedge, \vee, \rightarrow, \equiv\}$

(37) $\Sigma_{bool,\ bool} = \{\neg, \Box\}$

(38) *Quantifiers*: For every sort $s \in S_g$, $\forall_s$ (for all), $\exists_s$ (there exists), each of rank $\Sigma_{s,\ bool}$

(39) *Variables*: For every sort $s \in S_g$, there is a countably infinite set $\mathbf{V}_s = \{x_{s,0}, x_{s,1},...\}$ of variables such that $x_{s,i} \in \Sigma_{e,s}$.

(40) *Constants*: For every sort $s \in S_g$, there is a countable set $C_s = \{c_{s,0}, c_{s,1},...\}$ of constant symbols such that $c_{s,i} \in \Sigma_{e,s}$.

(41) *Predicates*: For every sort $s \in S_g$, there is a countable set $P_s = \{p_{s,0}, p_{s,1},...\}$ of predicate symbols such that $p_{s,i} \in \Sigma_{u_0 u_1 ... u_i,\ bool}$. The string $u_0 u_1 ... u_i$ is called the **arity** of $p_{s,i}$.

(42) *Auxiliary symbols*: "(" and ")".

Formation rules are expressed as follows:

(43) Every constant of sort s is a term of sort s.

(44) If $t_1,..., t_n$ are terms of sorts $s_1,..., s_n$, respectively, and f is an operation symbol of rank w, s, where $w = s_1,..., s_n$, then $f(t_1,..., t_n)$ is a term of sort s.

(45) If $t_1,..., t_n$ are terms of sorts $s_1,..., s_n$, respectively, and f is a predicate symbol of rank $w, bool$, where $w = s_1,..., s_n$, then $f(t_1,..., t_n)$ is a term of sort *bool*.

(46) If $\forall_s$ is a quantifier of sort s, u is variable of sort s and ϕ is a term of sort *bool* then $\forall_s u\phi$ is a term of sort *bool*.

(47) If $\exists_s$ is a quantifier of sort s, u is variable of sort s and ϕ is a term of sort *bool* then $\exists_s u\phi$ is a term of sort *bool*.

5.2.1. Interpretation of G and $\mathbf{L}_g$.

For the definition of the semantic interpretation of **G** and $\mathbf{L}_g$ we follow Dowty (Dowty et al 1981). A *model M for* $\mathbf{L}_g$ is an ordered tuple $<G, I, F>$, where $G = <G_s>_{s \in S}$ is an S-indexed family of non-empty sets, I is a set $\{i_1, i_2,...\}$ of states and F is an interpretation function whose domain is the set of all non-logical constants of $\mathbf{L}_g$ and whose range is described below. If α is constant of sort s, then $F(\alpha)(i) \in G_s$. In other words, the denotation of α at every state i is a member of G_s.

We define as well an assignment function g that has as its domain the set of all variables and has as a value a member of G_s for each variable of sort s.

Notational convention: the semantic value of any expression α with respect to a model M, an state $i \in I$ and a value assignment g is represented as:

(48) $[[\alpha]]^{M,\ i,\ g}$

Now we define the semantic interpretation of expression of $\mathbf{L}_g$: [7]

(49) If α is a constant of sort s, then $[[\alpha]]^{M,i,g} = [F(\alpha)](i)$.

(50) If α is a variable then $[[\alpha]]^{M,i,g} = g(\alpha)$.

(51) If f is an operation symbol of rank w, s in $\mathbf{G}$, for $w = s_1,\dots, s_n$, then $[[f]]^{M,i,g}$ is a function with domain in $G_{s_1} \times \dots \times G_{s_n}$ and range in G_s. For every operator symbol in $\mathbf{D}$, this function is defined as shown in Sections 5.2.2 and 5.2.3.

(52) If p is a predicate symbol of rank w, s, for $w = s_1,\dots, s_n$, then $[[p]]^{M,i,g} = [F(p)](i)$ such that

$$[F(p)](i) \subseteq G_{s_1} \times \dots \times G_{s_n}$$

(53) If $t_1,\dots, t_n$ are terms of sorts $s_1,\dots, s_n$, respectively, and f is an operation symbol of rank w, s, where $w = s_1,\dots, s_n$, then $[[f]]^{M,i,g}([[t_1]]^{M,i,g},\dots, [[t_n]]^{M,i,g})$ is the result of applying the function $[[f]]^{M,i,g}$ to its arguments $[[t_1]]^{M,i,g},\dots, [[t_n]]^{M,i,g}$.

(54) If α and β are terms of sort s then $[[\alpha = \beta]]^{M,i,g}$ is 1 if and only if $[[\alpha]]^{M,i,g}$ is the same as $[[\beta]]^{M,i,g}$.

(55) If ϕ and ψ are terms of sort *bool* then $[[\neg\phi]]^{M,i,g}$ is 1 if and only if $[[\phi]]^{M,i,g}$ is 0, and $[[\neg\phi]]^{M,i,g}$ is 0 otherwise.

(56) If ϕ is a term of sort *bool* then $[[\phi \wedge \psi]]^{M,i,g}$ is 1 if and only if $[[\phi]]^{M,i,g}$ is 1 and $[[\psi]]^{M,i,g}$ is 1. (The definition for other truth-functional connectives is given in the standard way along the lines shown for the connective $\wedge$).

(57) If ϕ is a term of sort *bool* and u is a variable of sort s then $[[\forall_s u\phi]]^{M,i,g}$ is 1 if and only if $[[\phi]]^{M,i,g'}$ is 1 for all g' exactly like g except possibly for the value assigned to u.

(58) If ϕ is a term of sort *bool* and u is a variable of sort s then $[[\exists_s u\phi]]^{M,i,g}$ is 1 if and only if $[[\phi]]^{M,i,g'}$ is 1 for some value assignment g' exactly like g except possibly for the value assigned to u.

(59) The interpretation of the modal operator $\Box$ is such that if ϕ is an expression of sort *bool*, then $[[\Box\phi]]^{M,i,g}$ is 1 if and only if $[[\phi]]^{M,i,g}$ is 1 for all i' in I.

Definition of truth:

(60) If ϕ is a term of sort *bool*, then ϕ is *true with respect to M and to I* if and only if $[[\phi]]^{M,i,g}$ is 1 for all value assignments g.

[7] In the semantic definition of equality and the logical connectives, we use the standard notation instead of the cumbersome prefix form that is formally required in $\mathbf{L}_g$.

5.2.2. Representation of Graphical and Linguistic Knowledge.

With $\mathbf{L_g}$ we can represent knowledge about the blocks world. The following examples are *wff* of $\mathbf{L_g}$:[8]

(61) $\exists \mathbf{x}_{\mathbf{drawing}}, \exists \mathbf{y}_{\mathbf{line}}[\, \mathbf{x = cons(y, stack(block\text{-}1, block\text{-}2))}].$

(62) $\exists \mathbf{x}_{\mathbf{pyramid}}, \exists \mathbf{y}_{\mathbf{rectangle}}\mathbf{y}, \exists \mathbf{z}_{\mathbf{pyramid}}[\, \mathbf{x = wedge(y, z)}]$

Formula (61) will be true with respect to *M*, *I* and *g* iff $[[g(y)]]^{M,\, i,\, g} = TABLE$, and $[[g(x)]]^{M,\, i,\, g}$ is is the well-formed blocks-world drawing shown in Figure 5.9.b

Formula (62) will be true with respect to *M*, *I* and *g* if $[[g(y)]]^{M,\, i,\, g} = BLOCK\text{-}3$, $[[g(z)]]^{M,\, i,\, g} = TRIANGLE\text{-}1$ and $[[g(x)]]^{M,\, i,\, g}$ is the drawing in Figure 5.9.c.

The language $\mathbf{L_g}$ is useful for integrating graphical and linguistic knowledge. We define *supports* as:[9]

(63) $\Box \forall \mathbf{x, y, z[x = stack(y, z) \vee x = wedge(y, z) \vee x = cons(y, z)}$
$\mathbf{\rightarrow x = supports(y, z)].}$

Note that formula (63) holds true at every state. Suppose that *John* is a constant term, *beautiful* a *1*-place predicate, and *sees* is a *2*-place predicate of $\mathbf{L_g}$. Then, we can express, for instance,

(64) $\exists \mathbf{x\, [sees(john, x) \wedge x = supports(table, block\text{-}1)]}$

(65) $\exists \mathbf{x, y[\, beautiful(x) \wedge y = supports(x, block\text{-}1)]}$

Expression (64) asserts that John sees something and the thing that John is seeing is the structure of the table supporting block-1. In GRAFLOG, linguistic knowledge is asserted in the natural language knowledge-base *NLKB* and the graphical knowledge is asserted in the geometrical data-base *g_db* as explained in Chapter 2, in Figure 2.7. Formula (65) will be true with respect to *M*, *I* and *g* if $[[g(x)]]^{M,\, i,\, g} = TABLE$, the fact *beautiful(table)* is asserted in *NLKB*, and *supports(table, block-1)* can be produced from the geometrical interpretation process and rule (63).

Another interesting formula might be

(66) $\Box \forall \mathbf{x\, clear(x) \equiv \neg \exists\, y, z\, [y = supports(z, x)].}$

In some representations of the blocks-world, the predicate *clear* is taken as basic, but as is shown, this word has an underlying layer of geometrical interpretation.

[8] We take the conventional decision of using the standard notation for logical formulas instead of the prefix form of the syntactic definition of $\mathrm{L_g}$.

[9] In the formulas presented below, we assume that all quantifiers are of the appropriate sorts. However, for some complex formulas the formal notation is given for clarity.

5.2.3. Modality in $\mathbf{L_g}$.

Now, we come to the issue of the set of states introduced in the algebraic interpretation of $\mathbf{L_g}$ and the interpretation of the modal operator $\Box$. Modality is required for dealing with changes in graphical representations. If the properties of the graphical symbols were given once and for all at some state, a standard multi-sorted logical system would be enough for referring to images of this blocks-world; however, we would like to be able to model the change of one or more properties of an image that is referred to by a term of $\mathbf{L_g}$.

Now, suppose that *block-2* is selected as shown in Figure 5.10.a and its position is changed as shown in Figure 5.10.b by a direct manipulation process in standard graphics interaction.

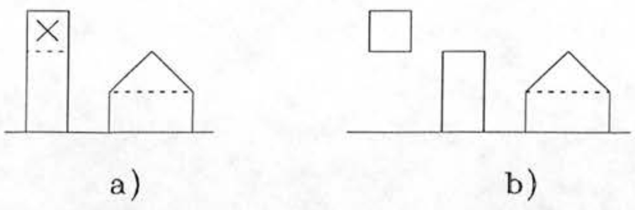

FIGURE 5.10

We assume that the relation between an individual constant of $\mathbf{L_g}$ and the image that it indexes remains the same in this kind of change. For instance, the constant *block-2* indexes, or is the name of, the block selected in Figure 5.10.a and also of the 'flying' block of Figure 5.10.b. We take basic graphical symbols to be rigid designators (Kripke 1980). That is, they denote the same individual in every possible world. As a consequence, the image that is indexed by a constant term will be a rigid designator as well.

The philosophical view that names in natural language are rigid designators is relatively recent, and is the subject of considerable philosophical debate (Kripke 1980). One rival view holds that individuals are identified by some set of 'essential' properties that they have, and once such properties are realised by some knowing individual, he is able to identify the thing exhibiting such a set of properties. According to this second view and in the version supported by Russell, an ordinary name, like *Luis*, is just an abbreviation of the description of the properties by which the individual Luis can be identified in the world (Russell 1905).

According to the view advocated by Kripke, however, things are named independently of all the properties that the named thing might have in any possible situation, and the reference of a name is known in terms of the knowledge that is shared by the linguistic community, and not by the knowledge of the properties that the thing referred to by a name might have. For our purpose, Kripke's view is more appropriate. At the conceptual level, it just means that once a basic graphical symbol has been created on the screen, it preserves its identity regardless the transformations that it might undergo in the course of an interactive session. I believe that this intuition can hardly be disputed in relation to computer graphics. At the implementational level, the view that graphical symbols are rigid designators means that a linguistic name indexes the same graphical symbol along the graphics interaction. The relation between the linguistic and graphical symbols is established by a permanent link in the computer's memory.

Suppose that Figures 5.10.a and 5.10.b are two 'possible worlds' in which the same individuals are present. Suppose as well that the modal index of the state in Figure 5.10.a is i_0, and the index of the state in Figure 5.10.b is i_1.[10] The interpretation function F of the model M for $\mathbf{L_g}$ assigns the same denotation to individual constants in every state as a consequence of taking names of $\mathbf{L_g}$ as rigid designators. For instance,

(67) $[[\mathit{block\text{-}2}]]^{M,\, i_0,\, g} = \mathit{BLOCK\text{-}2}.$
$[[\mathit{block\text{-}2}]]^{M,\, i_1,\, g} = \mathit{BLOCK\text{-}2}.$

The term *block-2* is a name of $\mathbf{L_g}$, and it can be thought of as the graphical symbol itself; on the other hand, *BLOCK-2* stands for a set-theoretical object in the model, a member of $G_{rectangle}$.[11] Now, consider the question: what is the position of *block-2*? Clearly this property of *block_2* is different in the two figures despite the fact that we are asking for the same property of the same object. Assume that the position of *block-2* is p_0 in Figure 5.10.a, and p_1 in Figure 5.10.b. The interpretation function F makes the following assignments:

(68) $[[\mathit{position(block\text{-}2)}]]^{M,\, i_0,\, g} = p_0.$
$[[\mathit{position(block\text{-}2)}]]^{M,\, i_1,\, g} = p_1.$

Index transitions can be achieved by an interactive manipulation as shown in Figure 5.10. Adding, deleting or changing a graphical symbol on the screen changes the state. However, such a change is meta-theoretical in the sense that it is produced by the user, and the transition itself does not have to be modelled within the formal system. The language $\mathbf{L_g}$ might produce the complex graphical object in the state before and after the transition, but the transition itself is not known to the system. The rules for producing such a transition are kept in the mind of the human-user or elsewhere. However, graphical transformations can be modelled if a transformation function for mapping graphical states to graphical states is defined. Here, we define the transformation function *change* that 'implements' such a kind of transition.

The purpose of *change* is to alter one proposition concerning an individual that undergoes a process of change. Let ϕ be a proposition about an individual α that is true in a state $i \in I$, and let ψ be a proposition about the same individual that is possibly true in some other state i', but is false in the current state i. The purpose of the function *change* is to produce a transformation

[10] Note that the composition in Figure 5.9 is defined at i_0.

[11] See Figure 5.7.

by which a new state i' is reached in which ψ becomes true of α and ϕ becomes only possibly true of α. In fact, ϕ becomes false of α in i'. The semantics of *change* is defined in (69) as follows:

(69) $[[change(\alpha, \phi, \psi)]]^{M,i,g} = 1$ iff

- *(1)* $[[\phi]]^{M,i,g} = 1$ and $[[\psi]]^{M,i,g} = 0$ and
- *(2)* $i \rightarrow i'$ and
- *(3)* $[[\phi]]^{M,i',g} = 0$ and $[[\psi]]^{M,i',g} = 1$ and
- *(4)* $[[\alpha]]^{M,i,g} = [[\alpha]]^{M,i',g}$.

Clause (69.4) is required because we do not want to constrain *change* to the modification of basic graphical symbols, so α must denote the same individual before and after a change has taken place. Through this function we can model the change from Figure 5.10.a to 5.10.b, and

(70) $[[change(block\text{-}2, position(block\text{-}2) = p_0, position(block\text{-}2) = p_1)]]^{M,i_0,g} = 1.$

Rule (70) defines a change of one property of a basic graphical symbol. However, if the symbol that is changed is a part of a complex graphical configuration there might be global consequences of this kind of local transformations. For instance, Figure 5.10.a is a normal element of sort *drawing*, but Figure 5.10.b denotes an error element of such a sort. This implies that the modification of a property of a graphical object might lead to certain instability in our representational system. We address this issue in the description of GRAFLOG's method for the solution of problems.

5.3. A Graphical Inference Process.

In this section we show how the representational system that has been developed in this chapter is used for modelling the process of change when the system is engaged in the solution of problems.

First, we follow the consequences of the modification of the drawing from Figure 5.10.a to 5.10.b by means of the transformation function *change* in (70). Consider that the whole graphical configuration in Figure 5.10.a, *blocks*, is a term of sort *drawing*. Given that *block-2* is a basic constituent of *blocks* then changing a property of *block-2* has the *a fortiori* consequence of changing a property of *blocks* too. The question that is interesting for us is whether the complex graphical object of sort *drawing* is the same before and after a change of one of its constituent symbols has taken place. We adopt the convention that it is in fact the same, and if it looks different this is just because one of its contingent properties has been changed. However, there is one limitation that has to be considered in adopting this identity convention: if in state i_0, *blocks* is a normal element of sort *drawing*, but the graphical configuration in state i_1 is an error element of the same sort, namely $e_{<drawing, j>}$, then these two objects are necessarily different. We consider state i_1 one in which an instability occurs, and we have to take an action in order to find another configuration in which *blocks* is in fact a normal element of sort *drawing*, and can be considered the same object as the one in the original state i_0, for the purpose of indentifying graphical objects that undergo a process of change.

We can state the same in more simple terms: after a change has occurred, we require that no error element of any graphical sort is introduced in the representation. When this is in fact the case, the representational system reaches a state of *equilibrium*.

Let β be a complex graphical object that is made out of a set Γ of basic graphical objects, and let $\alpha \in \Gamma$. We say that β of sort s 'depends' on α, and we express this dependency relation as $\beta_{<s, \alpha>}$. Now, we formulate the following change condition:

(71) **IF** $[[change(\alpha, \phi, \psi)]]^{M, i, g}$ **THEN** $[[change(\beta_{<s, \alpha>}, \phi, \psi)]]^{M, i, g}$

If the object $\beta_{<s, \alpha>}$ in the state i' is not an error element of sort s, then the change is meaningful and equilibrium is achieved. This is specified as follows:

(72) **IF** $[[\beta_{<s, \alpha>>}]]^{M, i', g} \neq e_{<s, j>}$ **THEN** $[[change(\beta_{<s, \alpha>}, \phi, \psi)]]^{M, i, g} = 1.$

However, if an error occurs, then a new change must be brought about. This change is defined as follows:

(73) **IF** $[[\beta_{<s, \alpha>}]]^{M, i', g} = e_{<s, j>}$
THEN $[[change(\beta_{<s, \alpha>}, \phi, \psi)]]^{M, i, g} = [[change(\alpha, \psi, \rho)]]^{M, i', g}$

Expression (73) defines a recursive process of change. In case the complex object $\beta_{<s, \alpha>}$ is an error of sort s, then the value of the original transition $i \rightarrow i'$ is the same as the value of a new transition from i' to another state i''. The new change is defined, however, not in relation to the complex drawing but in relation to the basic symbol that was originally changed. This object is α. Presumably, the property ψ that became true of α in the transition $i \rightarrow i'$ has the undesirable consequence of leaving the complex object β, of which α is a part, in error; so we take this property as the one that has to be changed from i' to i'' in a state of equilibrium. We define a new change by which a new property, namely ρ in (73), will become true of α, and hopefully, of $\beta_{<s, \alpha>}$ as well. If it is determined that there is no acceptable ρ, then the whole process of change ends in a non-equilibrated state. This condition is expressed as follows:

(73) $[[change(\alpha, \psi, \rho)]]^{M, i', g} = 0$

When the system reaches this condition, it would have to ask for help; for instance, by notifying the user of the current instability and asking for further directions.

In relation to our example, the change is defined as follows:

(74) **IF** $[[change(block\text{-}2, position(block\text{-}2) = p_0, position(block\text{-}2) = p_1]]^{M, i_0, g}$
THEN $[[change(blocks, position(block\text{-}2) = p_0, position(block\text{-}2) = p_1]]^{M, i_0, g}$

After the transition, *blocks* is an error of sort *drawing* in i_1, then:

(75) $[[blocks]]^{M, i_1, g} = e_{<drawing, j>}$

and a recursive process of change is defined as:

(76) $$[[change(blocks, position(block\text{-}2) = p_0, position(block\text{-}2) = p_1]]^{M, i_0, g}$$
$$= [[change(block\text{-}2, position(block\text{-}2) = p_1, \rho]]^{M, i_1, g}$$

Now, we come the issue of what is an acceptable ρ. As far as the representation in the example stands at the moment, there is no way to tell which property ρ should be brought about to reach equilibrium. For the definition of ρ in the state in Figure 5.10.b, any stable object could be considered; for instance, the structure in Figure 5.10.a, some structure with a different set of basic graphical symbols, or even an empty structure might be acceptable. As this point, there is no restriction on the sort of change that we would like the system to produce in order to achieve a state of equilibrium.

We take the view that that there is no way to think of an inference about the process of change without an explicit notion of 'intention'. By intention we mean 'the sense of direction' in which the representation must be updated in order to reach an equilibrium. The intention is the pulling force that causes the transformation. For this purpose, we could define any arbitrary goal, and according to its definition, a set of arbitrary equilibrium states would be determined. However, for the definition of such a goal we have to account for the kind of relation holding between the representational system and the rest if the world, because the system is part of the world as well. If *block-2* is selected by a human-user and it is moved from Figure 5.10.a to 5.10.b, it would not make much sense to achieve equilibrium by reversing that change. If that block is selected it is because there is the intention of changing its position, in a certain direction. For the definition of the goal of this particular process of change we have to take into account the partial specification of the end-goal that is implicitly defined in the graphical input event: the *stimulus*. However, this information does not determine a unique goal to be reached. For completing this information, we have to turn to the current 'beliefs' of the system. The system's expectation of how things in the world 'tend' to change. And that knowledge must be either in the representational system itself or embedded in the interpretation process; otherwise it would not be known. We do not have to make any commitment on the generality of these principles, or whether they are linguistically known or produced by some procedure; the only commitment here is that such a driving force must exist. It is also required that some reference for the transformation is considered. The most 'permanent' beliefs in the representational system are 'invariant' in some contexts and they cannot be changed. These beliefs set the 'polarity' that directs the process of change. Objects with contingent properties, on the other hand, are most likely to be modified. Of course, there is no absolute reference for all transformations, but the change of the most solid beliefs has to be prompted by a pulling force, by an intention, of great magnitude.

Let us continue with our example. Suppose that the most basic reference for a change, the most permanent belief about the blocks-world, is that the table is horizontal in its current position: it can never be modified. Suppose as well that we have very strong expectations that every entity in the blocks-world must be supported. And we believe in some sort of 'gravitational force' making unsupported blocks fall down. Suppose as well that the rule *if a block is unsupported it must fall down* is known by the system. In the blocks world, it is true that in every state of equilibrium, every block and pyramid are supported. This can be asserted in $\mathbf{L}_\mathbf{g}$ as a blocks-world axiom *BWA* in (77),

(77) $\forall \mathbf{x}_{\mathbf{drawing}}, \forall \mathbf{z}_{\mathbf{pyramid} \vee \mathbf{rectangle}}, \exists \mathbf{y}_{\mathbf{line} \vee \mathbf{rectangle}} [\mathbf{x = supports(y, z)}]$

Now, consider that in a process of change, the interpreter knows that there was an original equilibrium state. It also knows that a property of an individual was changed; for instance, when the drawing was modified from Figure 5.10.a to 5.10.b. The interpreter knows as well which is the object that was modified, and whose new position is the cause of the current instability. Let us say that such an individual is *free* in the current process of change. Finally, the interpreter knows that the table is an 'invariant' in the blocks world. With all of this information, a rule that guides the interpreter in the process of change can be defined. For this rule we take into account the individual who is free and an individual who is invariant in the same process. The change itself is defined in relation to some goal; for instance, axiom *BWA*. The form of this rule is

(78) **IF not** *BWA* and $p_n = distance(table, \alpha)$
THEN $\rho = [position(\alpha) = p_n]$

Once the property ρ is identified, the process of change in (76) can be continued:

(79) $[[change(blocks, position(block\text{-}2) = p_0, position(block\text{-}2) = p_1]]^{M, i_0, g}$
$= [[change(block\text{-}2, position(block\text{-}2) = p_1, \text{position}(block\text{-}2) = p_n]]^{M, i_1, g}$

According to definitions (69) and (72),

(80) $[[change(block\text{-}2, position(block\text{-}2) = p_1, \text{position}(block\text{-}2) = p_n]]^{M, i_1, g} = 1$

As a consequence, the transition from Figure 5.11.a to 5.11.b is achieved, and the system reaches a new state of equilibrium.

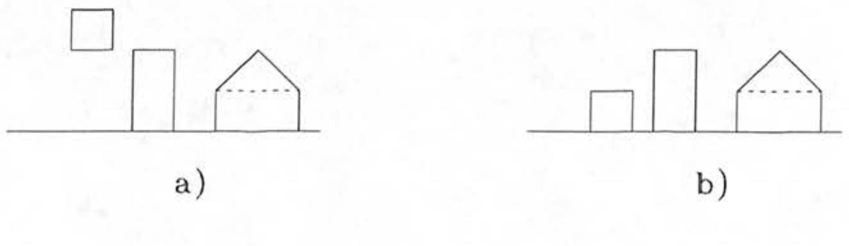

FIGURE 5.11

The interpreter must follow the consequences of applying a change rule until equilibrium is restored in the system. When every unsupported symbol has fallen down, and equilibrium is achieved, the interpretation in **D** of *blocks* in the new state must be found. This completes the inference cycle, and it involves change in both graphical and linguistic representations. The important notion is that such a change must be pulled by a coherent force that works within the system and is independent of the world.

We can summarise the process of change along the following lines. Consider the set of states involved in a process of change. Every state *i* in *I* is represented by a circle in Figure 5.12.

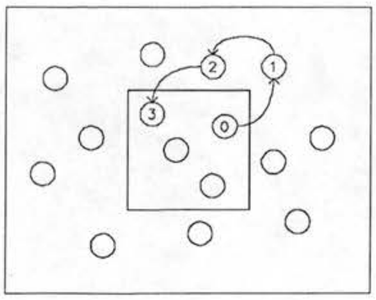

FIGURE 5.12

The outermost rectangle represents the universe of states *I*, and the inner rectangle contains the states in which there is a well-formed blocks drawing --terms of sort *drawing*. These states correspond to the graphical interactive states in which a well-formed drawing is displayed on the screen. If a term of sort *drawing* is defined through a graphics interactive session it will have an interpretation in some state in the inner rectangle, such as i_0. When there is a stimulus and a change of the representation occurs, a new state i_1 is reached as shown by the arrow. At the global level, the inference process depends on the general axioms that must hold for all compositions of pictures of sort *drawing*, and of the individuals that are invariant for such a process of change, such as the table. At the local level, transitions from state to state are achieved by modifying individuals that are free in the current transformation, for instance, unsupported blocks. Finally, when any state within the inner rectangle is reached again, the interpretation in **G** of the new drawing is computed by the graphical parsing procedure. Note that in both original and end states the interpretation of a drawing is local to the current state, and depends exclusively on the geometrical construction procedure.

We call this process of change a graphical inference because the 'bridge' proposition ρ in (78) can be known in virtue of the permanence of the geometrical and topological knowledge. If the algorithms that are named by the operations of **G** were not known, rule (78) could not be defined. Geometrical and topological knowledge hold *apriori* in every possible world.

Here, we conclude this chapter on the representational system and its associated inferencing mechanisms in GRAFLOG. The theory that has been presented is summarised as follows: a family of drawings composed by a set of basic graphical symbols can be described as a multi-sorted algebraic system. Basic graphical symbols are indexed by individual constants of the representational language, and complex graphical objects are referred to by means of terms in the inductive closure of the set of basic symbols of a drawing and the set of graphical operations of the language. Every complex graphical term of the language refers to a complex graphical structure that is build up by means of geometrical and topological constructive procedures. The algebra can be embedded in a modal first order logical language, in which graphical and linguistic knowledge can be represented in an integrated fashion. The logical component of the representational system requires a deductive process for making inferences; however, the graphical component requires a geometric constructive procedure, from which the objects in the representation are produced. This second form of inference can be traced back in history to ancient times, and it is characteristic of Euclidian geometry. Its relation to modern computational geometry and computer graphics has been highlighted by Preparata and Shamos (Preparata et al 1985).

The notion of constructive procedure is also held as fundamental in intuitionistic logic and mathematics. In this view, to grasp the meaning of a mathematical statement is to know what mental construction would constitute a proof of it, and the statement is true iff there is such a proof (Flew 1979).

The relevance of the constructive procedures can be appreciated in relation to the process of change. Logical inferences are meaningful in states of equilibrium, but the instabilities of the intermediate states of a process of change become meaningful because geometrical and topological knowledge is available at every state, regardless of the equilibrium condition.

In our view, to understand an image is to build up its representation by means of some appropriate constructive procedure. The view that an important role of 'imagery' is to provide a non-proof-procedural method for inference, and that such an analysis offers an approach for solving the so-called *frame problem* of AI and Cognitive Science, has recently been advanced (Lindsay 1988). In our representational system both forms of proof procedures are required.

In Chapter 6, a very simple language designed for representing design concepts and intentions, and also for expressing a function for design intention satisfaction in the two-dimensional wire-frame diagrams design domain is presented. The notion of graphical inference introduced in this chapter is further developed in the model of design inference presented in Chapters 7 and 8.

Chapter 6

A Graphical and Logical Language for a simple Design Domain

In this chapter a logical and graphical language for representing CAD knowledge in a simple design domain is presented. The language is useful for designing 2-dimensional wire-frame diagrams that are common in architectural and other kinds of drawings. The language allows the definition and interpretation of graphical symbols that are explicitly drawn on the screen, as well as the representation of other context-dependent space partitions that receive an interpretation in terms of the graphical context in which they emerge. For instance, the walls of an architectural drawing are explicitly drawn, but the rooms that those walls determine emerge from the graphical context. Nevertheless, both kinds of symbols receive an interpretation and are usually named by a similar linguistic device, namely, common nouns. For the identification of these complex graphical objects we introduce the notion of intension of a graphical symbol. Additionally, the language supports the definition and representation of construction lines. This facility is particularly useful for modelling causal relations between possible design states in the definition and production of design inferences.

The algebraic specification of this language corresponds with a formal specification of graphical objects as abstract data types. The algebraic specification of geometrical data types for CAD systems has been explored with promising results; for instance, in the so-called definitive programming framework (Beynon et al 1988, 1989). In this chapter we show how the definition of a language for geometric reasoning for CAD with a sound theoretical foundation can be implemented in a graphics interactive environment. The resulting system allows the representation and interpretation of complex object and functions, as well as providing a clear semantic interpretation.

In Section 6.1, a description of the language along the lines of the framework developed in Chapter 5 is presented. In Section 6.2 we discuss the notions of extension and intension of graphical representations, and we define a criterion by which drawings are identified as representing the same object in the course of a process of change. In Section 6.3, an interaction with the system in a simple design domain is illustrated. Here, some details of the implementation of GRAFLOG in PROLOG are shown. The way basic and emergent graphical symbols are represented and referred to by the language is highlighted in this section too. In Section 6.4, a procedure for inferring the interpretation of drawings in our simple design domain --the graphical parsing procedure-- is presented. For this purpose, a set of identification rules acting upon the graphical and linguistic input is defined. Finally, in Section 6.5, the definition of some design constraints and modelling rules in the course of graphics and linguistic dialogue is illustrated, and the way that such constraints are satisfied by the application of design rules is shown.

6.1. The Language $\mathbf{L_d}$ for Design.

In this section the definition of the language $\mathbf{L_g}$ in Chapter 5 is followed closely. First, the graphical structure **D** for drawings made out of dots, lines and polygons is defined, and its procedural interpretation in terms of a set of geometrical algorithms is illustrated in detail. Then, the structure **D** is embedded within the modal first order language $\mathbf{L_d}$. Finally, the algebraic interpretation for $\mathbf{L_d}$ is presented.

We now turn to the algebra **D**.

(1) The set S_D of sorts = {*bool, real, real_pair, dot, line, polygon*}.

Along the lines shown in Chapter 5, we include an infinite number of error elements in the carrier of every graphical sort *s*, namely $e_{<s,0>}, e_{<s,1>}, ... e_{<s,n>}$.

The individual constants:

(2) $\Sigma_{e,\,bool}$ is the set *{0, 1}*.

(3) $\Sigma_{e,\,real}$ is a (non-countably) infinite set of numerals.

(4) $\Sigma_{e,\,dot}$ is a (non-countably) infinite set $\{d_1, d_2, ...\}$ of dots.

(5) $\Sigma_{e,\,line}$ is a (non-countably) infinite set $\{l_1, l_2, ...\}$ of lines (construed as vectors).

(6) $\Sigma_{e,\,polygon}$ is a (non-countably) infinite set $\{p_1, p_2, ...\}$ of polygons.

The operator symbols are as follows:

(7) $\Sigma_{bool\ dot,\ real\_pair}$ = {*position_of*}.

(8) $\Sigma_{bool\ line,\ real}$ = {*length_of*}.

(9) $\Sigma_{bool\ polygon,\ real}$ = {*area_of*}.

(10) $\Sigma_{bool\ line\ line,\ real}$ = {*angle_between*}.

(11) $\Sigma_{bool\ line,\ dot}$ = {*end_of, origin_of*}.

(12) $\Sigma_{bool\ line\ line,\ dot}$ = {*int_oo, int_om, int_oe, int_mo, int_mm, int_me, int_eo, int_em, int_ee, int_ww, cross_at, t_joins_at, e_join_at, joint_at, intersect_at*}.

(13) $\Sigma_{bool\ polygon\ polygon,\ polygon} = \{union\_of,\ intersection\_of,\ difference\_between\}$.

Note that according to clause (11), there are operation symbols that associate with every line *l* values of sort *dot* which indicate the origin and end point of *l*. That is, *l* is oriented, and is in fact encoded as an ordered pair of dots. Other terms of sort *dot* are produced from the intersection of two terms of sort *line*. Clause (12) defines a set of operator intersections. This class covers the vector relations traditionally considered in computer graphics, and also a set of intersection modes used for the definition of construction lines. For instance, a term of the form $int\_mm(\phi, \alpha, \beta)$ denotes the dot in which the vectors α and β intersect each other. The interpretation of these operator terms is fully explained below.

There are some operation symbols taking boolean values for testing geometrical conditions, i.e. geometrical predicates:

(14) $\Sigma_{bool\ line,\ bool} = \{horizontal,\ vertical\}$.

(15) $\Sigma_{bool\ line\ line,\ bool} = \{perpendicular,\ parallel,\ collineal\}$.

(16) $\Sigma_{bool\ dot\ polygon,\ bool} \cup \Sigma_{bool\ polygon\ polygon,\ bool} = \{in\}$.

(17) $\Sigma_{bool\ dot\ line,\ bool} = \{on\}$.

6.1.1. Algorithmic Interpretation of D.

For every operation symbol of **D** there is a geometrical algorithm that is named by that symbol. In general, the geometrical information represented in **D** is computed by standard geometrical analysis. The sorts of **D** have been selected precisely because they are relevant for computing the geometrical algorithms involved in the operations. It can also be observed that the geometric predicates, i.e. the operation symbols of sort *bool*, assert relations between graphical entities and some primitive geometric property that can be derived from geometrical procedures. In order to illustrate more fully the procedural aspect of non-boolean valued operations, we shall now consider those with rank *bool line line, dot*. Consider first the diagram in Figure 6.1.

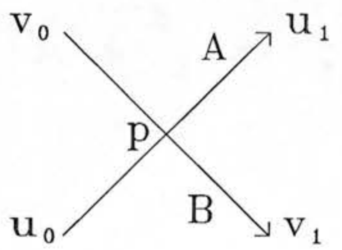

FIGURE 6.1

The vector **A** is defined by the position vectors $\mathbf{u_0}$ and $\mathbf{u_1}$, and the vectorial equation $\mathbf{A} = \mathbf{u_1} - \mathbf{u_0}$ holds. We have a similar equation for vector **B**. The position vector **p** of the intersection point between two arbitrarily defined vectors $\mathbf{u_1} - \mathbf{u_0}$ and $\mathbf{v_1} - \mathbf{v_0}$ in the space is defined in terms of the following equations, where t_1 and t_2 are scalar parameters:

(18)
$$p = \mathbf{u_0} + t_1 * (\mathbf{u_1} - \mathbf{u_0}).$$
$$p = \mathbf{v_0} + t_2 * (\mathbf{v_1} - \mathbf{v_0}).$$

Following Leon Lastra (Leon et al 1986). we classify the scalar parameter values t_1 and t_2 into five cases: $t_i < 0$, $t_i = 0$, $0 < t_i < 1$, $t_i = 1$ and $t_i > 1$. Within this classification there are 25 cases of parameter pairs, as shown on Figure 6.2. Every intersection case is computed by an operation of the form *int_xy*(ϕ, α, β), where α and β are two arbitrary vectors in the space, and ϕ is a boolean condition. The name of the operator for every 'overt' intersection case is of the form *int_xy* where *x* and *y* are symbols in the set *{o, m, e}*. These symbols stand for *origin, middle* and *end* respectively. The interpretation of these operators names is as follows: if *x* is *o*, the intersection point *p* between α and β occurs in the origin of α; if *x* is *m*, the intersection point *p* occurs in the middle of α; and if this symbol is *e* then *p* occurs at the end of α. The code symbol *y* relates the intersection point *p* with the vector β in a similar manner. The interpretation of operator *int_ww* covers the 16 cases in the outer-ring in Figure 6.2. This set of cases is considered in the language for the definition of construction lines. In Figure 6.2, the α vector is horizontally oriented, and the β vector is vertically oriented.

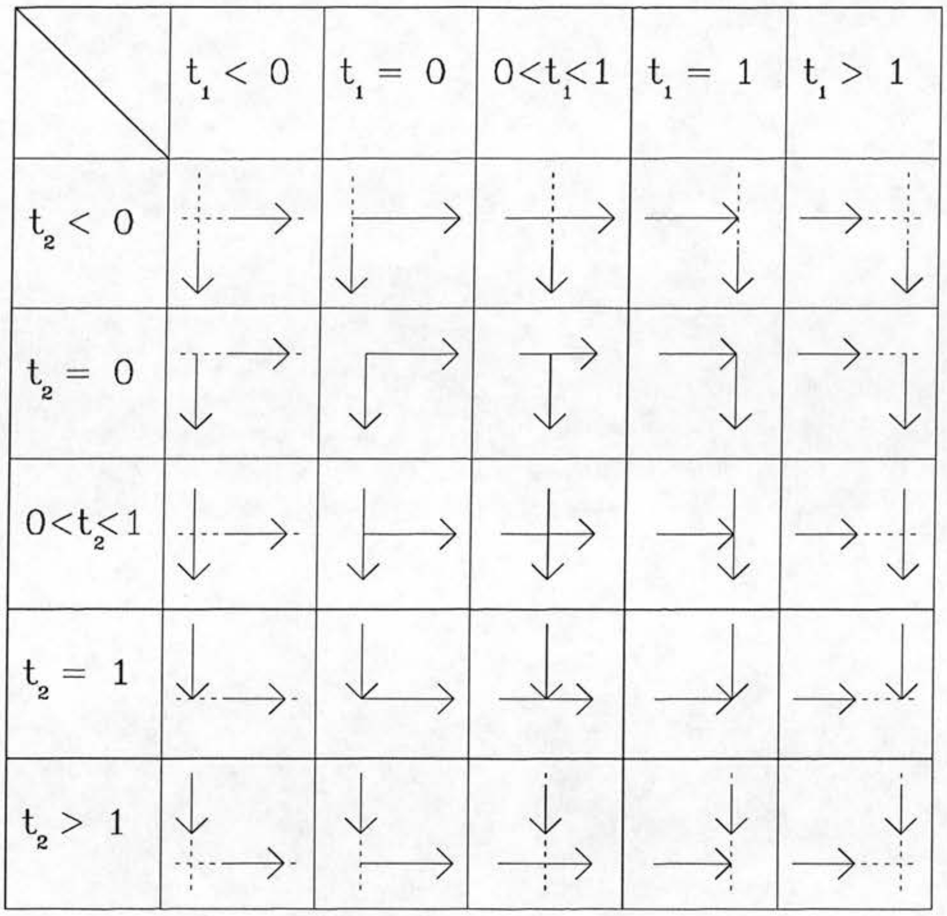

FIGURE 6.2

Now we consider the relation between the algebraic and algorithmic interpretations of **D** for operator symbols of rank *bool line line, dot.* The operation *int_mm*, for instance, is a function whose value is the intersection point of a 'cross intersection' between two vectors. The value of the boolean argument ϕ in *int_mm*(ϕ, α, β) is *true* if the pair $(0 < t_1 < 1$ and $0 < t_2 < 1)$ in Figure 6.2 is selected when α and β are compared. Then, *int_mm(true*, α, β) = p, where p is a 'normal element' in the carrier of terms of sort *dot*. However, if the value of the parameters pair is one among the other 24 possible cases, then *int_mm(false*, α, β) = $e_{<dot,\,i>}$. As was discussed in Chapter 5, errors are propagated and *int_mm*($e_{<bool,\,i>}$, α, β) = $e_{<dot,j>}$ as well.

The boolean argument in *int_em*(ϕ, α, β) is true if there is a 'joint in t' intersection between α and β. The corresponding case in Figure 6.2 is $(t_1 = 1,\ 0 < t_2 < 1)$. When both parameter values are either *0* or *1* a joint of two vectors in their extreme points is determined. The operation symbols in this class are *int_oo, int_oe, int_eo, int_ee.* When one or both of the parameters take a value in either $t < 0$ or $t > 1$ there is no actual intersection, but if one or both vectors were projected, an intersection would occur. These cases are subsumed under the operator symbol *int_ww*. We also refer to this class of intersections as 'conditional intersections'.

Operators of rank *bool polygon polygon, polygon,* namely *union_of, intersection_of* and *difference_between*, are interpreted as a topological operation between polygons. The inclusion of polygon comparison operations allows the definition of complex regions of the space by composition of simple polygons. These operators can be defined in **D** if polygons are defined as regular sets of dots, given that set comparisons between two arbitrary polygons result in normal elements of sort polygon (Tilove 1980). Algorithms for comparing two arbitrary polygons can be efficiently implemented (Weiler 1980, Pineda 1986). The output of these operations is illustrated in Figure 6.3.

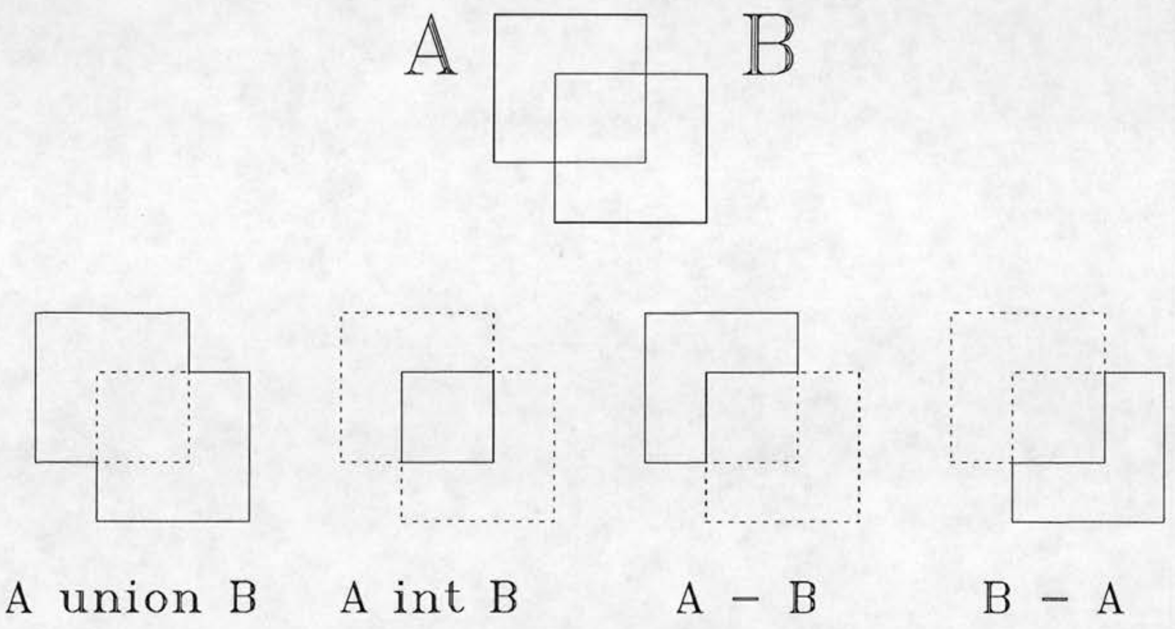

FIGURE 6.3

The algorithmic interpretation of the other operators in **D** is computed by standard geometric algorithms and will not be discussed further here; see, for example (Preparata et al 1985, Shamos 1978).

6.1.2. The Language $\mathbf{L_d}$.

Now, we embed the structure **D** within a modal first order logical language. This will allow us to represent graphical and logical knowledge in an integrated fashion.

The set of sorts $S_{\mathbf{d}}$ of $\mathbf{L_d}$ is obtained by adding a new sort: $S_{\mathbf{d}} = S_D \cup \{individuals\}$ as shown in Chapter 5, these individuals are referred to by natural language expressions, and might be physical or abstract entities.

The modal first order language $\mathbf{L_d}$ is defined as follows:[1]

(19) All operator symbols of $\mathbf{D}$ are also in $\mathbf{L_d}$.

(20) $\Sigma_{bool\, bool,\, bool} = \{\wedge, \vee, \rightarrow, \equiv\}$

(21) $\Sigma_{bool,\, bool} = \{\neg, \Box\}$

(22) *Quantifiers*: For every sort $s \in S_{\mathbf{d}}$, $\forall_s$ (for all), $\exists_s$ (there exists), each of rank $\Sigma_{s,\, bool}$

(23) *Functional Abstractor*: For every sort s in $S_{\mathbf{d}}$, $\Sigma_{s,\, bool} = \{\lambda\}$.

(24) *Equality*: For every sort s in $S_{\mathbf{d}}$, $\Sigma_{s\, s,\, bool} = \{=\}$

(25) *Variables*: For every sort $s \in S_{\mathbf{d}}$, there is a countably infinite set $\mathbf{V}_s = \{x_{s,0}, x_{s,1},...\}$ of variables such that $x_{s,i} \in \Sigma_{e,s}$.

(26) *Constants*: For every sort $s \in S_{\mathbf{d}}$, there is a countable set $C_s = \{c_{s,0}, c_{s,1},...\}$ of constant symbols such that $c_{s,i} \in \Sigma_{e,s}$.

(27) *Predicates*: For every sort $s \in S_{\mathbf{d}}$, there is a countable set $P_s = \{p_{s,0}, p_{s,1},...\}$ of predicate symbols such that $p_{s,i} \in \Sigma_{u_0 u_1 ... u_i,\, bool}$. The string $u_0 u_1 ... u_i$ is called the **arity** of $p_{s,i}$.

(28) *Auxiliary symbols*: "(" and ")".

Formation rules are expressed as follows:

(29) Every constant of sort s is a term of sort s.

(30) If $t_1,..., t_n$ are terms of sorts $s_1,..., s_n$, respectively, and f is an operation symbol of rank w, s, where $w = s_1,..., s_n$, then $f(t_1,..., t_n)$ is a term of sort s.

(31) If $t_1,..., t_n$ are terms of sorts $s_1,..., s_n$, respectively, and f is a predicate symbol of rank $w, bool$, where $w = s_1,..., s_n$, then $f(t_1,..., t_n)$ is a term of sort *bool*.

(32) If $\forall_s$ is a quantifier of sort s, u is variable of sort s and ϕ is a term of sort *bool* then $\forall_s u\phi$ is a term of sort *bool*.

(33) If $\exists_s$ is a quantifier of sort s, u is variable of sort s and ϕ is a term of sort *bool* then $\exists_s u\phi$ is a term of sort *bool*.

(34) If λ_s is a functional abstractor of sort s, u is variable of sort s and ϕ is a term of sort *bool* then $\lambda_s u\phi$ is a term of sort *bool*.

6.1.3. Algebraic Interpretation of $\mathbf{L_d}$.

A *model M for* $\mathbf{L_g}$ is an ordered tuple $<D, I, F>$, where $D = <D_s>_{s \in S}$ is an S-indexed family of non-empty sets, I is a set $\{i_1, i_2,...\}$ of states and F is an interpretation function whose domain is

[1] In the specification of the ranks of these operation symbols we make no explicit provision for the error condition for clarity. But we consider that the rank of each operation symbol and its corresponding semantic interpretation are modified along the lines shown in Chapter 5.

the set of all non-logical and non-graphical[2] constants of $\mathbf{L_d}$ and whose range is described below. If α is is constant of sort s, then $F(\alpha)(i) \in D_s$.

We define as well an assignment function g that has as its domain the set of all variables and has as a value a member of G_s for each variable of sort s.

Now we define the semantic interpretation of expressions of $\mathbf{L_d}$:

(35) If α is a constant of sort s, then $[[\alpha]]^{\mathbf{M, i, g}} = [F(\alpha)](i)$. The extension of a constant symbol at every index $i \in I$ is the same object in D_s.

(36) If α is a variable then $[[\alpha]]^{\mathbf{M, i, g}} = g(\alpha)$.

(37) If α is a term of sort *bool* and u is a variable of sort s, then $[[\lambda u\alpha]]^{\mathbf{M, i, g}}$ is a function h with domain in D_s, such that for any object x in that domain $h(x) = [[\alpha]]^{\mathbf{M, i, g'}}$ where g' is that value assignment exactly like g with the possible difference that $g'(u)$ is the object x.

(38) If f is an operation symbol in $\mathbf{D}$ of rank w, s, for $w = s_1,..., s_n$, then $[[f]]^{\mathbf{M, i, g}}$ is a function with domain in $D_{s_1} \times ... \times D_{s_n}$ and range in D_s. This function, for every operator symbol of **D**, is defined in terms of geometrical analysis, and is computed by its associated algorithm.

(39) If p is a predicate symbol of rank w, s, for $w = s_1,..., s_n$, then $[[p]]^{\mathbf{M, i, g}} = [F(p)](i)$ such that

$$[F(p)](i) \subseteq D_{s_1} \times ... \times D_{s_n}$$

(40) If $t_1,...,t_n$ are terms of sorts $s_1,..., s_n$, respectively, and f is an operation symbol of rank w, s, where $w = s_1,..., s_n$, then $[[f]]^{\mathbf{M, i, g}}([[t_1]]^{\mathbf{M, i, g}},..., [[t_n]]^{\mathbf{M, i, g}})$. The result of applying the function $[[f]]^{\mathbf{M, i, g}}$ to its arguments $[[t_1]]^{\mathbf{M, i, g}},..., [[t_n]]^{\mathbf{M, i, g}}$.

(41) If α and β are terms of sort s then $[[\alpha = \beta]]^{\mathbf{M, i, g}}$ is *1* if and only if $[[\alpha]]^{\mathbf{M, i, g}}$ is the same as $[[\beta]]^{\mathbf{M, i, g}}$.

(42) If ϕ and ψ are terms of sort *bool* then $[[\neg\phi]]^{\mathbf{M, i, g}}$ is *1* if and only if $[[\phi]]^{\mathbf{M, i, g}}$ is *0*, and $[[\neg\phi]]^{\mathbf{M, i, g}}$ is *0* otherwise.

(43) If ϕ is a term of sort *bool* then $[[\phi \wedge \psi]]^{\mathbf{M, i, g}}$ is *1* if and only if $[[\phi]]^{\mathbf{M, i, g}}$ is *1* and $[[\psi]]^{\mathbf{M, i, g}}$ is *1*. (The definition for other truth-functional connectives is given in the standard way along the lines shown for the connective $\wedge$).

(44) If ϕ is a term of sort *bool* and u is a variable of sort s then $[[\forall_s u\phi]]^{\mathbf{M, i, g}}$ is *1* if and only if $[[\phi]]^{\mathbf{M, i, g'}}$ is *1* for all g' exactly like g except possibly for the value assigned to u.

(45) If ϕ is a term of sort *bool* and u is a variable of sort s then $[[\exists_s u\phi]]^{\mathbf{M, i, g}}$ is *1* if and only if $[[\phi]]^{\mathbf{M, i, g'}}$ is *1* for some value assignment g' exactly like g except possibly for the value assigned to u.

[2] The graphical constants are the operation symbols of **D**.

(46) The interpretation of the modal operator $\Box$ is such that if ϕ is an expression of sort *bool*, then $[[\Box\phi]]^{M, i, g}$ is *1* if and only if $[[\phi]]^{M, i, g}$ is *1* for all *i'* in *I*.

Definition of truth:

(47) If ϕ is a term of sort *bool*, then ϕ is *true with respect to M and to I* if and only if $[[\phi]]^{M, i, g}$ is *1* for all value assignments *g*.

At this point, it is worth highlighting one important difference between the way of specifying logical truth functions and functions that are named by the operation symbols of **D**. The semantics of truth functors, like $\wedge$ and $\rightarrow$, is usually defined in terms of a truth table, or as shown in clauses (42) and (43). Given that there are just three distinctive elements in the carrier of sort *bool*, namely *1*, *0* and e_{bool}, we can give these definitions in a discrete fashion. However, the domains of the functions that are named by operation symbols of **D** have an infinite number of distinctive elements. For these definitions, we rely in a background theory that is provided by geometrical analysis. For instance, the function named by operation symbols of rank *bool line line, dot* in clause (12) is implicitly stated in the system of vectorial equations in (18). Furthermore, for each of these operation symbols there is an algorithm that for every collection of arguments of the proper sort, computes the function's value. If the arguments are not proper, an error message is produced. Such a message 'implements' an error element of the operation's sort. Each of these algorithms with its associated error messages can be thought of as computing a total function. For every operation symbol in **D**, the definition and application of such a function are respectively specified in clauses (38) and (40).

Functional application in (37) is defined in the standard way: if x is a variable of sort s, $\phi(x)$ is an expression of sort *bool* and a is a constant of sort s, the application of $\lambda x\phi(x)$ to a, $\lambda x\phi(x)(a)$, is $\phi(a)$. The introduction of λ-terms will be useful for the definition of design concepts as will be shown in Chapter 7.

6.2. Intensionality in $\mathbf{L_d}$.

The language $\mathbf{L_d}$ is intended to express design knowledge in a very simple design domain. Design can be thought of as an activity in which a design object and its properties are identified. Design statements express propositions about objects that do not yet exist in the actual world. However, we can say that a design object is a part of some possible world. For instance, I can say that I am going to design *my house* even if there is no concrete thing in the world that is 'my house'. But there might be a future time in this world in which such a house has been built, or there might be a world that I can imagine in which such a house is already there. This name can be, of course, a graphical representation: the drawing of such a house.

In the design process we discover a set of properties and relations that the design object must have and preserve. However, all of these properties are contingent and varying some, or even all of them, does not destroy the identity of the object as long as there is the intention of designing such an object. We can even consider varying its name; for instance, I might use the term *my house* to refer to the design object at the beginning of the design process, but most likely, I will have to use the term *my wife's house* at the end of it. But this small naming accident is unlikely to change the identity of the object itself. The same can be said of drawings: there might be many graphical expressions that refer to my house and if I am designing this object I must be able to tell whether or not the drawing --or more generally, the representation-- that is produced at some particular design state refers to my house. Of course, there might be several possible representations, but in many situations finding just one is enough. However, in order to make such an identification we need a criterion of identity. This poses a rather puzzling question: how can we identify something that does not yet exist in the actual world? Or rather how can we identify something that is undergoing a process of change?

At the beginning of interactive computer graphics, Sutherland advanced a criterion for the identification of design objects that are represented through drawings in the seminal Sketchpad program (Sutherland 1963, p. 332).

> Construction of a drawing with Sketchpad is *itself* a model of the design process. The locations of the points and lines of the drawing model the variables of a design, and the geometrical constraints applied to the points and lines of the drawing model the design constraints which limit the values of design variables. The ability of Sketchpad to satisfy the geometric constraints applied to the parts of a drawing models the ability of a good designer to satisfy all the design conditions imposed by the limitations of his materials, cost, etc. In fact, since designers in many fields produce nothing themselves but a drawing of a part, design conditions might well be thought of as applying to the drawing of a part rather to the part itself. When such design conditions are added to Sketchpad's vocabulary of constraints, the computer will be able to assist a user not only in arriving at a nice looking drawing, but also in arriving at a sound design.

In GRAFLOG, we explore the same issues from a semantic approach. For this, we introduce the notion of *intension* of a graphical representation.

6.2.1. Extension and Intension of Graphical Objects.

The relation of equality poses many interesting questions about the identity of an object. To understand a statement of the form $\alpha = \beta$, we have to know what object is designated by the expressions α and β. To understand the relation of equality, we have to consider as well 'the manner' or 'mode of presentation' in which an expression designates an object. A statement of the form $\alpha = \beta$ suggests that there are two different modes for designating the same thing. For instance, expressions *3* and *2 + 1* designate the same number. Geometrical objects can be designated in different manners too. Frege gives the following example (Frege 1952, p. 57):

Let *a, b, c* be the lines connecting the vertices of a triangle with the midpoints of the opposite sides. The point of intersection of *a* and *b* is then the same as the point of intersection of *b* and *c*. So we have different designations for the same point, and these names ("point of intersection of *a* and *b*", "point of intersection of *b* and *c*") likewise indicate the mode of presentation; and hence the statement contains actual knowledge.

Frege's triangle is illustrated in Figure 6.4.a.

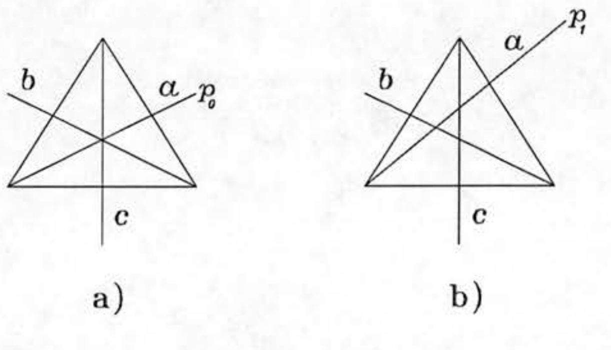

FIGURE 6.4

In relation to the triangle in Figure 6.4.a, the statements "point of intersection of *a* and *b*" and "point of intersection of *b* and *c*" present the same object in two different manners. These two expressions refer to the same object, but they express different concepts and have a different informative content. According to Frege, they express a different **sense**.

Now, consider the interpretation of a statement whose form is $\alpha = \alpha$. It holds *a priori* given that the only thing we need for understanding the truth that it expresses is knowledge of the language. The statement "point of intersection of *a* and *b* = point of intersection of *a* and *b*" expresses an analytic truth. The truth of this statement is granted just by the form of the expression, and we do not have to think of the contingent properties of the lines *a* and *b*, such as their length and position, because there is no drawing in which this statement becomes false. Through $\mathbf{L_d}$ we can generalise this definition for two arbitrary lines in $\mathbf{L_d}$ as follows:

(48) $\Box \forall \mathbf{x}, \mathbf{y}_{\mathbf{line}}[\mathbf{int\_mm(x,y) = int\_mm(x,y)}]$.

Note that *int_mm(x, y)* can denote a normal element of sort *dot* or the error element $e_{<dot, i>}$, but in any case (48) holds for every state *i* in *I* in any model for $\mathbf{L_d}$.

Consider now a statement of the form $\alpha = \beta$. For instance, "point of intersection of *a* and *b* = point of intersection of *b* and *c*". We can imagine some situations in which this statement is true, as is the case in Frege's triangle, but there are other graphical contexts in which it is false, for instance, the state of affairs illustrated in Figure 6.4.b. Here, the position p_0 of one of the extreme points of the line *a* has been changed to p_1 and some of the contingent properties of this line --like length and orientation-- have been altered. Then, the statement "point of intersection of *a* and *b* = point of intersection of *b* and *c*" becomes false because the intersections of *a* and *b* and *a* and *c* are different objects of sort *dot*.

So-called extensional logical systems with equality obey *Leibnitz's Law*. This states that if a statement of the form $\alpha = \beta$ is true, and if a statement ϕ containing α is true, then the result of replacing any occurrence of α in ϕ by β is also true. Consider, for instance, the following expression:

(49) **on(int_mm(a, b), a)**

where *a* and *b* are the lines in Frege's triangle in Figure 6.4.a. This expression asserts that the point of intersection between the lines *a* and *b* is *on* the line *a*. The expression *int_mm(a, b)* is a term of sort *dot* that refers to the intersection of *a* and *b* and (49) is true in relation to the drawing in Figure 6.4.a because this point is also on line *a*. If we substitute *int_mm(a, b)* by another term that has the same referent, say *int_mm(b, c)*, the referent of the composite expression --its truth value-- should not be altered. In fact, (50) is true in relation to the drawing in Figure 6.4.a as well.

(50) **on(int_mm(b, c), a)**

An interesting point that was originally noted by Frege is that such a kind of substitution is not allowed in the context of modal expressions such as *necessarily*. This corresponds with the necessity operator $\Box$ of $\mathbf{L_d}$. Note that the truth value of expression (51) is true,

(51) $\Box$**on(int_mm(a, b), a)**

but the truth value of (52) is false.

(52) $\Box$**on(int_mm(b, c), a)**

Expression (52) is false because there are situations such as the case in Figure 6.4.b where the intersection of *b* and *c* is not on the line *a*.

Failure of this kind of substitution implies that the extension of a complex expression at some given state of affairs is not necessarily a function of the extension of its constituent parts in such a state. Frege noticed this problem in relation to the meaning of natural language expressions, and advanced a hypothesis that has become fundamental for modern semantic studies. According to Frege, the semantic value, or denotation of an expression that appears within the context of an 'opaque operator'[3] like modal expressions, is not the normal denotation of such a term, but rather its sense. In the semantic interpretation of $\mathbf{L_d}$ we have adopted Frege's approach in an explicit way. But before explaining how this assumption has been implemented we have to be more specific on the notion of sense or intension of expressions of $\mathbf{L_d}$.

Following Montague and others (Dowty 1981) we define the intension of an expression as a function from states to extensions. In $\mathbf{L_d}$, the intension of an expression of sort *s* is a function whose domain is the set *I* of states and whose range is contained in the set D_s. The underlying

[3] These kinds of context are known as *referentially opaque constructions*. See, for instance, Dowty et al. pp.143.

idea is this: if the intension of an expression is known, its referent --or extension-- can be identified at every state of affairs in *I*. Next, we illustrate this notion with the help of some simple examples.

Assume that Figures 6.4.a and 6.4.b are respectively associated with the indices i_0 and i_1 in the set of states *I* of some design process.

The intension of an individual α is a constant function that assigns the same object to α for every interpretation state. For instance, the intension of line *a* is the function $\{<i_0, A>, <i_1, A>\}$. Note that we use the same symbol *a* as a name in $\mathbf{L_d}$ and *A* as member of the carriers of sort *line*. However, these two objects should not be confused. A name is a linguistic object, and the range of the function is a set theoretical object. Alternatively, we can think of the linguistic name *a* as the graphical symbol itself.[4]

The intension of an expression ϕ which express geometrical properties of graphical objects is a function that maps every design state *i* to the value of ϕ in *i*. For instance, the position of the end-extreme of *a* in Figure 6.4.a is p_0,[5] but the same value of the same expression in Figure 6.4.b. is p_1. Accordingly, the intension of the expression *position_of(end_of(a))* is the function: $\{<i_0, p_0>, <i_1, p_1>\}$.

The intension of a term of sort *bool* --a formula-- is a proposition: a function mapping every design state to the truth value of such a formula in that state. For instance, the proposition that *on(int_mm(b, c), a)* expresses is the function $\{<i_0, 1>, <i_1, 0>\}$.

A question that is relevant for our enterprise is how the intension of an expression can be known. In modern formal semantic studies, the interpretation function relating the expressions of a language with their corresponding referents in every state is given by definition, and the epistemological question of how such a function can be known is not usually faced directly. In GRAFLOG, the knowledge of the intension of a graphical expression is embedded in the geometrical algorithms that compute its referent. For instance, the position of the point that is referred to by the term *int_mm(a, c)* in Figures 6.4.a --an object of sort *real_pair*-- can be discovered by computing the value of the expression *position_of(int_mm(a, c))* at that state. The value of this expression is either a normal element of sort *real_pair* or the error element of this sort. The referent of such an expression can be determined in other interpretation states in the same manner. If we know the geometrical algorithms that compute the functions named by the operator symbols of **D** then we can come to know the intension of every graphical

[4] See Figure 5.7.

[5] Such a coordinate pair can be referred to by means of operator symbols (7) and (11) of the structure **D**.

expression of **D** and, as has been shown, this is in fact the case.

We can now explain how the principle of compositionality is implemented in the interpretation of $\mathbf{L_d}$. The meaning of constant and predicates names is defined by the interpretation function *F*. The extensions of individual constants and predicates depend on the current state of the knowledge-base. However, the extensions of graphical terms of *D* are computed through geometry. The extension of a complex but non-modal expression is determined by applying the function denoted by the operator term to the extensions of its arguments, as defined by the semantic rules in the interpretation of $\mathbf{L_d}$. In this case, the extension of a complex expression is a function of the extensions of its constituent parts at the current interpretation state. However, the semantic value of expressions of the form $\Box\phi$ is a function of the intension of ϕ. In the context of the modal operator, ϕ has not its customary denotation, but rather denotes the proposition that it expresses. We require for $\Box\phi$ to be true that the semantic value of ϕ is true at every index *i* in *I*.

Now, we give some formulas of $\mathbf{L_d}$ that hold for every interpretation state.[6]

(53) $\Box\forall \mathbf{x, y}\ \mathbf{int\_mm(x, y) \equiv cross\_at(x, y).}$

(54) $\Box\forall \mathbf{x, y, z\ [x = int\_om(y, z) \vee x = int\_em(y, z) \vee x = int\_mo(y, z) \vee x = int\_me(y, z)]}$
$\mathbf{\equiv x = t\_joins\_at(y, z).}$

(55) $\Box\forall \mathbf{x, y, z\ [x = int\_oo(y, z) \vee x = int\_oe(y, z) \vee x = int\_eo(y, z) \vee x = int\_ee(y, z)]}$
$\mathbf{\equiv x = e\_join\_at(y, z).}$

(56) $\Box\forall \mathbf{x, y, z\ [x = e\_join\_at(y, z) \vee x = t\_joins\_at(y, z)] \equiv x = join\_at(y, z).}$

(57) $\Box\forall \mathbf{x, y, z\ [x = cross\_at(y, z) \vee x = join\_at(y, z) \vee x = int\_ww(y, z)]}$
$\mathbf{\equiv x = intersect\_at(y, z).}$

These formulas define a hierarchical categorisation on the 25 intersection cases in Figure 6.2. For instance, the term *e_join_at* denotes any of the four modes in which an 'extreme joint' between two vectors can occur. The term *t_join_at* stands for the four cases of 'joins in t'. The term *joint_at* stands for any kind of join, and the term *intersects_at* denotes any 'actual' or 'possible' intersection between two lines. The term *cross_at* is just a more intuitive term to refer to the intersection *int_mm*.

Terms such as *cross_at, joint_at* etc., can be given a natural language translation, and can be used to talk about graphical relations. In the context of a graphical and linguistic conversation, it might be more natural to say, for instance, *This is the join of these walls* rather than *This is where the end of this wall joins the origin of this wall.* Although the latter expression is more informative, the former is general and abstracts over low-level geometrical and topological

[6] Assume that quantifiers and variables in these formulas are of the appropriate sorts. We also omit the boolean argument for the error condition of each operator symbol for clarity.

considerations. We can imagine different contexts in which it is better to utter one rather than the other. In a conversation between an architect and his customer, the former expression might be natural; but for a designer involved in the definition of the relations that must hold when a drawing is modified, say in a CAD system, the latter expression might be most appropriate. We include both kind of terms for referring to drawings at different levels of abstraction as a means of adding expressivity to our representational environment.

So far, we have explained in detail the notions of extension and intension of a drawing represented through the language $\mathbf{L_d}$. Next, we show the utility of these distinctions for defining a criterion of identity for graphical objects that undergo a process of change.

6.2.2. The Identity of Graphical Objects in $\mathbf{L_d}$.

One fundamental question that is not often raised in the definition of so-called design systems is how the identity of a graphical object is determined. In many traditional approaches design is considered an exploratory task in which a sequence of states of a design process are visited, and eventually, a design object is 'recognised' by the human designer. However, if something is identified at all it is because there is a criterion of identity. This criterion might be implicit; nevertheless, it has to be there. Here, we define a convention for identifying drawings when the geometrical properties of their constituent symbols are altered. Of course, this criterion is conventional and the design task on which we are about to embark is extremely simple. However, the criterion is useful for the solution of some problems that traditional design systems have often suffered from.

In GRAFLOG, the linguistic interpretation of a graphical expression is stated by means of an ostensive definition. Now consider the following question: does a linguistic symbol that is introduced by ostension name the extension of the graphical symbol or rather its intension? For instance, if the expression *This is a wall* is uttered at the time a line in an architectural drawing is pointed out, is the term *a wall* naming the intension or rather the extension of such a line?[7] In this case we might say that *a wall* is naming an extension, that is, the object that the line stands for; this graphical symbol has its 'customary denotation'. However, given that basic graphical symbols are considered rigid designators every basic symbol expresses an intension, although it is a constant function. On the other hand, consider the situation in which a region of the space that is determined by a set of walls is selected at the time *This is a house* is typed in. Here, we could also say that *a house* and the graphical symbol denote an extension: the house itself. However, if such a drawing is altered in the course of a design process --for instance, by modifying the length of the walls-- how can we tell which space partition corresponds to the

[7] Or *wall_1* in the internal representation of the system, as explained in Chapter 2 and 3, and also below in this chapter.

object named by the term *a house* in a new graphical configuration? In other words, how can we tell whether after some modification the drawing still denotes the house or whether it is just a meaningless set of graphical patterns? We assume that in this situation the term *a house* names the intension of the graphical symbol. We can be more specific: the denotation of a graphical symbol that emerges in the context of a graphical context is not 'its customary denotation' but rather its intension.

In the implementation of GRAFLOG we distinguish two kinds of ostensive definitions. The first kind is used for giving names to context-independent symbols whose internal geometrical 'unity' is given beforehand. For instance, the walls of an architectural drawing. The second is used for giving names to complex graphical structures that emerge from the composition of more basic units. For instance, for naming a region of the space standing for a house in terms of a set of basic lines.

The first class of ostensive definitions was illustrated in Chapter 2 when the notion of graphical language was introduced. The symbols that are named through definitions of this kind are taken from a graphical menu and act as the basic building blocks in the graphical compositions. In the representational system, the linguistic names are the indices of their corresponding entries in the geometrical data-base *g_db* in Figure 2.7, where the geometrical information for the construction of the symbols is stored. The second class of definition for naming complex graphical structures will be further illustrated later in this chapter. This kind of ostensive definition establishes a link between a linguistic name and the expression from which a complex space partition is produced. The extension of these expressions is computed in every particular state by the geometrical algorithms that are named by the operator symbols of **D**.

Consider as well that an expression of $\mathbf{L_d}$ might denote normal objects of its corresponding sort, or the error element of that sort. We adopt the following convention: if the denotation of a graphical expression is a normal element of its corresponding sort, its extension corresponds with the object named by the expression; however, if its denotation is the error element of its sort we consider that there is not a state of equilibrium --in the sense defined in Chapter 5-- at that state and the identity of the referred object is not fully determined. We adopt the additional convention that a name in the linguistic domain that is an index to a graphical symbol denotes if and only if the current graphical term denotes a normal element of its sort. In other words, if the graphical expression denotes the error element of its sort, its linguistic name does not denote. This condition is very important for modelling the processes of change in our representational system, and will be discussed further.

Next, we show how the representational environment provided by $\mathbf{L_d}$ is implemented and used in GRAFLOG.

6.3. Interactive Definition of Design Objects.

In this section we present the definition of graphical structures whose basic symbols are expressed in the course of standard graphical interaction, and whose interpretation is captured in the language $\mathbf{L_d}$. For the implementation, the representational structures *NLKB* and *g_db* of GRAFLOG are used.[8]

In this version of GRAFLOG we consider a graphical menu with only one kind of symbol: the line. Symbols of sorts *dot* and *polygon* will be identified in terms of the geometrical and topological relations that are established between the basic lines.

Suppose that the drawing in Figure 6.5 is created by editing five lines, at the time the ostensive definition in (58) is typed in by the user.[9]

(58) *These are walls.*

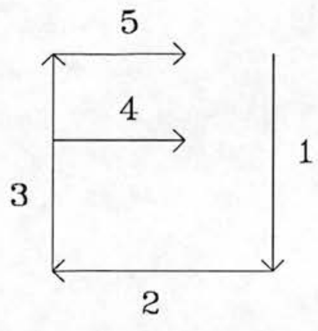

FIGURE 6.5

The symbols introduced by this definition are independent of the graphical context, and they constitute the basic building blocks of the graphical composition. The graphical and linguistic knowledge that is expressed by this definition will update *NLKB* and the geometrical data-base *g_db* in an appropriate manner, along the lines shown in Chapter 2. In the linguistic domain, the following facts will be added to *NLKB*, where the individual names of the lines, like *wall_1*, are arbitrary identifiers provided by the system:

(59) (1) **wall(wall_1).**
(2) **wall(wall_2).**
(3) **wall(wall_3).**
(4) **wall(wall_4).**
(5) **wall(wall_5).**

The geometrical data base is constituted by a set of clauses such that for each graphical object in

[8] See Figure 2.7.

[9] For clarity, we show the order and orientation in which the lines are edited on the screen, but the labels and arrows are not part of the drawing.

the representation there is a clause of the form

(60) **g_db(name, type, description).**

Here, *name* is a constant identifying the graphical object, *type* is the name of some sort in $\mathbf{L_d}$, and *description* is a list of terms denoting the points of the space, the parameters, by which the graphical object is defined. In this version of GRAFLOG, we consider three kinds of descriptions: of basic lines, of polygons, and of construction lines.

The description of basic lines is stated as a list of two constant terms of sort *real_pair* which stand for the positions of the origin and end of the corresponding line.[10] These terms are of course computed by the system at the time the symbols are defined on the screen. The notation is as follows: p_{ij} is interpreted as point i of line j, where i is o for the origin and e for the end of the corresponding line. For instance, p_{o4} is interpreted as the position of the origin of the fourth line. Definition (58) is reflected in the graphical domain as follows:

(61) (1) **g_db(wall_1, line, [**p_{o1}, p_{e1}**]).**
(2) **g_db(wall_2, line, [**p_{o2}, p_{e2}**]).**
(3) **g_db(wall_3, line, [**p_{o3}, p_{e3}**]).**
(4) **g_db(wall_4, line, [**p_{o4}, p_{e4}**]).**
(5) **g_db(wall_5, line, [**p_{o5}, p_{e5}**]).**

The identifier of a given object in *g_db* is also used as the name of the same individual in the domain knowledge-base *NLKB*. For instance, the object that has the property of being a wall in (59.1) is depicted on the screen in terms of the graphical information in (61.1). This naming convention implements the translation relation introduced in Chapter 2.

Now, we come to the definition of the second kind of graphical symbol and its associated description list: objects of sort *polygon.* Polygons are used for representing entities that emerge in the context of overt lines. The interpretation of a polygon is stated through an ostensive definition. However, this kind of ostension differs from the previous one in that polygons are identified in terms of the basic symbols that are already drawn on the screen. This ostension does not introduce overt graphical patterns on the screen but rather establishes a relation between symbols, like the walls, that were introduced before. The linguistic name introduced by this kind of definition names not only a polygon in the actual definition state, but rather the function by which the polygon can be known in different graphical states. The expressions included in the description list of polygons are intensions rather than extensions.

Suppose that when the expression

(62) *This is a house*

[10] Objects of sort *dot* are not implemented directly, and dots are represented through their corresponding positions.

is typed in, a polygon whose vertices are the dots identified in Figure 6.6 is defined on the screen by the user. This identification consists in pointing out the dots in the order indicated by the subscripted values.

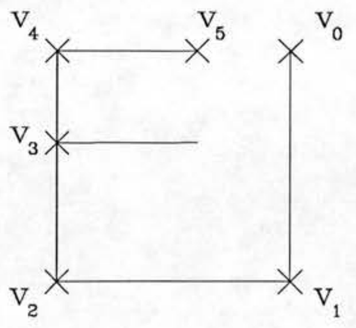

FIGURE 6.6

The definition can be started by pointing at any of those vertices, but the order in which these are identified will be reflected in the polygon's representation in *g_db*; otherwise, a different object would be defined. In the definition of this polygon we have used a graphical cursor of sort *dot*. This is of course an implementational convention and other strategies could be considered; for instance, we could have pointed to the screen with a graphical cursor of sort *polygon*. In Figure 6.6, the edges of the polygon are not explicitly drawn, but this is also a convention.

The linguistic and graphical expressions representing this object are asserted in the representational structures of GRAFLOG. The fact

```
(63)        house(house_1).
```

is added in *NLKB*. In the graphical domain, the graphical object is represented by a clause in *g_db*. The form of this clause is as follows:

```
(64)        g_db(house_1, polygon,
                    [origin_of(wall_1, _),
                    e_join_at(wall_2, wall_1, _),
                    e_join_at(wall_3, wall_2, _),
                    t_join_at(wall_3, wall_4, _),
                    e_join_at(wall_5, wall_3, _),
                    end_of(wall_5, _)]).
```

Note that the polygon's description takes the form of a list, where each term is the translation into PROLOG of a term of sort *dot* of L_d that denotes one of the polygon's vertex. These terms denote the origin or the end of a line, or the intersection of two lines.[11] Obviously, the

[11] The expressions representing the polygon's vertices are inferred to by GRAFLOG's interpreter in the course of the graphical interaction, as will be shown when the graphical parser for D is explained.

translation from $\mathbf{L_d}$ in PROLOG has to accommodate the fact that PROLOG does not allow functions to be expressed directly. Thus, in place of the $\mathbf{L_d}$ operator $origin\_of_{line,\,dot}$ we use a PROLOG relation *origin_of*/3 where the last argument-place stores a term denoting the dot which will be the value of the function. We implement other operator symbols of **D** in a similar way.

According to our criterion of identity for graphical objects, the linguistic designator *house_1* names the intension of the graphical symbol that it indexes. Additionally, *house_1* will denote at every state in which its associated polygon is a normal element sort *polygon*.

The identification of graphical objects in different states is constrained by the geometrical relations between the terms included in the description list of these objects. The condition of equilibrium for this representational system requires that graphical symbols satisfy the following conditions:

(65) If the sort of the symbol is *line*, the positions of the dots defining such a line must be distinct.

(66) If the sort of the symbol is *polygon*, the terms of sort *dot* in the description list define a simple closed path in which no edges intersect each other.

With these considerations, we can update the lines defining the house, as shown in Figure 6.7.

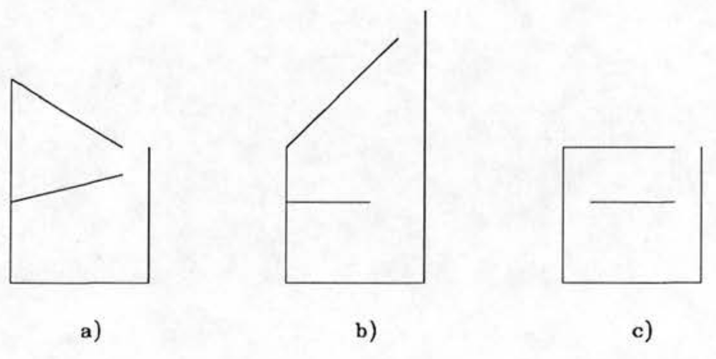

FIGURE 6.7

Under the current house's definition, Figures 6.7.a and 6.7.b are permissible variations of *house_1*. However, the drawing in Figure 6.7.c cannot be identified as the same entity, because one topological specification in the original definition is not satisfied by such a drawing. The error condition can be expressed in $\mathbf{L_d}$ as follows: $t\_join\_at(wall\_3, wall\_4) = e_{<dot,i>}$. As a consequence, $house\_1 = e_{<polygon,j>}$.

Here we can also appreciate why the name of a graphical symbol does not denote if the symbol is in an error state. If we are engaged in the task of designing *house_1*, it makes sense to say that the drawings in Figures 6.7.a and 6.7.b are, according to definition (62), graphical representations of the house that we are designing, but it makes no sense to say the same of the drawing in Figure 6.7.c.

Now, we come to the definition of the third kind of graphical object: construction lines. Construction lines are required for identifying space partitions that are not fully determined by the origin, the end or the intersection of overt lines. Consider the definition of a room of the house. This room is defined be typing in the following expression:

(67) *This is a room.*

Assume that expression (67) is supported by pointing to the marks in Figure 6.8.a, as was shown above for the house.

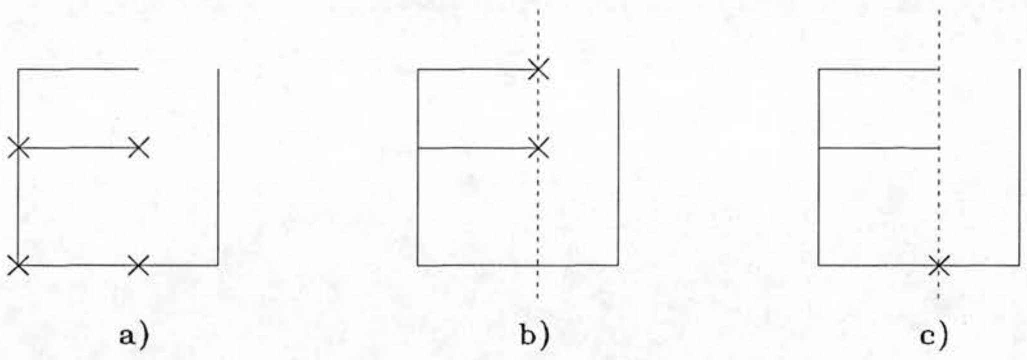

FIGURE 6.8

The room is represented along the lines of other graphical objects named by common nouns, and the fact

(68) **room(room_1).**

is asserted in *NLKB*. For the definition of the graphical object we require, however, a construction line. Three dots are identified as the origin or the end of a line standing for a wall; but one vertex (i.e. the one in the lower right-hand corner) cannot be fully determined just by virtue of the extremes and junctions of the walls.

Construction lines can be defined by selecting two points on the screen as shown in Figure 6.8.b. This line is represented in the graphical domain as follows,

(69) **g_db(c_line_1, line, [end_of(wall_5, _), end_of(wall_4, _)]).**

The identifier *c_line_1* is provided by the system, and the description list is constituted by the two terms of sort *dot* that refer to the points identified in Figure 6.8.b Note that this line 'emerges' from the graphical context in which it is defined. For that reason, terms that refer to its defining dots are expressions of sort *dot* rather than their actual values in the definition state. This is an important consideration because if the lines in terms of of which the construction line is defined, namely *wall_5* and *wall_4*, are altered, the construction line will be modified accordingly. But the relations in the description list of the construction line will remain constant across changes. What varies is the extension of *c_line_1*, but its intension remains the same. In the same way that the a description list of a polygon contains the intension of such a polygon, the description list of a construction line is an intensional description.

We consider a construction line to be the infinite projection of the vector that the terms in its description list define. For that reason, points identified by a construction line are referred to by the operation symbol *intersect_at* in (57), where one or both of the arguments of sort *line* of this operation symbol are construction lines themselves. The function that is named by *intersect_at* has as its value a normal element of the carrier of sort *dot*, unless the two lines that it takes as arguments are parallel or collineal.

Now we show how the construction line is used for the definition of the bottom-right vertex of the room. It corresponds with the intersection of *c_line_1* and *wall_2*, as shown in Figure 6.8.c. Such a dot is denoted by the term *intersect_at(c_line_1, wall_2, _)*. The other three vertices are identified in terms of the basic lines, and the entry for the room in the geometrical representation is the following clause:

```
(70)        g_db(room_1, polygon,
                      [end_of(wall_4, _),
                       intersect_at(c_line_1, wall_2, _),
                       e_join_at(wall_3, wall_2, _),
                       t_join_at(wall_3, wall_4, _)]).
```

Now we can appreciate that the bottom-right vertex of the room will be properly defined in different states of the graphical representation unless the two points defining the construction line become the same, or the construction line and *wall_2* become parallel or collineal. In the case that either of these conditions occurs, not only the bottom-right vertex of the room becomes an error of sort *dot*, but the room itself becomes an error of sort *polygon*.

Before concluding this section it is worth summarising the relation between the abstract representational language $\mathbf{L_d}$ and the representational structures of GRAFLOG that are given in terms of PROLOG clauses.

First, every graphical symbol represented by a clause *g_db* in the implementation corresponds to a basic constant in the abstract specification in $\mathbf{L_d}$. For instance, the individual denoted by the name *wall_1* in (58.1) and (61.1) happens to have the property of being a wall, and happens to have a graphical realisation as a line as well. The points p_{o1} and p_{e1} in its description list in (61.1) are individuals of sort *dot*, and they are basic constants in the abstract representation language $\mathbf{L_d}$ as well. Note that the relation between formal structures in $\mathbf{L_d}$ and the implementation in GRAFLOG is captured by constructing the description list of complex graphical objects, like lines and polygons, out of terms of a more simple geometrical type, like dots.[12] The thread between the formal language and the PROLOG implementation is knitted by making the terms of $\mathbf{L_d}$ that denote the most simple graphical objects into a single object with more structure which is a basic constant of other sort in the formal language.

The translation of expressions of $\mathbf{L_d}$ into PROLOG is as follows:

(71) Every clause of *NLKB* is the translation into PROLOG of an expression of sort *bool* in $\mathbf{L_d}$.

(72) Every clause *g_db* in the geometrical data-base corresponds to a basic constant in $\mathbf{L_d}$ that names an object that has both abstract and graphical properties.

(73) Every term in the description list of graphical objects of types *line* and *polygon* in *g_db* is the translation into PROLOG of a term of sort *dot.*

An important distinction that might require further clarification is which properties of the representational system $\mathbf{L_d}$ and its PROLOG implementation remain invariant across different interpretation states. That is to say, what properties of $\mathbf{L_d}$ hold for every possible model, on the one hand, and which kind of information is contingent and depends on the current information state of the system, on the other.

In the definition of formal languages, there are normally distinguished two kind of constants: logical and non-logical constants. The semantic interpretation of logical constants is defined by an explicit function that holds for every model of the language. For instance, the semantics of *and, or,* etc. are given by their corresponding truth-tables. On the other hand, the interpretation of non-logical constants --individual constants, predicates and relation names-- is stated by the function *F* of the model for the language. This function changes from model to model.

In our theory, the interpretation of logical constants of $\mathbf{L_d}$ was defined in Section 6.1.3. The interpretation of truth-functors, as "$\wedge$", "$\vee$", "$\rightarrow$", etc. was stated in a standard way. Additionally, the interpretation of operation of symbols of **D** was stated once and for all in clause (38). The function *F*, on the other hand, is dynamically defined in every interactive session and depends on the current state of the information of the system.

In the interpreter of GRAFLOG, each logical constant of $\mathbf{L_d}$ is represented by a primitive functor of PROLOG. For instance, the implication "$\rightarrow$" is translated as ":=", and the negation symbol "$\neg$" is translated as PROLOG's functor *not.* Of course, the meanings of these operator symbols are not completely equivalent, and for the implementation some concessions have to be made. There is in particular an important asymmetry between the logical and PROLOG negation (Reiter 1985) but for the purpose of the implementation we adopt the convention that

[12] However, there is no an explicit counter part in $\mathbf{L_d}$ of the geometrical and topological relations between basic graphical objects and the more complex object that they construct. For instance, the relation *part_of*(p_{o1}, *wall_1*) is neither implied nor required in the representational system. The linguistic description of a drawing does not necessarily corresponds with the structure of such a drawing.

these meanings are equivalent. Additionally, there is in GRAFLOG's interpreter a primitive functor that corresponds to each operation symbol of **D**. For instance, *end_of, intersect_at* and *union* are primitive GRAFLOG's functors. When the interpretation of an expression that is formed by these operation symbols is required, its associated geometrical algorithm computes the corresponding function's value. These algorithms are given beforehand to the system, and the function that they compute is the same in every model for the language.

Now, we come to the kind of information that is contingent and depends of the state of the interactive session. In fact, every state of the data bases *NLKB* and *g_db* implicitly defines a particular model. For instance, the constant *wall_1* is graphically realised as a line, and both of these symbols refer to an abstract object *WALL_1* that is only in our minds. However, this association is model dependent. If the line is deleted in the course of the graphical interaction, the association vanishes, and the definition of *F* changes accordingly. *WALL_1* is just an abstraction that we know when we interpret the expressions of the language. In a similar way, the predicate *wall*, for instance, is interpreted as the set {*WALL_1, WALL_2, WALL_3, WALL_4, WALL_5*}. The interpretation function maps such a predicate name to the set of objects that happen to have the property that *wall* names. It is worth emphasising that the objects in this set, the set itself, and the map between predicate name and set are also abstractions in our mind.

Here, we conclude the illustration of the graphical and linguistic definition of 2-dimensional wire-frame diagrams in this version of GRAFLOG. Next, we show how this kind of drawings can be queried in the course of graphical interaction.

6.3.1. Deictic Questions in Graphics Interaction.

The user can address queries about the meaning of the drawing. Suppose, to continue our example, the user picks out one of the locations indicated by a cross in Figures 6.9.a, 6.9.b or 6.9.c, and then types,

(73) *What is this?*

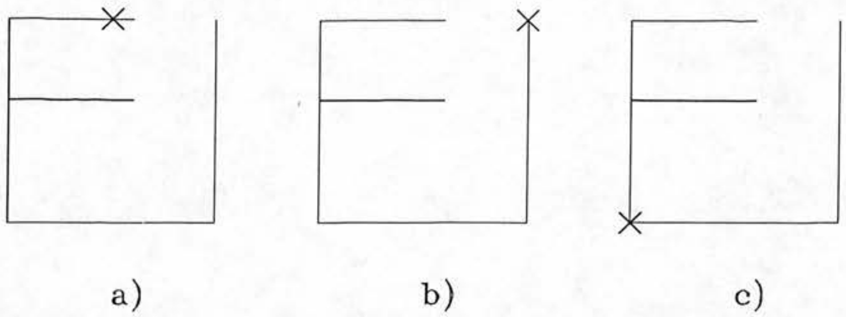

FIGURE 6.9

The answers produced by the system depend on the drawing and the position of the pointing device. Thus, for each of the locations selected above, the responses for the corresponding questions will be

(74) (a) *a wall.*

(b) *the origin of this wall.*

(c) *the join of these walls.*

The linguistic answers are supported by graphical feedback as shown in Figure 6.10. That is, the deixis required by the terms *this wall* and *these walls* is supplied graphically by highlighting the relevant components of the drawing on the screen. This 'highlighting' is indicated by the dotted lines below.

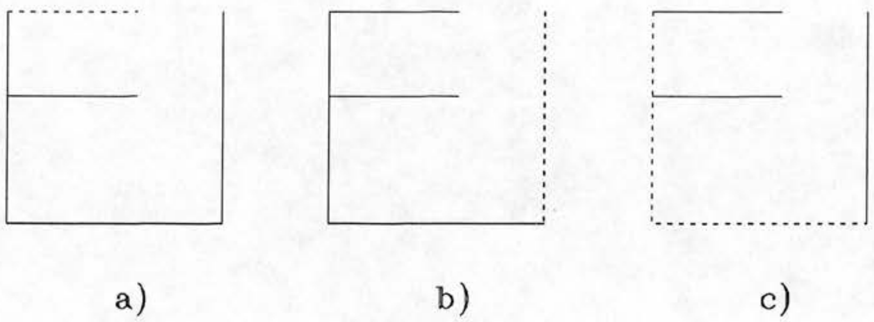

FIGURE 6.10

The expressions *origin of* and *end of* in the natural language answers in (74.a) and (74.b) correspond to the operators *origin_of* and *end_of* in $\mathbf{L_d}$. The words *a, the, this* and *these* are 'function' words that determine quantification and deixis in these referring expressions.

It is worth pointing out that the facility for producing these natural language answers has not been fully developed in GRAFLOG. Although the system is able to identify the referents for the linguistic answers, and also provides a graphical and linguistic context in which they are expressed, the production of an appropriate natural language referring expressions is not a trivial matter, and further research is required. However, within the grammatical and computational framework that we have adopted in the linguistic section of this dissertation, various approaches to the production of referring expressions have been advanced (Dale 1989).

We can ask as well for the identity of objects represented by symbols of sort *polygon*. If the location in Figure 6.11.a is pointed out, and the question

(75) *What is this?*

is asked, the answer is *a house*.

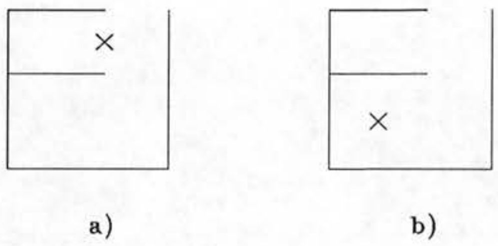

FIGURE 6.11

However, if the same question is asked when the location in Figure 6.11.b is selected, there is a referential ambiguity. The object that is pointed out might be either the house or the room. Nevertheless, there are contexts where such an ambiguity can be resolved, namely when *this* is used anaphorically rather than deictically. Consider the context of a conversation between two individuals, namely *A* and *B*, who are looking at an architectural drawing: a plan of a house with several rooms. Suppose that *A* points to one particular room at the time he says *This room is too small* and then *B* points to the same room and asks *This?* The antecedent for the pronoun *this* in *B*'s question would have to be *a room* although the word *this* in the question is deictic. Furthermore, the room cannot be pointed out without pointing out the house at the same time. This referential ambiguity could not be solved exclusively on the basis of deictic information. The house cannot be the referent of *this* in *B*'s question, but it could be the referent of the same expression in another context in the same conversation. This issue is not further explored in this thesis; however, it is worth pointing out that the relation between deixis and focus in a spatially oriented dialogue is a very interesting and puzzling issue.

6.4. The Parsing of Wire-frame Drawings in L_d.

The goal of the graphical parsing procedure is to find out the interpretation in the structure **D** of the graphical entities that are pointed out on the screen in each individual pointing action. In GRAFLOG, this task is performed by the process *GIP*[13] for the interpretation of the graphical input in the course of the graphics interaction.

Graphical objects are identified by a graphical cursor. Given that there are three sorts of objects that have a graphical representation, we can use three kinds of cursors for identifying objects of

[13] See Figure 2.7

their corresponding sorts. We consider in detail the use of a graphical cursor of sort *dot*, and sketch the utility of cursors of sorts *line* and *polygon*.

In general, there are several terms that denote the same point at some state of the graphics interaction. For instance, the dot v_2 in Figure 6.6 was identified by the term *e_joint_at(wall_3, wall_2)* in the corresponding entry in the description list of *house_1*. However, other terms, like *int_eo(wall_3, wall_2)* or *int_oe(wall_2, wall_3)*, could also be considered. Although all of these terms denote the same object at the time in which the house is identified, they have a different informative value and express a different proposition. Here, we have to choose which term is the one that expresses better the knowledge intended by the user. We partition the term selection task in two stages: first, we have to find out the set of all terms that denote the mark of the pointing device in an individual pointing action. Secondly, we define a criterion for selecting the term that has to be included in the description list of the identified object. Next, we discuss the first of these two tasks.

An expression of the graphical language $\mathbf{L_d}$ can be thought of as a partial description of a drawing. That is, an expression of the form $p = position\_of(origin\_of(\alpha))$, where p names a coordinate point and α is the description of a line, is construed as a statement which is true or false of a given two-dimensional wire-frame drawing Γ. In this case, the drawing plays the role of being a model for a set of expressions of $\mathbf{L_d}$, and we would need to develop a recursive account of expressions being true in such a model.

Alternatively, we can think of a drawing Γ as being a collection of well-formed terms of $\mathbf{L_d}$, closed under some rules of inference (e.g. *modus ponens*). In this case, Γ is construed as a theory in $\mathbf{L_d}$. Here, we need to develop an account of which logical and non-logical axioms would be required for finding out the set of equations that hold for a drawing in terms of the basic graphical facts. We require as well some proof procedures for this theory.

In the current approach we explore the second alternative.

Thus, we think of a drawing as being a set Γ of terms and expressions of the language $\mathbf{L_d}$. We also need to encode information about the pointing device, the graphical cursor *mark*, at any given time. We use the contextual information provided by the set I states of an interactive session, as a way to encode the deictic aspect of *mark*. In standard 'indexical semantics', an expression like *here* is evaluated in a particular context of use i, which then supplies an appropriate referent, say $\mathbf{here}_i$. By analogy, we treat *mark* as a distinguished individual constant of sort *dot* in $\mathbf{L_d}$, such that for any graphical state i, $F(mark)(i) = \mathbf{m}_i$, where $\mathbf{m}_i$ is the dot selected by the graphical cursor in that state i.

We also parameterise Γ in two respects. First, since objects might disappear or be created as we pass from state to state, we need to treat a drawing as a set $\Gamma(i)$ of graphical objects at a given state i. Second, we distinguish the subsets $\Gamma_s(i)$ which belong to a particular sort s. Thus, for example, $\Gamma_{line}(i)$ is the set of terms in Γ at state i which have the sort line. In our example,

(76) $\Gamma_{line} = \{wall\_1, wall\_2, wall\_3, wall\_4, wall\_5, c\_line\_1\}$.

Given the following set of inference rules, we will be able to derive various theorems from $\Gamma(i)$. We write

(77) $\Gamma(i) \vdash \phi$

to specify that the expression ϕ is derivable, or can be deduced, from the theory $\Gamma(i)$.

If t_1, t_2 and t_3 are terms of $\mathbf{L_d}$, (78.1), (78.2) and (78.3) are valid inference rules:

(78) (1) **IF** $t_1 = t_1$ **THEN** $\phi(t_1) = \phi(t_2)$
(2) **IF** $t_1 = t_2$ **THEN** $t_2 = t_1$
(3) **IF** $t_1 = t_2$ **AND** $t_2 = t_3$ **THEN** $t_1 = t_3$

For example, suppose that the drawing represented by the set $\Gamma(i)$ includes $\{wall\_1, wall\_2, end\_of(wall\_1) = e\_join\_at(wall\_1, wall\_2), mark = position\_of(end\_of(wall\_1))\}$. Then we have

(79) $\Gamma(i) \vdash mark = position\_of(e\_join\_at(wall\_1, wall\_2))$

Let $MARK_{dot}(i)$ be the set $\{\alpha \text{ in } \Gamma_{dot}(i) \mid \Gamma(i) \vdash mark = position\_of(\alpha)\}$. This is, every expression α that denotes the same position that *mark* in the state i, belongs to $MARK_{dot}(i)$. Next, we give an inductive definition of how to determine the set of terms in $MARK_{dot}(i)$.

(80) For every α and β in $\Gamma_{line}(i)$ which are not parallel or collineal, if $\Gamma(i) \vdash mark = \text{position\_of}(int(\alpha, \beta))$ then $int(\alpha, \beta)$ is in $MARK_{dot}(i)$, where *int* is an intersection operator in clause (12) of the definition of **D**.

(81) For every α in $\Gamma_{line}(i)$ if $\Gamma(i) \vdash mark = \text{position\_of}(\beta(\alpha))$ then $\beta(\alpha)$ is in $MARK_{dot}(i)$, where β is an operator in clause (11) of the definition of **D** (*end_of, origin_of*).

(82) Nothing is in $MARK_{dot}(i)$ except as stipulated by clauses (80) and (81).

Now we come to the identification of lines and polygons by a cursor of sort *dot*. We define the sets $Selected_{lines}(i)$ and $Selected_{polygon}(i)$ of lines and polygons respectively that are identified by a cursor of sort *dot* at state i. The terms included in these sets are specified in the following inductive definitions:

(83) $Selected_{line}(i) = \{\alpha \in \Gamma_{line} \mid on(mark, \alpha)\}$.

(84) $Selected_{polygon}(i) = \{\alpha \in \Gamma_{line} \mid in(mark, \alpha)\}$.

In this section, the mark of the pointing device has been defined as a constant of sort *dot*. As was mentioned, a more flexible scheme can be defined by considering graphical pointing devices, graphical cursors, of sorts *line* and *polygon*. Such a functionality would allow not only the identification of basic lines and polygons but also of lines and polygons which emerge in drawings in terms of the overt symbols and the composition rules of **D**. The particular interest in our design domain context is the identification of polygons that are produced by the polygon comparison operations; that is to say, by terms produced by operators of rank *bool polygon polygon, polygon* in **D**. Such a device would allow us to impose linguistic interpretations by ostension on complex structures that emerge in drawings in a very direct manner. Suppose, for instance, that polygons *A* and *B* in Figure 6.3 are interpreted as *linguistics* and *programming* respectively. Then we could point to the intersection between these two polygons with a pointing device of sort *polygon* --a graphical cursor for polygon definition-- when the ostensive definition *This is science* is typed in. Now, suppose that one of the basic polygons is modified. The interpretation of the intersection, *science*, would vary accordingly. The geometrical clause for *science* is as follows:

(85) **g_db(science, polygon, [intersection_of(linguistics, programming, _)])**

where *linguistics* and *programming* are symbols of sort *polygon*.

The implementation of graphical cursors of sort *line* and *polygon* would allow the definition of the sets $MARK_{line}(i)$ and $MARK_{polygon}(i)$ along the lines of the set $MARK_{dot}(i)$ above. These facilities are useful because not only dots, but also lines and polygons can be referred to by a number of terms of $\mathbf{L_d}$ at any interaction state *i*.

Now, we come to the selection of the particular term in $MARK_{dot}(i)$ that is chosen by the GRAFLOG interpreter in the graphical parsing procedure. Note that the terms included in the description list of a polygon are not the basic operator terms of **D** which compute the actual intersection point, but they are rather terms produced by one of the intersection naming axioms which subsumes a family of intersection cases. The operators *e_join_at*, *t_joint_at* or *intersect_at* reflect better the knowledge about a graphical relation that a user might have at the definition state. Terms build up with these operators are less informative than the operator symbols that name the topological relations that actually hold between the vectors involved, but that information is hidden in normal human communication as well. When a human user point to an overt dot on a drawing, he might not be aware of the underlying geometrical information that is available through the representational system. Selecting these terms underspecifies the definition of the graphical object, but leaves a greater space for modification of drawings. If the more informative terms were included in the description list of a graphical object, the possible alterations for a drawing might be artificially restricted.

For the definition of the selection procedure, the terms that are considered for identifying a dot are ranked according their discriminatory value in relation to the other expressions that could refer to the same dot. For instance, both of the terms *e_joint_at* and *t_joint_at* cover four basic modes of intersection; however, the *e_joint_at* case marks not only the mode but also the location of the dot at which the intersection takes place. Accordingly, *e_joint_at* has a higher priority value than the term *t_joins_at*. We also take the conventional decision that origin and end points of vectors have a larger priority of selection than the 16 intersection cases in Figure 6.3 that are covered under the term *int_ww*. The operator term *int_ww* is not directly used, and it is subsumed in the intersection term *intersect_at* for the definition of construction lines. Terms that are included in the description list of a graphical object in *g_db* are selected according to the following priorities: *cross_at = 5, e_joint_at = 4, t_join_at = 3, origin_of = 2, end_of = 2, intersect_at = 1,* and *int_xx = 0.*

Now, the term of $MARK_{dot}(i)$ that is included in the description list of a graphical object is selected as follows:

(83) Among the terms *cross_at, e_join_at, t_join_at, origin_of, end_of, intersect_at* or *int_xx* select the term with a highest priority value. If there are several terms with the same priority, such that there are no terms with a higher priority, select any one of them.

(84) If $MARK_{dot}(i)$ is an empty set, no intersection is identified.

For instance, for selecting the term that refers to the point v_0 in Figure 6.6 and is included in the description list of *house*, the set $MARK_{dot}(i)$ is

(85) **{origin_of(wall_1), int_ww(wall_5, wall_1), int_ww(wall_1, wall_5)}**

According to the selection procedure, the term included in the description list of *house_1* is *origin_of(wall_1).*

6.5. Causal Relations and the Interpretation of State Transitions.

Intersections of construction lines are useful for modelling causal relations in the process of modifying a drawing. In fact, these points provide a reference that is common to the drawings before and after a modification takes place. More generally, an intersection point --between overt and construction lines-- is a kind of 'pivot' over which transformation functions can be defined.

Let us define a geometrical constraint that a drawing must satisfy in the course of a design task. We define as well a transformation function that reinforces such a constraint on a drawing whenever is required. Suppose that in the main example of this chapter, we have defined an additional construction line, namely *c_line_2*,[14] and it extends from the origin of *wall_5* to the origin *wall_1*, the right wall. Now, suppose that we impose the following two constraints upon

[14] *c_line_1* was introduced in the definition of the room.

the drawing that hold for every design state:

(86) **collineal(wall_5, c_line_2).**

(87) **∃x.x = join_at(wall_5, c_line_1).**

The interpretation of these formulas is as follows: for every design state, (86) requires that the front wall, *wall_5*, has the same direction than the construction line *c_line_2* and (87) stipulates that an extreme of *wall_5* joins the other construction line *c_line_1*. The references for such design constraints can be graphically expressed in the drawing as shown in Figure 6.12.a. Now suppose, for the sake of the example, that we are interested in modifying the house, and the origin of the right wall is selected and extended in the course of the interactive session, as shown in Figure 6.12.b. In order to satisfy constraints (86) and (87) above, the position of the end extreme of *wall_5* has to be translated to the intersection point of the two construction lines, as shown in Figure 6.12.c.

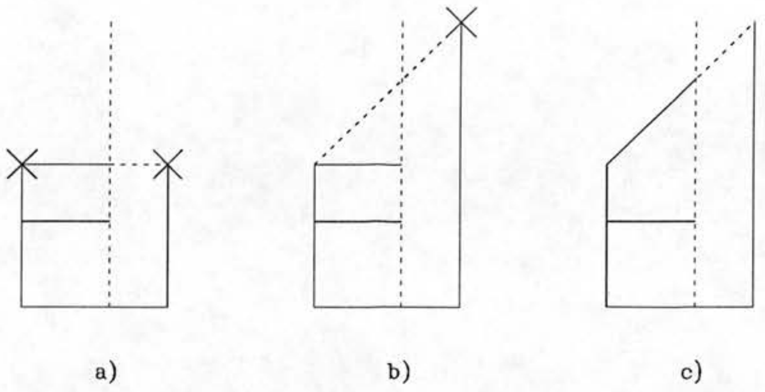

FIGURE 6.12

Now, we come to the definition of a transformation function that enforces constraints (86) and (87) in the course of an interactive session. Such a function can be defined as a PROLOG clause that is proved by the interpreter at the end of every interactive cycle, before the interactive control is returned to the human-user. For the current example, the clause is as follows,

```
(88)   change_rule :- not collineal(wall_5, c_line_2),
                      intersect_at(wall_5, c_line_1, POINT),
                      move_end(wall_5, POINT).
```

where *collineal* and *intersect_at* are the translations in PROLOG of the corresponding geometrical predicate of **D**; *move_end* is a transformation function that changes the parametric position of the end-extreme of a line. The arguments of *move_end* are of types *line* and *real_pair*. This rule could be paraphrased as follows:

IF not collineal(wall_5, line_2) AND intersect_at(wall_5, line_1, POINT)
THEN move_end(wall_5, POINT).

In the implementation of the design interpreter there are two clauses, namely *move_end* and *move_origin*, which modify the end and origin extremes of a line of the graphical representation. These two clause have no translation in $\mathbf{L_d}$, but they stand, at the implementational level, for the *change* transformation function. The semantics of these transformation functions is defined along the lines of the change rule in Section 5.3, and we follow the consequences of this kind of transformations in Chapters 7 and 8.

This example is an illustration of the kind of interactive manipulation that can be supported by an underlying graphical and logical representational structure $\mathbf{L_d}$. It is worth pointing out that different strategies for defining such a kind of constraints and transformation functions through the natural language and graphics dialogue can be envisaged. This facility of the design interpreter can be thought of as a very straightforward AI production system, but with a strong interface component. In the current prototype, human-users can define complex PROLOG applications. Furthermore, the rich semantics provided by the language $\mathbf{L_d}$ is used for the definition of an additional layer of functionality. In our simple design domain, transformation functions are not only applied to drawings by the design interpreter: change rules can also be deduced by the system in the context of some design tasks, as will be shown in Chapters 7 and 8.

Here, we conclude this chapter on the definition of the graphical and logical language for design $\mathbf{L_d}$. The language is useful for representing the semantics of drawings made out of dots, lines and polygons. The notions of extension and intension for drawings represented in $\mathbf{L_d}$ were discussed. We show how the representational system is integrated to the interactive interface and how the semantic interpretation of drawings is introduced by means of direct manipulation on drawings in the context of natural language expressions. The conditions that a name and its associated graphical representation must fulfill in order to refer were also given. The semantic representation of drawings is automatically deduced by the interpreter by means of a set of identification rules that act upon the graphical and linguistic input. We have also illustrated how the language can be used for the definition of causal relations between states in simple design tasks. In Chapter 7, we show how this language is used for the definition of objects that have not only graphical but also linguistic properties, and the notions of design concept and intention in our simple design domain are introduced. In Chapter 8, a function for design intention satisfaction in the domain of 2-dimensional wire-frame diagrams is presented.

Chapter 7

The Notions of Design Concept and Intention

In this chapter the notions of design concept and intention in a very simple design are presented. In Section 7.1, we define the notion of design concept and then how design concepts are expressed and represented in the graphical and logical language $\mathbf{L_d}$ is illustrated. In Section 7.2, an example of a design concept in our simple design domain is presented. In Section 7.3, the notion of design intention is introduced. This notion is related to the process in which a design object undergoes a process of change. Human designers express design intentions about the objects that are being designed. The role of the system is to infer such intentions and to produce the change that is intended by the human designer. The design process is modelled as a logical and graphical inference, in the lines presented in Chapter 5.

7.1. The Notion of Design Concept in GRAFLOG.

We define a design concept as the conjunction of properties and relations, geometrical and linguistic, that a design object must have at some state in the design cycle. In this view, design concepts are definitions: the list of characteristics, the specification of the function and requirements that a design object must satisfy at some state of knowledge of the designer. Here, we have to consider two kind of definitions: the first is an original 'baptism' by which the referent of a design task is fixed; the second is the assertion of the properties that such an object must have. The baptism occurs once when the design process is started; however, the definition of the properties of the design object can vary according to the evolution of the design task in the design cycle. Design concepts can be expressed through an appropriate representational language as, for instance, $\mathbf{L_d}$.

In the course of a design task, design objects are baptised by some arbitrary name. In GRAFLOG, the name of a design object is given by an overt ostensive definition. We consider the names of design objects as rigid designators; accordingly, they denote the same design individual through the whole of a design cycle. Given that this baptism is always explicit, and the cluster of properties of a design object is equated its name, we take the Russellian view that the definition of a design concept holds as an *apriori* and necessary true in the idiolect of the designer and the CAD system at some given design state. Of course, such definitions can vary along the evolution of the design task, but at every state of equilibrium, they are considered analytic truths.

In the kind of design task that we are interested in, design knowledge is expressed through drawings and also through language. Geometrical form or shape is usually expressed by means of graphical representations, but its interpretation --function, specifications, standards, costs, materials, tolerances, etc. -- is expressed through natural language, or through an appropriate logical or mathematical language. We have to consider as well that a design specification might be satisfied by more than one object; for instance, several geometries or different materials might satisfy all the requirements expressed in some given design concept. Accordingly, we define a design concept as a function from individuals to truth values. That is to say, if an individual satisfies all specified requirements and constraints at some design state then it is within the extension of the current design concept. Let Φ be a cluster of properties that are known by the human-designer at some design state, and let x be a variable standing for some object to be designed. A design concept is represented in $\mathbf{L}_\mathbf{d}$ as

$$(1) \qquad \boldsymbol{\lambda x.\Phi(x)}$$

In a more intuitive way, we say a design object is the particular individual within the extension of a design concept whose graphical description is provided in the current design state of a design process. Note that a graphical representation of a design object is satisfied by only one individual within the extension of its design concept. This is so because all the functions named by the operators of the graphical structure are instantiated by every graphical representation of the design object. The values provided by this instantiation will be different for different drawings. Then, each individual in the set of design objects has, in addition to the overtly defined properties, a set of properties provided by the representational system and the overt drawing.

In our theory, the relation between a human-designer and the representation of design objects and design concepts is illustrated in Figure 7.1.

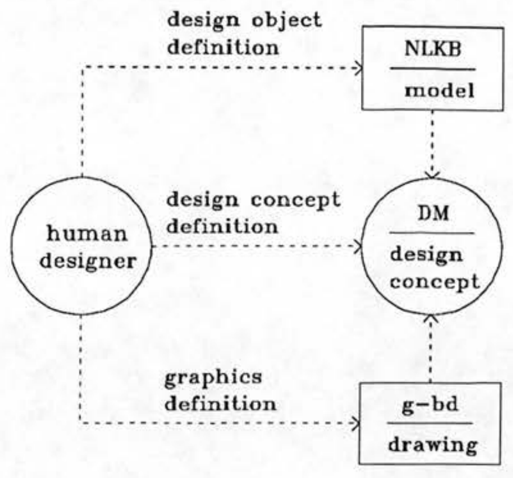

FIGURE 7.1

In the lines of the graphical and linguistic dialogue of Section 6.3, objects are expressed in the representational system by drawing their graphical representations and by specifying their linguistic properties. In the design cycle, one such object is selected as the subject of a design task, fixing in this way the referent for a design concept. Then, the definition of the properties that a design object must have --the design concept-- is specified in the same kind of dialogue. Note, however, that the definition of the design concept is not asserted in the natural language knowledge-base, but rather in the Dialogue manager *DM*. This is so because the design concept is the knowledge that actively guides the design task, and it is used by the dialogue manager to test the equilibrium between the graphical and linguistic representation of the design object. We consider that in every definition state, there is equilibrium between these two sorts of representations. Here, it is worth recalling that equilibrium in this context is not a notion of 'consistency' but rather a notion of reference: equilibrium holds whenever both linguistic and graphical representations of the design object refer. According to our definition, a graphical representation refers if and only if the current description of a graphical object satisfy the geometrical properties of its graphical type, as defined in sections 6.2. and 6.3.

In GRAFLOG, the representation of a design concept and of the current design object is accessible to the design interpreter *ADP*. This relation is illustrated in Figure 7.2. Design conditions asserted in *NLKB* are terms of sort *bool* in $\mathbf{L}_\mathbf{d}$. The drawing, on the other hand, is interpreted by constructive geometrical and topological procedures.

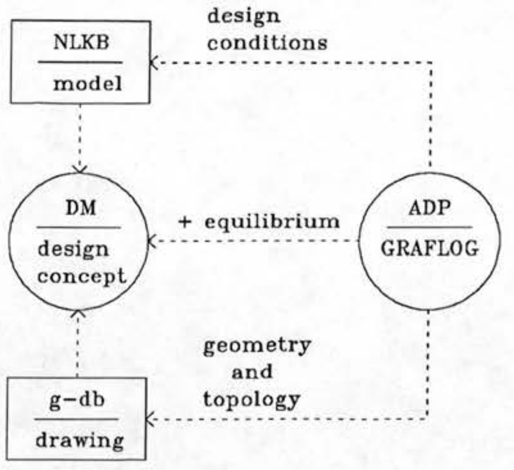

FIGURE 7.2

Note that the design interpreter *ADP* 'sees' the equilibrium between graphical and linguistic systems of representation through *DM*. In fact, in every main control cycle of the system, there is an equilibrium testing phase, and *ADP* becomes active only when this condition is not satisfied.

Next, we come to the analysis of the dynamic behaviour of the system. Assume that there is a design concept expressed in the representational structures of GRAFLOG. Assume as well that there is a state of equilibrium between *NLKB* and *g_db*. Now, suppose that the human-user manifests a design intention, and some graphical symbol is altered on the drawing by standard graphics interaction. The immediate consequence of the human manipulation is unconditionally executed by the interpreter, along the lines of graphics interactive editors. When such an action takes place, the human designer is aware of the current interpretation of the design object, as illustrated by the dotted arrow labelled *design object interpretation* in Figure 7.3. He must know the meaning of the current graphical and linguistic representation in order to express a design intention. In relation to the graphical input event itself, we make the conceptual distinction between 'the information' and 'the intention' that such an action expresses. The intended interpretation for the dotted arrow with the label *graphics manipulation* in Figure 7.3 is that it conveys the geometrical information required to update the geometrical data-base *g_db*. The interpretation of the arrow labelled *human design intention*, on the other hand, expresses the 'intention' underlying such an action. The design intention itself has to be inferred by the system on the basis of the control information that passes from the human designer *HU* to the dialogue manager process *DM*, and from the graphical and linguistic context in which such an action is expressed. The system's inference process starts when *DM* enters into an equilibrium testing phase in relation to the current drawing and design concept. In other words, *DM* verifies that the graphical representation of the subject of the current design task can be constructed in the representational system. If this is so, the design concept has a referent. If the graphical representation still satisfies the current design concept, then the equilibrium is preserved and the control of the interaction is returned to the human-user, as indicated by the arrow with the label + *equilibrium*. However, if the equilibrium test fails, the design interpreter starts a design inference process, and the control is passed to the design interpreter *ADP*, as indicated by the arrow labelled - *equilibrium* in Figure 7.3.

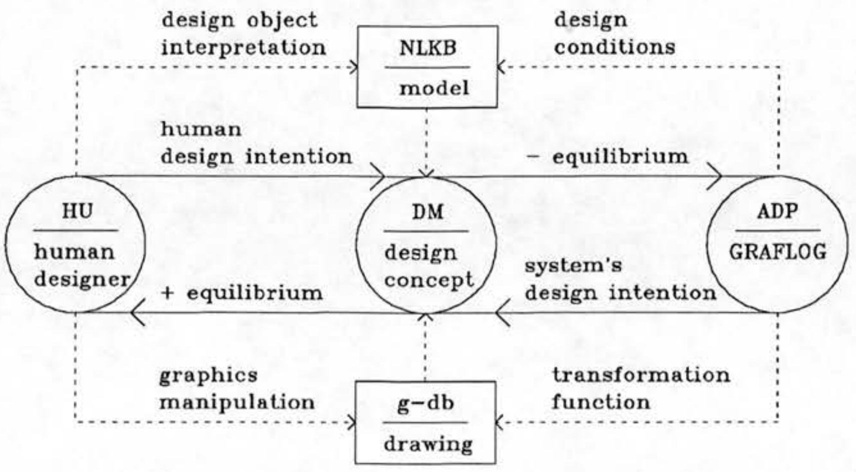

FIGURE 7.3

Now, GRAFLOG has to make a design inference by which the representation of the design object that is intended by the human designer is found. In this process, several factors have to be considered. In the first place, the goal for the overall process of change has to be determined by the beliefs that the system holds as necessarily true in the design state in which the design intention was manifested. These beliefs are accessible to *ADP* in *NLKB*, as shown by the dotted arrow *design conditions*. In general, equilibrium will be achieved when a constructive procedure for the graphical representation of the design object in some new state is found. Finally, a state of equilibrium is achieved when *ADP* modifies the drawing in some appropriate manner.

In the lines of modifications of drawings produced by human designers, we make a conceptual distinction between the information and the intention of modifications produced by the system. The dotted arrow with the label *transformation function* expresses the geometrical information by which *g_db* is modified, and the arrow labelled *system's design intention* expresses the intention behind such a modification. System design intentions are deduced in relation to the overall design task, and they are of course conditioned by the main design intention expressed by the human designer. There is an equilibrium test in every transformation cycle, and the process goes on until this test succeeds. In this scheme, we have to consider whether the process converges to the satisfaction of the design intention, or whether there is not such satisfaction at all. When the inference cycle is concluded, the interactive control is returned to the human-designer, as shown by the the arrow + *equilibrium*.

The definition of a design concept can be extended in every state of equilibrium by modifying the list of properties and relations of the design object. Design concepts undergo processes of change when the human-designer realises new requirements and constraints of the design object. This is a natural step in the traditional design cycle. However, changes in the linguistic definition of design concepts do not usually set the system to disequilibrium states because the design object exists as long as there is a well formed drawing. However, updating the linguistic representation can lead to internal inconsistency in the linguistic representation. For dealing with this sort of inconsistency a Truth-Maintaining System (Doyle 1979) would be required. Logical inferences in such a situation would be nonmonotonic, and our representational system is unable to cope with such a problem at the moment. However, the study of nonmonotonic logics has been subject of a very large number of studies and it is not further considered here (Moens 1989). Here, is worth pointing out that there are some theories of design (Tomiyama 1987, Millington et al 1988, Veerkamp 1988) in which the role of Truth-Maintenance Systems is thought of as paramount, and failure in keeping track of nonmonotonic inferences is considered one of the larger obstacles for modelling design processes. However, I believe that the most important issue in a design process is the construction of the design object itself, and it is not clear how nonmonotonic reasoning is related to this more fundamental issue. But it might be the case that these two issues are altogether unrelated.

7.2. Incremental Definition of a simple Design Concept.

In this section, we develop further the graphical and linguistic dialogue of Section 6.3. We introduced the definition of a set of walls, a house, etc. Now, consider that although these objects have been described, nothing has been said so far of them being the subjects of a design process. In fact, any such individual or any arbitrary subset of them could be the subjects of a design task. We might be engaged in designing the house, or one room within the house or even one wall in such a context. It is clear that the shape of the house, for instance, will determine the properties of the other individuals that are contextually related. However, we have to express, in an explicit manner, which entity is being designed, and that statement will have repercussions in the overall design process. A design object has a cluster of properties and one such property asserts the fact that it is the subject of the current design task. According to the general definition in (1), a design concept is of the form,

$$\text{(2)} \qquad \boldsymbol{\lambda x.(\text{design\_object}(x) \wedge C_0 \wedge C_1 \wedge \ldots \wedge C_n).}$$

where x designates the design object, and $C_0 \wedge C_1 \wedge \ldots \wedge C_n$ is an arbitrary and possibly null conjunction of properties, or constraints. This set of properties can be null given the fact that the design object can be represented by a basic term in the graphical domain, when the design object is just a basic geometrical construction that has been given a name.

When the interpretation of the basic graphical objects --like the house and the room-- has been introduced, we can define one among these objects as the subject of a design task. We can express the following ostensive definition,

(3) *The design object is this house.*

at the time the house that was defined in Figure 6.6 is pointed out. We repeat this figure here for clarity.

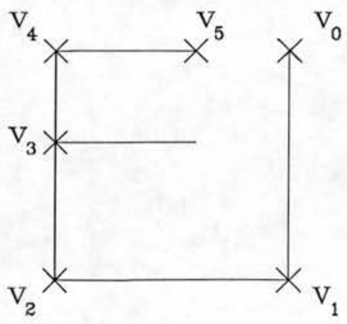

FIGURE 6.6

Definition (3) fixes the referent for the current design task, and the fact *design_object(house_1)* is asserted in *NLKB*.

Now, the current design concept is changed to

(4) **λx.(design_object(x) $\wedge$ house(x)).**

The individual denoted by *house_1* is within the extension of such a concept because the application

(5) **λx.(design_object(x) $\wedge$ house(x))(house_1)**

is equivalent to

(6) **design_object(house_1) $\wedge$ house(house_1).**

This is turn is satisfied in the graphical and linguistic domain in the current state of the system.

In GRAFLOG, there is a current design concept and a current design individual in every design task. The design concept is represented as a PROLOG clause supported by the Dialogue Manager *DM*. The form of this clause is

(7) **design_concept(X) :- design_object(X), Φ(X).**

where Φ stands for the conjunction of all the properties of the design object in the current design state. In our example, the current design concept is represented in *DM* as,

(8) **design_concept(X) :- design_object(X), house(X).**

Now, we can illustrate the equilibrium testing phase of every interactive cycle in detail. There is a state of equilibrium if conditions (9) and (10) hold:

(9) The graphical representation of the design object x is not an error element of its corresponding graphical sort in *g_db* in the current state.

(10) The goal *design_concept(x)* is satisfied by a unique object in the current state of *NLKB*, where *x* denotes the current design object.

The satisfaction of clause (9) depends on the geometrical procedures associated with the graphical structure *D* in Section 6.1. The satisfaction of (10) depends of the knowledge represented in the current state of *NLKB*. In the example, the clause *design_concept(house_1)* can be satisfied in *NLKB* in terms of the basic facts *design_object(house_1)* and *house(house_1)*.

In the graphical domain, however, the design concept given above certainly identifies a number of houses. To appreciate this point, we repeat Figure 6.5 and clause (6.64) for clarity.

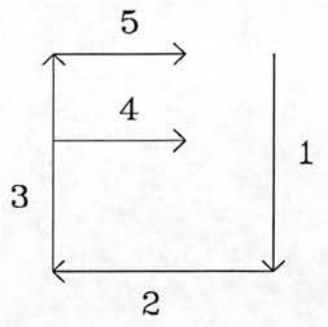

FIGURE 6.5

```
(6.64)          g_db(house_1, polygon,
                          [origin_of(wall_1, _),
                           e_join_at(wall_2, wall_1, _),
                           e_join_at(wall_3, wall_2, _),
                           t_join_at(wall_3, wall_4, _),
                           e_join_at(wall_5, wall_3, _),
                           end_of(wall_5, _)]).
```

For instance, the three drawings in Figure 7.4 are valid houses.

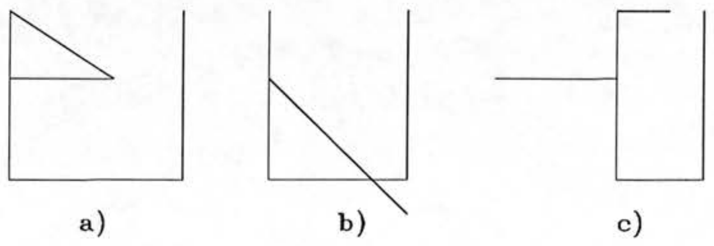

FIGURE 7.4

The design process can be concluded by choosing one of the objects in in Figure 7.4. Or we can specify further the definition of the design object, and type expressions (a) to (c) in (11) at the time their corresponding graphical referents are pointed out in the screen, as shown in Figure

7.5.

(11) **(a)** *These are parallel.*

(b) *This is horizontal.*

(c) *This is in the house.*

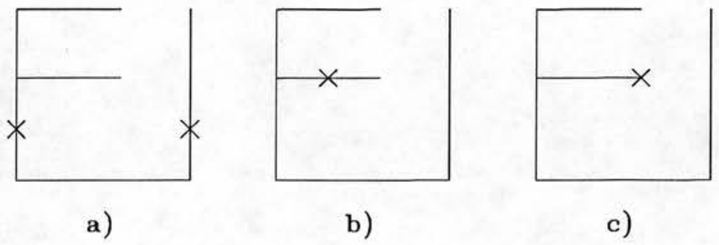

FIGURE 7.5

These properties increment the definition of the current current design concept, as shown in (12),

```
(12)        λx.[   design_object(x) ∧
                       house(x) ∧
                       parallel(wall_1, wall_3) ∧
                       horizontal(wall_4) ∧
                       in(end_of(wall_4), x)
              ].
```

In the implementation, the current design concept in *DM* is the clause

```
(13)        design_concept(X) :-
                       design_object(X),
                       house(X),
                       parallel(wall_1, wall_3),
                       horizontal(wall_4),
                       end_of(wall_4, Y),
                       in(Y, X).
```

Examples of individuals within the extension of this more refined design concept are shown in Figure 7.6.

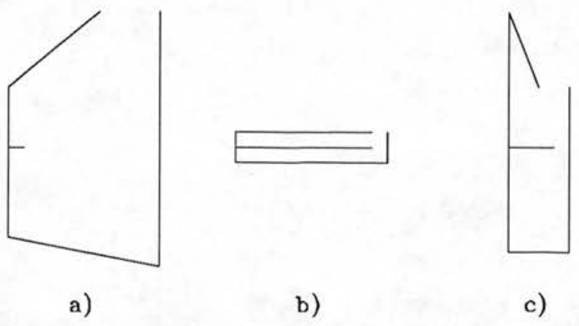
FIGURE 7.6

Now, suppose that the rooms and the corridor are defined by typing the expressions (a) to (c) in (14) at the time their referents are graphically pointed out in the corresponding drawings in Figure 7.7.

(14)	**(a)**	*This is the bedroom.*
	(b)	*This is the living-room.*
	(c)	*This is a corridor.*

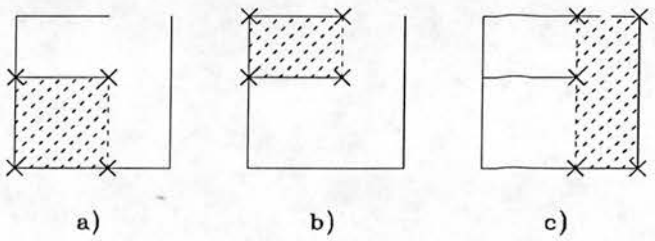
FIGURE 7.7

The corresponding representation of these expressions in *NLKB* is

(15)	**(a)**	**room(bedroom).**
	(b)	**room(living_room).**
	(c)	**corridor(corridor_1).**

Now, suppose that we want to say as part of the design concept that the living-room is always larger than the bedroom. The concept of the house is then extended to[1]

$$
\begin{aligned}
(16)\quad \lambda x.[\quad & \mathbf{design\_object(x)} \wedge \\
& \mathbf{house(x)} \wedge \\
& \mathbf{parallel(wall\_1, wall\_3)} \wedge \\
& \mathbf{horizontal(wall\_4)} \wedge \\
& \mathbf{in(end\_of(wall\_4), x)} \wedge \\
& \mathbf{y = area\_of(bedroom)} \wedge
\end{aligned}
$$

[1] Assume that we have extend the signature of L_d to allow arithmetic operations and relations.

z = area_of(living_room) ∧

z > y

].

This new definition of the design concept is still related to the space in a very direct manner. Now, we consider the definition in $\mathbf{L_d}$ of properties that have a more abstract character. Suppose that we want to say that the floor of the bedroom is made of parquet, the floor of the living-room is made of marble, and that the price of the bedroom floor must be lower than the price of the living-room floor. If the prices per unit of both materials are known, then this concept can be expressed as

```
(17)        λx.[   design_object(x) ∧
                   house(x) ∧
                   parallel(wall_1, wall_3) ∧
                   horizontal(wall_4) ∧
                   in(end_of(wall_4), x) ∧
                   y = area_of(bedroom) ∧
                   z = area_of(living_room) ∧
                   floor_of(bedroom, parquet) ∧
                   floor_of(living-room, marble) ∧
                   $_per_unit(parquet, p1) ∧
                   $_per_unit(marble, p2) ∧
                   $_bedroom = y times p1 ∧
                   $_living_room = z times p2 ∧
                   $_living_room > $_bedroom
            ].
```

In this more complex concept, spatial properties and other abstract requirements that are expressed through language are represented in an integrated fashion. This feature is highly desirable for the definition of intelligent CAD systems when they are thought of within the context of integrated manufacturing environments (Gandhi 1989, Pratt 1988, Cunningham 1988, Dixon 1986, Requicha 1988, Opas 1988). This facility is also useful in architectural design systems (Tweed 1989).

This process of specifying the design object can go on for ever. For any set of conditions that we specify in drawings there are many moves to make in pursuing the final design. We can add, delete, update the properties and relations of the individuals, and we can redefine the design concepts limiting the space of possible designs; however, at some point in the process, we will have to make subjective choice among the possibilities given for us at that state.

Design concepts guide the designer through the whole design process; they are not known in advance by designers, and they are discovered throughout the process in the same way that the design object is. Of course, in many design domains, a detailed definition of the concept can be given on the basis of previous design experiences, and the design cycle does not have to be started from scratch. The important point is that when a design concept has been clarified it actively guides the designer along the design process.

7.3. The Notion of Design Intention.

The design concepts presented so far are 'passive' because they are used for verifying whether some particular drawing satisfies some explicit set of conditions, and the designer is responsible for taking every initiative for updating the representation in the design process. Now, we consider the definition of 'active' design concepts in which the system takes the initiative and starts a cycle of design.

For our first approximation of the notion of design intention, consider that the dynamic behaviour of the system can be specified by expressing simple programs in the course of the graphics and linguistic interaction like, for instance, the change rule presented in Chapter 6, in Figure 6.12. Although the interactive graphical and linguistic definition of these programs seems natural and powerful, such a scheme is in fact, very similar to traditional AI rule-production systems. Such a kind of rules produce local transformations in drawings, but they might have undesirable consequences at the global level. Here, we explore an additional layer of functionality in which such programs are inferred and applied to by the system in the course of the design session. The purpose of this layer is to trace the global consequences of local transformations in the process of modifying a drawing. Next, we discussed some preliminary considerations for the automatic definition of such processes. The notions introduced below in this chapter are formalised in Chapter 8.

The design task that we consider here is the geometric modeling of 2-dimensional wire-frame drawings that are common in architecture and other kinds of design domains. The problem in this design task is how to specify and produce a change of the graphical representation of a design object, given a set of overt geometrical and topological constraints. This design domain has been subject of much research (Arbab 1989, Borning 1981, Kimura 1986, Suzuki 1988, Light 1982, Rossignac 1989, Santana 1987). In these approaches, human designers are supposed to understand some predefined mathematical model --the geometric model-- and in such a context, express the parameters for the required modification. Not surprisingly, traditional geometric modellers betray in some situations the expectations of human designers. Much work and effort is spent in defining geometric modellers that behave better in relation to human expectations. However, none of the traditional methods face directly the fact that geometric modellers are rather models of a kind of human design inference that happen to be based on geometry. But of course, these models are not complete and they only explain a human design inference when the object that they produce is according to the expectations of the human designer. Here, we can raise an interesting question: how can we know which is in fact the object intended by the human designer in this sort of design task?

In fact, if a geometric modeller behaves as every human being would have expected in some design task, the process by which the system would come to the solution of a specific design problem would be similar to the process by which such a problem is solved by the human designer. If such a geometric modeller is ever built, its definition would give an explanation for the corresponding human behaviour. We have to consider, however, that the product of a design task depends not only on 'the design skills', but also on the state of knowledge of the human designer at the time such a task is performed. Differents states of knowledge would determine different models for the solution of a specific design problem, in the same way that different human designers can produce different solutions for the same design problem. Then, what has to be explained is an underlying design faculty: the design process itself and its relation to the contingent beliefs of human designers. When this faculty is exercised, a human design inference is performed and a design object is produced. Although a solution of this problem is far from being found, facing head-on the fact that geometric modellers are models of a kind of human inference might shed some light on the deep nature of this very hard problem.

The method that is presented below is semantically motivated. Here, we take an explicit account of the current design concept and intention, and these definitions determine the model upon which the solution is found. There is of course a basic layer of geometry that the human user is supposed to be aware of. However, the geometry is used in a non-deterministic process, and several solutions for a particular design task can be produced. It is worth pointing out that there is no intention to make any psychological claim in relation of how human designers come to find design objects in this task. However, I shall show that the process introduced below in this chapter and in Chapter 8 produces objects within the extension of the current design concept, and also provides a causal link between the original object, the intention manifested by the human designer, and the new design object. The method is also able to determine that some problems have no solution. Furthermore, it is possible to produce a natural language and graphical interactive explanation of the process by which the solution of some particular problem is found, along the lines of the other conversations in GRAFLOG.

In Figure 7.8.a our well known house is shown again. Suppose that the current design concept for the house is as shown in (12) and (13) above. We might like to try a different object and to produce a new graphical representation. For instance, by pointing out one of vertices of the polygon standing for the house, and dragging it to a new location, as shown in Figure 7.8.b.

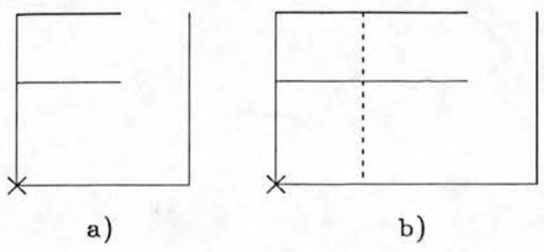

FIGURE 7.8

We will refer to this kind of desire as a *design intention*. It is worth noticing that both houses in Figure 7.8.a and 7.8.b are within the extension of the design concept that was introduced above. They satisfy both the graphical and logical conditions stated when the concept was made explicit through the graphical and logical interactive dialogue.

7.3.1. Intended and Free individuals in Design Intentions.

The design intention has as a purpose to select an alternative description of the design object, but within the current design concept: the model that is determined by the current state of the data-bases of the system. To achieve the new representation, we could erase the whole drawing and start all over again, but that would be a very uneconomic behaviour, and to modify the current representation would be a much better strategy. In fact, that is what a human designer would do in the circumstances: he or she would erase the changing constituents of the graphical representation and then draw them again in the new spatial positions.

The question is now, which objects have to be modified? Here, the human designer has to make a decision and some individuals would have to remain as they were before the modification, and some others will be changed. In fact, from comparing Figures 7.8.a and 7.8.b we can see that the only individual whose graphical properties remained the same before and after the process of change is *wall_1*.[2] If the human designer modifies the house in this way, we can see that besides selecting an alternative instance in the extension of the design concept, he or she wants to see some new instance of the designing object in which the individual that remains fixed -- like, for instance, *wall_1*-- is the reference for the whole transformation. This is a very important consideration because it imposes a very strong restriction among the set of possible objects that satisfy the design intention and are within the extension of the design concept.[3] Let us name this individual --the reference of the transformation-- *the intended individual* because the transformation is performed with the *intention* of finding the new drawing with respect to this --intended-- individual. In fact, there might be a set of intended individuals and

[2] The identifiers for the walls of the house are shown in Figure 6.5. This figure was repeated above in this chapter for clarity.

[3] In the implementation, several strategies for defining 'invariant' individuals can be considered. For instance, to point out the individuals that are invariant in some context before the design intention is expressed.

modifications would be further restricted.

Individuals that are modified in pursuing the design object will be referred to as *free individuals*, because they can be updated in the modification process. It is clear that modifying one or another free individual in transformations would produce different new designing objects and they will be generated through different sequences of actions. Here, we have to decide which free individuals have to be modified, and also what is the order in which the modifications will be made. In Figure 7.9 two possible modifications for the house are shown. The intended individual is *wall_1* as before.

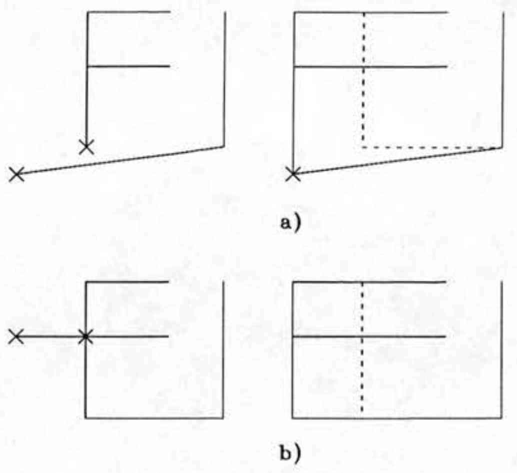

FIGURE 7.9

In Figure 7.9.a, the end of *wall_2* was moved to the bottom-left, and in Figure 7.9.b the origin of *wall_4* was moved to the left. The point to see is that for some intended individual, the choice of the individual that is considered free for the initial modification, and the particular way it is updated, will eventually produce a highly restricted set of possible configurations, and in some situations a single particular one. Furthermore, the individual that the human designer chooses to modify determines somehow the whole design intention and it is considered an invariant for the transformations applied by the system. A very important consequence of this last consideration is that the initial design object cannot be reached by a cyclic sequence of transformations.

Conceptually, the human design intention and the system design intention are different, though the later is subordinated to the former. When the human designer expresses a design intention he or she selects two sets of individuals: the references for the whole transformation, that are not subject to modification, and the set of individuals that he or she modifies for expressing the intention. From the point of view of the system, however, the individuals in these two sets are invariant they cannot be modified. The individuals that the system is allowed to modify for solving a problem are included in the difference set between the set of all individuals and the invariants in its view.

7.3.2. Degrees of Freedom.

In this version of GRAFLOG there is just one kind of basic symbol: the line. As was shown, dots and polygons are determined by the graphical structure **D**. It is clear that a dot cannot be altered without changing the properties of the line that it defines, and lines and polygons can be modified through updating their constituent dots. Now, we define the degree of freedom as the number of terms of sort *dot* that can be altered in a single transformation. This definition is introduced with the purpose to impose a constraint in the complexity of transformations for satisfying design intentions. The lower the degree of freedom, the lower the transformation complexity. In our theory, all transformation functions that are deduced by the design interpreter have degree of freedom one. For the definition of this kind of transformations, the functions *move_origin* and *move_end* will be used. Transformation functions have the form of the change rule defined in Section 6.5, and they are applied by the design interpreter *ADP*.

Through a sequence of modifications, the graphical representation that satisfies the design concept can be found. In Figure 7.10 a sequence of transformation for achieving the design object of Figure 7.9.a is shown.

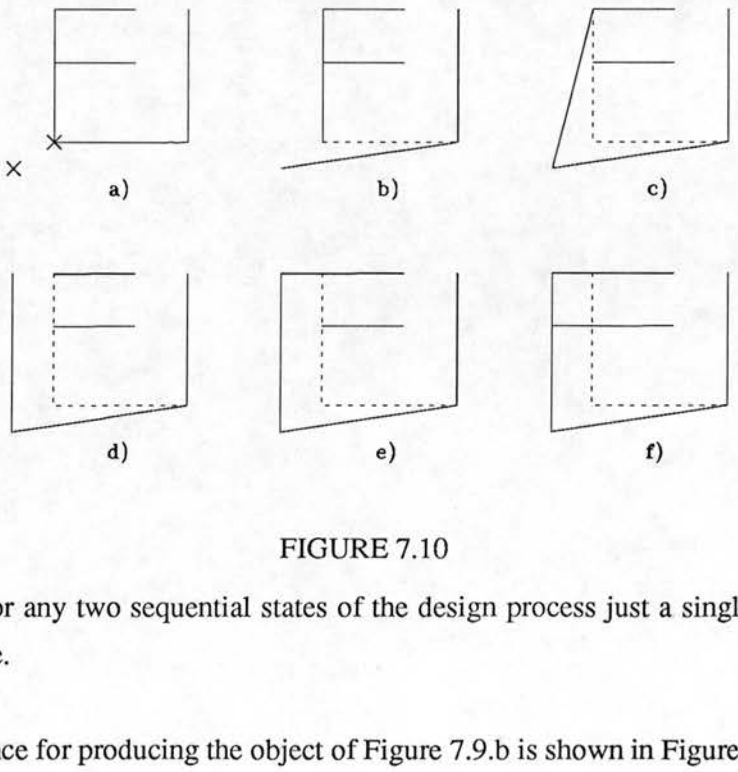

FIGURE 7.10

As we can see, for any two sequential states of the design process just a single dot of a line is allowed to change.

A possible sequence for producing the object of Figure 7.9.b is shown in Figure 7.11.

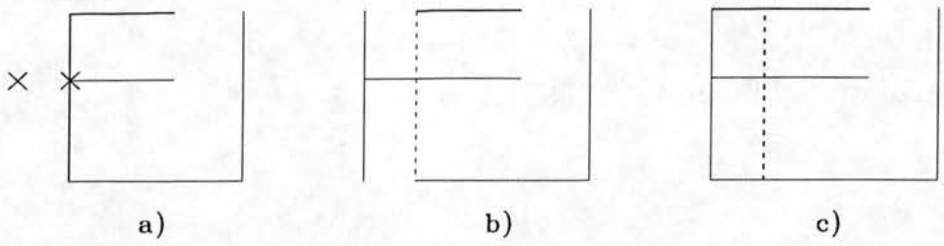

FIGURE 7.11

However, from Figure 7.11.a to 7.11.b, two extreme points of a line are updated, and from Figure 7.11.b to 7.11.c two points, each one of a different line, are updated as well. These transformations have a degree of freedom of *two*.

7.3.3. Propagation of Design Intentions.

In the process of modifying some design object, the sequence of transformations for achieving the new instance of the design object is determined by the intended and free individuals expressed in the main design intention. Consider again the design intention illustrated in Figure 7.10. In this process, the human designer choses *wall_1* as the intended individual, and *wall_2* as the free individual for starting the transformation sequence, and the drawing in Figure 7.10.b is produced. This second drawing cannot be the new design object according to the intention expressed because it is not within the extension of the current design concept. The drawing does not satisfy the design concept because the condition

(15) $\exists v_2 . v_2 =$ **e_joint_at(wall_3, wall_2).**

is not satisfied in Figure 7.10.b. The transformation from Figure 7.10.b to 7.10.c seems to be fully determined because the only condition that is not satisfied by the drawing is that *wall_3* does not join *wall_2* at v_2. However the next transformation from Figure 7.10.c to Figure 7.10.d is not fully determined since there are two independent conditions that the design object must satisfy to be within the extension of the current design concept. These conditions are

(18) **(a)** $\exists v_3 . v_3 =$ e_join_at(wall_4, wall_3).

(b) parallel(wall_1, wall_3).

The sequence shown in Figure 7.10 is produced with the intention of satisfying the parallel condition in 18.b but if condition in 18.a is satisfied first by the interpreter, an alternative sequence of transformations is produced, as shown in Figure 7.12.

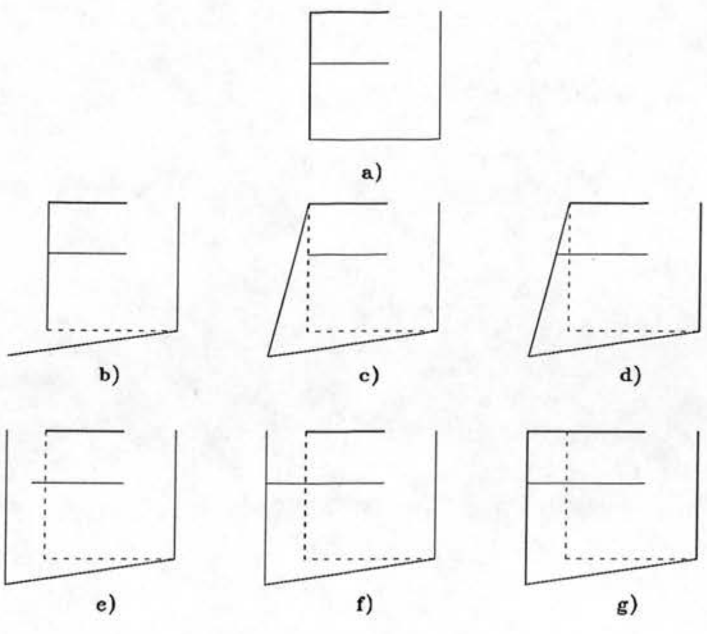

FIGURE 7.12

As can be seen, the two different sequences imply a different number of state transitions. This process in in general non-deterministic: different sequences can produced different instances of the design object, sequences can have different lengths, and some sequences might even be cyclic. Finding out which sequence is intended by the human designer for an arbitrary design concept and an arbitrary design object is a very hard problem. In fact, the human designer might neither be consciously aware of the shape of the desired object nor of the sequence of changes by which it is reached. The intention can be manifested in order to satisfy a local desire, and yet it has consequences at the level of the whole drawing.

Next, the notion of subordinated system design intention, as opposite to the main design intention that is specified by the human-designer, is introduced. A subordinated design intention is defined as the intention to modify a drawing from any disequilibrium state to the next state in a transformation sequence. In other words, once the main design intention has been expressed by the human designer, a sequence of subordinate intentions have to be defined for mapping the current state to the next until equilibrium is restored in the system. In the same lines, we define a subordinate design concept as the condition --or set of conditions-- that are specified in the current design concept but that are not satisfied by the graphical representation in some disequilibrium state. In the definition of subordinated design intentions, there is always an intended individual and one free individual. We define the intended individual for a subordinated design concept as **the individual that was modified in the transformation function that produced the current disequilibrium state**.

Consider again the process shown in Figure 7.10. When the main design intention is expressed, as shown in the transition from Figure 7.10.a to Figure 7.10.b, the free individual in the human design intention is *wall_2*. The next transformation --from Figure 7.10.b to 7.11.c-- is produced by a subordinated design intention in which the intended individual is *wall_2*, and the free individual is *wall_3*. This last individual is the one updated for satisfying the condition --the subordinated design concept-- that was not satisfied by the drawing in the disequilibrium state in Figure 7.10.b. Once this subordinated design concept is satisfied and the next state in the sequence is produced --Figure 7.10.c-- a new local design intention has to be expressed.

Subordinate design concepts depend on local conditions of every disequilibrium state in the design process. In their satisfaction process there must be an individual that is considered the reference for the local change --the subordinated intended individual; there is also some degree of choice that depends on the possible free individuals that can be updated, and a sense of direction which is determined by the beliefs that the system holds as necessarily true when the design intention was manifested by the human designer. These beliefs are represented in the main design concept and in *NLKB*.

Now, we define the basic heuristics for guiding the search for new design objects in this theory of design. We refer to this rule as **intention propagation**, and it is defined as follows: if an individual that was free in a disequilibrium state was updated by a subordinated design intention then it becomes the intended individual in the next subordinate design intention. This notion will be formalised below when the function Ω is introduced.

The definition of the intention propagation process is, however, not yet completed and some additional considerations are given below. In fact, the intention propagation process --as it stands at the moment-- is unable to produce the sequences in Figures 7.10 and 7.12. In the transition from Figure 7.10.c to 7.10.d the intended individual was *wall_3*, but it itself was the one updated in the transformation in order to satisfy the subordinated design concept *parallel(wall_1, wall_3)*. However, the reference for such a transformation has to be *wall_1*, and there is no way, according to the current definition of the intention propagation, for *wall_1* to be the intended individual for such a subordinate design intention.

Consider now the transition from Figure 7.12.d to 7.12.e in which *wall_3* is modified. This transformation is applied for satisfying the subordinate design concept *parallel(wall_1, wall_3)* as well. However, the individual that was modified in the previous transformation in the same sequence --from Figure 7.12.c to 7.12.d-- was *wall_4*, and according to the definition of intention propagation, it should be the reference for the next transformation. In this case, *wall_4* is not even referred to by the current subordinate design concept.

Notice that *wall_1* is the reference in both of the problematic examples. Notice as well that it is the intended individual of the main design intention. Now, we extend the definition of intention propagation by considering the satisfaction of subordinate design concepts in relation to the main intended individual; more generally, in relation to the individuals that are invariant in the human specified main design intention. In the two cases discussed above, the intended individual of a subordinate design concept is one of the invariants in the main design intention. We define that the intended individual of a subordinate design intention can be one individual in the set of invariants in the main design intention as well.

It is worth noticing that the intended individual in the main design intention might be left unspecified. In such a situation, the only invariant individual in a sequence of transformations would be the one modified by the human designer when the main design intention is expressed. Of course, the number of possible configuration produced by the process can be very large, but in some problems this might be a plausible situation. In Figure 7.13 this situation is illustrated. The same sequence of transformations is shown, but as can be appreciated in the transformation from Figure 7.13.d to 7.13.e, the intended individual is *wall_3* and the free individual for satisfying the parallelism condition is *wall_1*. If *wall_1* is not the intended individual of the main design intention, and if neither *wall_1* or *wall_3* is constrained to be vertical, then the new intended design object could be the one shown in Figure 7.13.e.

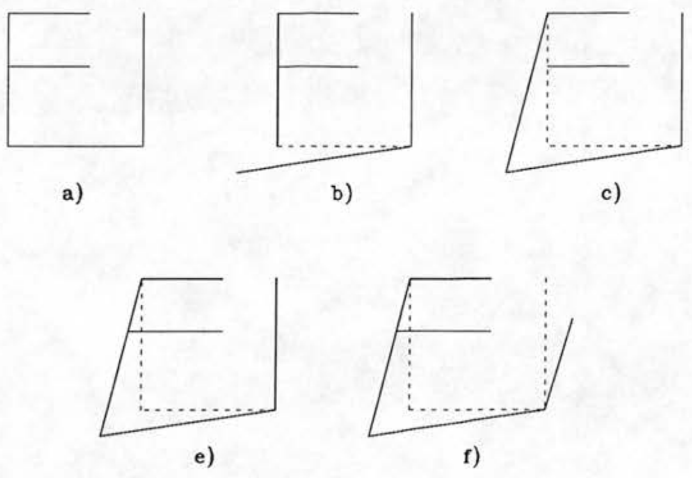

FIGURE 7.13

An additional important consideration is that intention propagation blocks the production of some sequences that would arrive otherwise at objects within the extension of the main design concept. For instance, the sequence is Figure 7.14 is not produced. This is so because the object in Figure 7.14.e would not be house intended by the human-designer in that context. That design object would be produced from the house in Figure 7.14.a, but by a different main design intention.

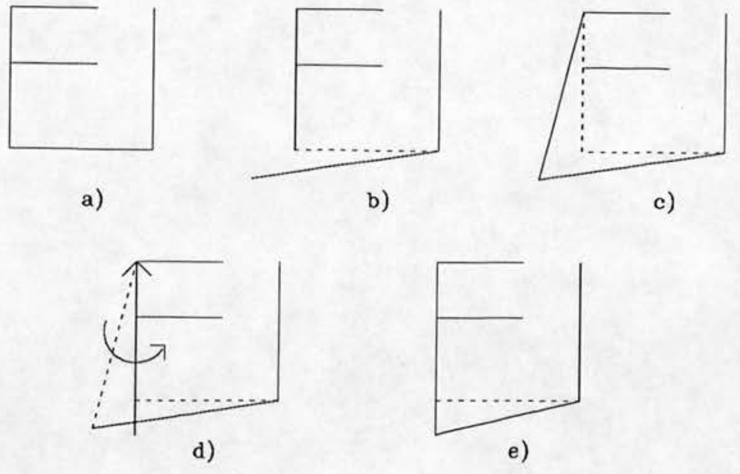

FIGURE 7.14

Another related consideration is that if both intended and free individuals that are referred to by a subordinated design concept are in the list of invariants of the main design intention, then such a subordinated design intention cannot be satisfied.

7.3.4. Parallel Satisfaction of Design Intentions.

Subordinate design intentions can be satisfied by simultaneous transformations. Consider the transitions from Figure 7.10.d to 7.10.e and from 7.10.e to 7.10.f. Both of these transitions have the same intended individual, and the conditions triggering both transitions are independent. Intuitively, they might be performed in parallel. Furthermore, in that sequence it is important to consider whether the intended individual for the last transformation can be determined. In fact, in Figure 7.10.e *wall_3* is neither one of the invariants for the main design intention, nor the individual that was free in the previous transformation, and the whole sequence in Figure 7.10 cannot be produced by intention propagation. Now, consider that *wall_3* is the current intended individual in Figure 7.10.d and both of the conditions

(17) **(a)** $\exists v_3 . v_3 =$ **joint_at(wall_4, wall_3).**
(b) $\exists v_4 . v_4 =$ **joint_at(wall_5, wall_3).**

can be satisfied by a parallel pair of transformations that have *wall_3* as the intended individual, but involve the modification of different free individuals: *wall_4* and *wall_5*. If the condition of parallel satisfaction of design intentions is not considered, there is the risk that the referent for the main design intention is lost, as it happens in the sequence shown in Figure 7.10. However, when the intended individual for several independent transformations is the same, and the free individuals of these changes are different, a number of conditions can be satisfied simultaneously, as shown in the transition from Figure 7.15.d to 7.15.e.

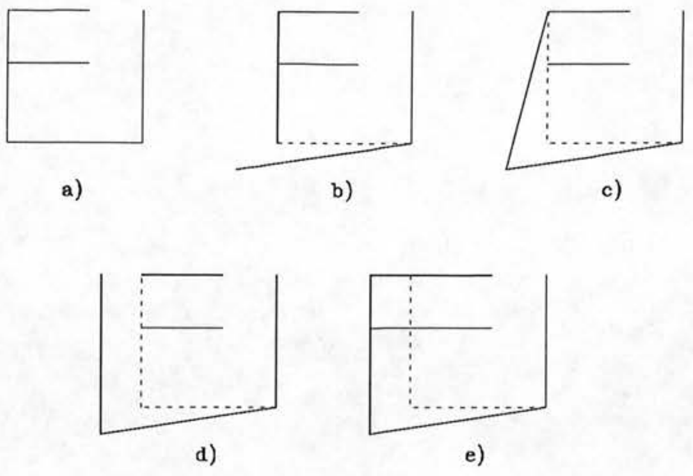

FIGURE 7.15

In the same way that the application of parallel transformations reduces the sequence in Figure 7.10 to the sequence in 7.15, the sequence in Figure 7.12 is reduced to 7.16.

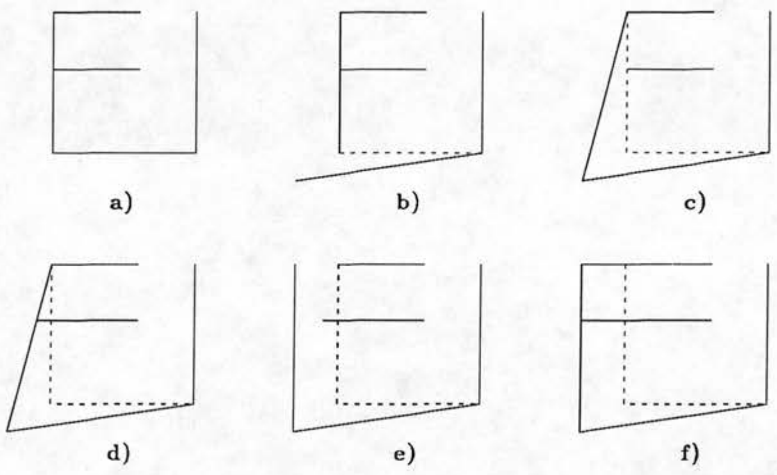

FIGURE 7.16

7.3.5. The Satisfaction Rules and the Models of Design.

The satisfaction of local conditions is a non-determined process. Given a design condition that has to be satisfied in relation to some given pair of intended and free individuals, there might be more than way, and in many cases an infinite number of ways, to satisfy such a condition in terms of such a pair of individuals. For every problem specified by a set of constraints where the references for the transformations are known there might be an infinite number of 'mathematically sound' solutions, but just a very few of them might be considered 'intuitive solutions'. Although specific design conditions are defined in terms of local parameters, the rule defining the actual mode of satisfaction might be contextually determined. Furthermore, different design contexts might demand different rules for satisfying the same design condition.

Consider for instance the satisfaction of the *parallel(wall_1 wall_3)* condition from figure 7.15.c to 7.15.d, or the satisfaction of the same condition in Figure 7.13. Now, how can we tell what rule such transformations obey? Why was the length of the wall that was rotated altered in these transformations? In fact, the number of modes for satisfying the condition in these examples is infinite, and yet the transformation mode that was chosen might or might not seem to some readers 'natural and intuitive enough'. I myself find quite difficult to explain convincingly why this example was chosen in the first place, and what rule I followed when I satisfied the parallel condition in that way in this particular example. To know which rule we follow is not a trivial matter. According to Wittgenstein (Wittgenstein 1963, p. 38),

> [82] What do I call 'the rule by which he proceeds'?--the hypothesis that satisfactorily describes his use of words, which we observe; or the rule which he looks up when he uses signs; or the one which he gives us in reply if we ask him what his rule is?-- But what if observation does not enable us to see any clear rule, and the question brings none to light? --For he did indeed give a definition when I asked him what he understood by "N", but he was prepared to withdraw and alter it.-- So how am I to determine the rule according to which he is playing? He does not know himself. --Or, to ask a better question: What meaning is the expression "the rule by which he proceeds" supposed to have left here?

If we look back to a particular behaviour, we can describe how it actually happen. If the same things happen often enough, we can say that there is a rule, but if the behaviour is to be anticipated, we can only say that what follows is embedded in a process.

Here, rather than following all possible consequences of the different satisfaction modes for a design intention, we define a specialised procedure --a heuristic method-- for satisfying basic subordinated design concepts that are considered in the intention satisfaction process. The heuristic rules are presented in detail in Chapter 8, and they produce the sequences illustrated in Figures 7.15 and 7.16. The way these procedures are defined determines the 'abilities' of the system for the solution of problems in some design domain. For the definition of these methods, we can consider local and global properties of the structure in some particular design state. We can also consider the design states that would be reached if a condition is satisfied in some or another way. These rules might be contextually determined in relation to specific design tasks. Furthermore, the fact that these transformations are defined in some or another way in geometric modellers determines an implicit model of the human design intuitions in the design domain that is targeted by the system's designer. And as was mentioned, these models are usually incomplete in as much as they behave according to expectations in some situations, but betrayed our design intuitions in others. It is rather paradoxical that when we are engaged in the definition and implementation of a geometric modeller, we cannot avoid the fact that we are also developing an implicit model of human design intuitions, which is as incomplete as our grasp of the human design activity itself is.

The set of heuristics procedures presented in Chapter 8 has been defined having architectural drawings in mind --or rather the drawing of a house with two rooms and a corridor in mind. The definition of such collection of procedures for some application domain is itself a topic for a complete research project. For architectural drawings Szalapaj (Szalapaj 1988) has developed a powerful and simple set of rules based upon local properties of drawings. In that approach architectural structures are tied to construction lines, and the relations between construction lines dictates the sort of change produced by a transformation. Another interesting example of the use of this kind of heuristics is developed by Santana-Sepúlveda in a program called SIGRASITT for the interactive definition of pylons (Santana 1987).

Here, we conclude this chapter in the notions of design concept and intention in GRAFLOG. The definitions of design concept and design object, and their relation to the design cycle were presented. An example of the definition of these two notions was developed in detail. Then, the notion of design intention was introduced, and the considerations for the interpretation and satisfaction of design intention expressed by human designers in the 2-dimensional wire-frame design domain were discussed. In the next chapter, we introduce a function by which these intention are satisfied in the course of a graphics and linguistic interactive design session, and a set of heuristic methods that are used for the solution of design problems in our very specific design domain.

Chapter 8

A Function for the Satisfaction of Design Intentions

In this chapter a function for the satisfaction of design intentions is presented. This function corresponds, in the wire-frame drawings domain, to the function *change* that was presented in Section 5.3 for following up the global consequences of local transformations in the blocks-world domain. This function determines the consequences of modifying wire-frame drawings in normal graphical interaction, and also the consequences of applying domain specific design rules by GRAFLOG's interpreter itself.

In Section 8.1 the intention satisfaction function Ω is formally defined. In Section 8.2, a procedure for its evaluation is presented. In Section 8.3, the transformation functions that are applied by the system in the satisfaction of design intentions are illustrated, and the assumptions taken into account for their definition are discussed. In Section 8.4 some pragmatic considerations for evaluating Ω are illustrated. In Section 8.5, a history mechanism for explaining the methods by which GRAFLOG's interpreter comes to the solution of problems is presented. Finally, the characteristics of the system's integrated design environment are highlighted and discussed in Section 8.6.

8.1. Definition of the Function Ω.

We define the intention satisfaction function Ω as a function from design intentions to truth values. If an intention can be satisfied in the current graphical context, and with the current state of knowledge of the design interpreter $\Omega = 1$; otherwise, $\Omega = 0$. An instance of a design intention is represented by the ordered tuple $<P, x, \Phi, \Upsilon>$.

In the intention satisfaction process, design intentions are expressed and satisfied at design states that are indexed by an ordered tuple, namely $<i, j>$. The pair $<i, j>$ indexes a modal state in the interpretation of the language $\mathbf{L_d}$. A design intention that is manifested at some state $<i, j>$ is represented in our notation as $\Omega_{<i, j>}$. The state in which an intention is manifested will be sometimes referred to as the state $\Omega_{<i, j>}$ as well. This notation indicates that states visited by the intention satisfaction process are organised in a tree structure in which i stand for 'the level' or depth of computation and j 'the delta' or the j-son of its corresponding father state. When a design state is referred in relation to its sons, the index of the father state is, by convention, $<i, \_>$.

Now, we come to the description of the argument list of Ω: P stands for a set of terms of sort *dot* that were modified in state $<i-1, \_>$ --the father of the current state in the intention satisfaction process-- x stands for the design object in the current state $<i, j>$, Φ stands for a design concept, and Y stands for the set of terms of sort *line* that are invariant in the satisfaction of the current intention. In our notation, a design intention that is expressed at some state $<i, \_>$ with an overt set of parameters is expressed as

(1) $\Omega_{<i, \_>}[P, x, \Phi, Y]$.

The function Ω is defined as follows,

(1) IF $\Phi(x) = 1$ at state $<i, \_>$ then $\Omega_{<i, \_>}[P, x, \Phi, Y] = 1$ as well. This means that the design intention is satisfied in the current state $<i, \_>$.

(2) If $\Phi(x) = 0$ or undefined in the state $<i, j>$, and the chain of causal relations between design states that is established in the intention propagation process is broken, then $\Omega_{<i, j>}[P, x, \Phi, Y] = 0$. This condition means that some given design intention cannot be satisfied from the information that is available at the current state $<i, j>$.

(3) If neither of the previous conditions hold, then there is some state $\Omega_{<i+1, k>}$ that is produced by the intention propagation process and the value of the function Ω is the same for the states that are related in the process. In general, $\Omega_{<i+1, k>} = \Omega_{<i, j>}$.

We take the convention that the index of the state in which the intention is expressed by the human designer is $<0, 0>$. The human action causes a state transition to the state $<1, 0>$. For instance, consider that once the object in Figure 7.15.a is modified by the human designer for producing the object in Figure 7.15.b, the design intention that has to be satisfied by *ADP* is represented as follows,

(2) $\Omega_{<1, 0>}$**[{end_of(wall_2)}, house_1, Φ, {wall_1, wall_2}].**

where the graphical representation of the design object --*house_1*-- is defined in *g_db* as shown in Section 6.2, and the design concept Φ is as in (12) in Chapter 7. The set of invariants are given by definition: *wall_1* has to be explicitly pointed out before the intention is manifested,[1] and *wall_2* is the individual that was modified by the human designer. It should be clear that the value of $\Omega_{<1, 0>}$ cannot yet be known, because *house_1* does not refer in such a state.[2] In particular,

[1] An special interactive mode is defined for such a specification.

[2] Consider the conditions of reference in Section 6.3.

(3) $$\exists v_2 . v_2 = \textbf{e\_joint\_at(wall\_3, wall\_2)}.$$

denotes the error element of sort *bool* in the state $\Omega_{<1,\,0>}$.

Now, we introduce the indexed set Δ_{level} of conditions specified either in the definition of the current concept Φ or in the description list of the graphical object x in its corresponding entry in *g_db* which are not satisfied by *NLKB* or *g_db* at some state *<level, _>*.

(4) $$\Delta_{\mathbf{level}} = \{\delta_{<0,\,\alpha>}, \delta_{<1,\,\beta>}, ..., \delta_{<n,\,\gamma>}\}$$

Each term $\delta_{<j,\,\alpha>}$ in Δ_{level} is the j indexed set of subordinated design concepts that can be simultaneously satisfied in relation to some particular individual α. In our example,

(5) $$\Delta_1 = \{\delta_{<0,\,\mathbf{wall\_2}>}\}$$

where

(6) $$\delta_{<0,\,\mathbf{wall\_2}>} = \{\textbf{e\_joint\_at(wall\_3, wall\_2)}\}.$$

For the satisfaction of the only condition in $\delta_{<0,\,wall\_2>}$ in Δ_1 in Figure 7.15.b *wall_3* has to be modified in relation to the invariant *wall_2*. Such a transformation produces the drawing in Figure 7.15.c. The design object in this new state has no denotation, and in fact, there is a new design intention that is manifested in that state; such an intention is expressed as,

(7) $$\Omega_{<2,\,0>}[\{\textbf{origin\_of(wall\_3)}\}, \textbf{house\_1}, \Phi, \{\textbf{wall\_1, wall\_2}\}].$$

Note that $\Omega_{<2,\,0>}$ comes from satisfying $\Omega_{<1,\,\_>}$ in relation to the only subordinated design concept in Δ_1 --$\delta_{<0,\,wall\_2>}$. We can express this relation as

(8) $$\Omega_{<2,\,0>} \Leftarrow \Omega_{<1,\,\_>}$$

In general, design intentions of father and son states of the design process are related by satisfying the design intention manifested in the father in relation not to the main design intention, but rather to some condition that is not satisfied in the father state. The reference for this local transition comes from the intention propagation process. This relation can be expressed as,

(9) $$\Omega_{<i+1,\,j>} \Leftarrow \Omega_{<i,\,\_>}[\mathbf{P}, \mathbf{x}, \delta_{<j,\,\alpha>}, \{\alpha\}]$$

Now, consider that both $\Omega_{<i,\,\_>}$ and $\Omega_{<i+1,\,j>}$ have to be satisfied not only at the local level, but also in relation to the main intention. Next, we define the total design intention $\Omega^{+}_{<i,\,j>}$ as an ordered pair of the local and global design intentions that have to be satisfied at every state $<i, j>$ in the intention satisfaction process. The total design intention is represented as follows,

(10) $$\Omega^{+}_{<i,\,j>}[\Omega_{\mathbf{local}}, \Omega_{\mathbf{global}}].$$

Where *local* and *global* stand for the index pair $<i, j>$ and $<0, 0>$ respectively. The total design intention of two states that are directly related --father and son-- is recursively defined as follows,

(11) **IF** $\Phi(\mathbf{x}) = \mathbf{1}$ **THEN**

$$\Omega^{+}_{<i,\,j>}(\Omega_{\mathbf{local}}, \Omega_{\mathbf{global}}) = \mathbf{1},$$

```
ELSE IF it is known that $\Omega_{local}$ cannot be satisfied THEN
    $\Omega^{+}_{<i, j>}(\Omega_{local}, \Omega_{global}) = 0$,
ELSE FOR ALL $\delta_{<j, \alpha>}$ in $\Delta_i$
    $\Omega^{+}_{<i+1, j>}(\Omega_{<i+1, j>}, \Omega_{global}) = \Omega^{+}_{<i, \_>}(\Omega_{<i, \_>}[P, x, \delta_{<j, \alpha>}, \{\alpha\}], \Omega_{global})$.
```

The sentence *it is known that* Ω *cannot be satisfied* in (11) means that although there might be some conditions in the set Δ_i at the state $<i, \_>$, the system might detect that their satisfaction involves the change of an invariant individual, or simply, there might not be a transformation function that is known to the system by which the required change can be produced. It should be clear that the design intention will be satisfied at some state in which the design object denotes and satisfies all the conditions expressed in the design concept. In fact, the main design intention is satisfied at some state $<n, m>$ in which Δ_m is an empty set, and $[[\Phi(x)]] = 1$.

8.2. A Procedure for the satisfaction of the function Ω^{+}.

In this section, we define the procedure Ω for the satisfaction of a design intention as

(12) $\Omega_{<i, \_>}[P, x, \Phi, Y]$.

where the local and global parameters are defined as before. The computational procedure is given from (i) to (iv) as follows,

```
(13) (i)   IF $\Phi(x) = 1$ THEN
               $\Omega^{+}_{<i, \_>} = 1$ and x is the new design object
     (ii)  IF $\Phi(x) = 0$ OR undefined THEN
               compute $\Delta_i = \{\delta_{<0, \alpha>}, \delta_{<1, \beta>}, ..., \delta_{<n, \gamma>}\}$
               delete all unsatisfiable $\delta_{<j, \alpha>}$ from $\Delta_i$
     (iii) IF $\Delta_i = \varnothing$ THEN
               $\Omega^{+}_{<i, \_>} = 0$
     (iv)  FOR ALL satisfiable $\delta_{<j, \alpha>}$ in $\Delta_i$
               $\Omega^{+}_{<i, \_>} = \Omega^{+}_{<i+1, 0>} \vee \Omega^{+}_{<i+1, 1>} \vee ,..., \vee \Omega^{+}_{<i+1, j>}$
```

Now, we define the procedure ξ that computes the set of basic lines that are referred to by some expression in L_d. For instance,

(14) **$\xi(\Phi)$ = {wall_1, wall_2, wall_3, wall_4, wall_5}.**
ξ(e_joint_at(wall_3, wall_2)) = {wall_3, wall_2}.

For the computation of (ii) in (13) we consider the sets Γ, **I**, and **F** defined from (15) to (17) as follows,

(15) Γ is set of individuals of sort line that are referred to in the design concept Φ such that $\Gamma = \xi(\Phi)$, as shown in (14).

(16) *I* is the set of individuals that are invariant for the satisfaction of the design intention at the current state. These are either the individuals that come from the intention propagation

process or the invariants for the whole transformation. For computing the set I consider that every expression of sort *dot* in the set P refers to a dot that was modified in the transformation that produced the current state. We define the set L of lines modified in such a transformation such that for all $p \in P$, $\xi(p) \in L$. The set of invariants in the current design state is $I = \Upsilon \cup L$.

For instance, consider Figure 7.15.b in which $P = \{end\_of(wall\_2)\}$; then, $\xi(end\_of(wall\_2)) = \{wall\_2\}$ and $L = \{wall\_2\}$. Consider as well the set of invariants for the whole design intention $\Upsilon = \{wall\_1, wall\_2\}$, and $I = \{wall\_1, wall\_2\}$ in the state $<1, 0>$.

(17) F is the set of free individuals for the current design intention. This set includes all the lines that can be modified for achieving the next state. F is the set of lines referred to by the concept Φ without the invariants in the state: $F = \Gamma - I$. Note that $I \cap F = \varnothing$, and $F = \{wall\_3, wall\_4, wall\_5\}$ in Figure 7.15.b.

The next step in the computation of (ii) in (13) is to find the sets in $\Delta_{level} = \{\delta_{<0, \alpha>}, \delta_{<1, \beta>}, ..., \delta_{<n, \gamma>}\}$. In general, a subordinated design concept in $\delta_{<j, \alpha>}$ can be satisfied if it refers to one intended individual α in I and to some free individual β in F. However, if the satisfaction of a condition in some $\delta_{<j, \alpha>}$ is blocked by additional constraints in the objects that are referred to in the design concept, such a condition is deleted from this set. This would happen, for instance, if the length or the angle of some line that is considered the free individual in the satisfaction of some subordinated design concepts has been assigned a constant value in *NLKB*.

Now, consider that the cardinality of the set Δ_i determines the number of next states in $level = i + 1$ that the satisfaction of the current subordinated design intention produces. Consider as well that if the cardinality of a set $\delta_{<j, \alpha>}$ in Δ_i is n then the state $\Omega_{<i+1, j>}$ is reached by applying n simultaneous transformation functions of one degree of freedom upon the graphical configuration in the state $\Omega_{<i, \_>}$.

We illustrate the more representative cases of a father and son states relation in the intention propagation process:

(1) One son-state one-transformation transition: in Figures 7.15.b and 7.16.b there is only one member in the set Δ_1, namely $\delta_{<0, wall\_2>} = \{\text{e\_joint\_at(wall\_3, wall\_2)}\}$ whose cardinality is also one. Then, the graphical configurations in Figures 7.15.c and 7.16.c are all the same, and are reached by a single transformation of one degree of freedom.

(2) Several son-states reached by single transformations: consider state $level = 2$ in Figures 7.15.c and 7.16.c. The set

$$\Delta_2 = \{\delta_{<0, \text{wall\_1}>}, \delta_{<1, \text{wall\_3}>}\}.$$

where,

$$\delta_{<0,\ \mathrm{wall\_3}>} = \{\mathrm{t\_joins\_at(wall\_4,\ wall\_3)}\}.$$
$$\delta_{<1,\ \mathrm{wall\_1}>} = \{\mathrm{parallel(wall\_1,\ wall\_3)}\}.$$

These two transformations determine two different sequences for the intention propagation process. The referents *wall_1* and *wall_3* come from the set of invariants Γ and from the intention propagation process respectively.

(3) One son-state reached by *n* parallel transformations: consider the state *level = 3* in Figure 7.15.d. The set

$$\Delta_3 = \{\delta_{<0,\ \mathrm{wall\_3}>}\}.$$

where,

$$\delta_{<0,\ \mathrm{wall\_3}>} = \{\mathrm{t\_joins\_at(wall\_4,\ wall\_3),\ e\_join\_at(wall\_3,\ wall\_5)}\}.$$

Here, *wall_3* is the referent for the satisfaction of both of the conditions, and the two free individuals in these transformations *wall_4* and *wall_5* are different.

(4) The case of several son-states reached by a number of parallel transformations is analysed along the same lines.

From these conditions and the examples in Figures 7.15 and 7.16 we can see that the design object produced by both of the transformation sequences is the same. In fact, the sequences of local design intention are partially ordered in a natural lattice structure, as shown in Figure 8.1.

$$\begin{tikzcd}
& \Omega_{<0,0>} \arrow[d, dash] & \\
& \Omega_{<1,0>} \arrow[d, dash] & \\
& \Omega_{<2,0>} \arrow[dl, dash] \arrow[dr, dash] & \\
\Omega_{<3,0>} \arrow[d, dash] & & \Omega_{<3,1>} \arrow[ddl, dash] \\
\Omega_{<4,0>} \arrow[dr, dash] & & \\
& \Omega_{<5,0> = <4,0>} &
\end{tikzcd}$$

FIGURE 8.1

The design object denoted in both of the states $\Omega_{<5,\,0>}$ in the left branch and $\Omega_{<4,\,0>}$ in the right branch is in fact the same instance of *house_1*. For more complex problems, there might be several solutions, and also several sequences of transformations from which these objects are reached. An issue that has not yet been explored is how this lattice can be used for optimising the search space of the intention propagation process.

8.3. The Transformation functions.

The intention propagation process states --in a non-deterministic way-- the conditions that have to be satisfied in the interpretation of a design intention and the production of a design object. However, the process does not define the mode of satisfaction of such conditions. In fact, from a geometrical and topological point of view, there might be a very large number --in many cases infinite-- of ways to satisfy a design constraint. The definition of the design intention Ω and its corresponding computational procedure tell us **what** has to be computed for satisfying a design intention; however, neither the function definition nor its associated procedure tell us **how** such computations are performed. We could consider the definition of Ω as a specification of what has to be computed to satisfy a design intention, in order to abstract over the implementational details. The 'implementational knowledge' of how to satisfy a design condition could be encoded as some or other algorithm. However, this would be the case just if the satisfaction of a design condition were a fully determined process; but the 'know-how' knowledge in this case is not a simple implementational detail: it is rather a model of what we --the people who design this kind of systems-- believe that a human designer would do in order to satisfy certain conditions in some given context.

Now, we come to the definition of our set of heuristic procedures. It is worth emphasising that there is no intention to claim that the methods are related to some human psychological process, nor that these heuristic rules are 'good' for satisfying design intentions in architectural drawings. The purpose of this section is rather to illustrate the way these methods are defined and used in the intention propagation process. The transformation functions presented in this section are defined in a similar manner to the change rule in Section 6.5. For the definition of these transformation rules, we assume that whenever there is an incompletely determined transformation, the system draws a pair of construction lines in the direction of the overt lines involved in the transformation. Then, the free individual is modified in relation to the intended individual and according to the intersections determined by such a pair of construction lines.

In the definition of these relations we consider two default assumptions.

(1) Projective assumption: if the satisfaction of a subordinated design concept determines that a line has to be modified to achieve certain mode of intersection, the angle of such a line remains the same in the transformation, if possible.

(2) Rotational reference assumption: if the satisfaction of a subordinated design concept requires that a line should be rotated, the pivot for such a rotation is either the current origin or end extreme of the line to be rotated, if possible.

Although these default rules reduce the space of possible solutions, there are many situations in which these criteria cannot be followed, and the human-designer must have learned the idiosyncratic procedures that the system has been endowed with.

In this version of GRAFLOG we only consider the satisfaction of conditions expressed as terms of rank *line line, dot*, with the exception of *int_mm* whose satisfaction, if required, would have to be defined by the user in relation to specific design tasks. Constrains defined in terms of expressions of sort *bool* are verified in order the design concept is permitted. However, the satisfaction of such a kind of constraints has to be determined by a direct manipulation on the graphical objects, or by a task specific transformation rule defined by the user. For instance, by change rules that are interactively defined along the lines of Section 6.5.

The satisfaction of some basic terms of rank *line line, dot* in $\mathbf{L_d}$, like *int_oo, int_oe, int_eo* and *int_ee* is fully determined because not only the intended and free individuals for this process are known, but also the mode in which they have to be combined is unique. For the satisfaction of terms *int_om, int_mo, int_em* and *int_me* there are an infinite number of ways in which the free individual can join the intended individual and vice-versa. According to the projective default assumption, for the satisfaction of these conditions the free individual is projected in its own direction, and its end or origin is moved to the intersection point with the intended individual. If the projection of the free individual does not joint in 't' the intended individual, the transformation fails, and the condition is marked unsatisfiable in the current state. The satisfaction of these terms is illustrated in Figure 8.2 where α and β stand respectively for the intended and free individual in the transformation.

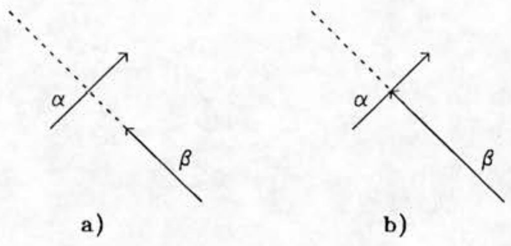

FIGURE 8.2

Now, we come to the conditions that are directly relevant for the satisfaction of design concepts in our example. In the current definition of GRAFLOG, every vertex in the description list of a symbol of type polygon is denoted by one of the following terms: *t_joins_at, e_joins_at, end_of* and *origin_of*. Consider that the last two terms impose no restriction in the line that they define, but just define an unconstrained reference for a polygon vertex. We require, however, a transformation for the satisfaction of the conditions expressed by the terms *t_joins_at* and *e_joins_at*. We also define a procedure for the satisfaction of the parallelism condition. Additionally, we define a general heuristic rule that improves the performance of the overall process and works in conjunction with the other basic transformation rules. The transformations are defined from (1) to (4) as follows:

(1) *t_joins_at* terms are satisfied along the lines of basic join terms as shown in Figure 8.2.

(2) For the satisfaction of *e_joins_at* we consider that these terms denote vertices of polygons in the description of spatial regions in architectural drawings. The edges of a polygon are oriented and the end of every edge is linked to the origin of its next edge in the original definition direction. Suppose that we assign a polarity value '+' to the end-extreme of a line, and a polarity value '-' to its origin extreme. Now, we assume that there is an 'attraction force' between edge extremes of opposite 'polarities', and a 'repulsion force' between edge extremes of the same 'polarity'. For instance, the end of *wall_2* is attracted to the origin of *wall_3* in the transition from Figure 7.15.b to 7.15.c, and there is also a repulsion force between the end of *wall_3* and the end of *wall_2* in the same state. Consider that without these pragmatic considerations, there are four possible ways to satisfy the *e_join_at* condition.

The criterion of 'extremes polarity' discriminates between just two of the four modes in which the *e_joint_at* condition can be satisfied. For discriminating between the two remaining cases we consider the magnitude of such an attraction force. Consider Figure 8.3.a in which *e_join_at*(α, β) has to be satisfied. Suppose that α is the intended individual, d_1 is the distance between the end of α and the origin of β, and d_2 is the distance between the origin of β and the end of α. We consider that the magnitude of the attraction force between two vertices is inversely proportional to the distance between the 'attraction poles'. Then, if $d_1 < d_2$ the transformation produces the relation in Figure 8.3.b and if $d_2 < d_1$ the transformation produces the relation in Figure 8.3.c. We consider the case in which both of these forces have an equal magnitude a very unfortunate one in which we make a random choice.

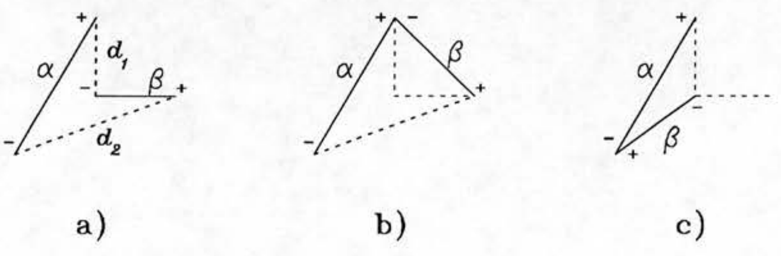

FIGURE 8.3

(3) The satisfaction of the *parallel* condition between two given lines, one of which is taken as the reference for the transformation and the other as the free individual is a very undetermined process. The free line can be rotated in relation to any point lying not only along the line itself but also in the infinite projection of such a line. Besides, the length of the rotated line might be left constant, but it can also be altered in the transformation. Furthermore, the parallelism can be achieved by rotating the free line in either clock-wise or counter clock-wise direction.

In the definition of the heuristic transformation for satisfying the parallel condition, the center of rotation is defined to be either the origin or the end extreme of the line to be rotated. The decision on which extreme of the free line is the one to be modified and which extreme is the center of rotation is made by considering the sense of these two dots, as defined in Chapter 6. If there is a term of the form *origin_of*(α) or *end_of*(α) included in the sense set of a dot, and $\alpha \in \Gamma$ --is an invariant for the whole transformation-- such a dot is considered the center of rotation; if such a dot were modified, an invariant for the whole transformation would require a future modification in the intention propagation process. Consider, for instance, that the terms of the form *origin_of* and *end_of* in sense of v_2 *in Figure 7.15.c are {origin_of(wall_3), end_of(wall_2)}, and the sense of* v_5 includes {*end_of(wall_3), origin_of(wall_5)*} in the same figure. We have chosen v_3 as the center of rotation, preventing a future modification of the dot *end_of(wall_2)*, because *wall_2* is an invariant for the whole transformation. The sense of the dot v_5, on the other hand, is referred to by the origin or end of lines that are free individuals for the whole intention satisfaction process, and this last dot can be set to vary without future negative repercussions. If an individual that is in the set Γ is referred to by terms of both of the dots of the line to be rotated, then the satisfaction of the parallel condition is prevented. If there are no contextual restrictions on either of the dots involved in the transformation, the center of rotation is selected by a random choice.

Despite these considerations, there is still a great deal of non-determinism in the satisfaction of the parallel condition: we have to consider which is the direction of rotation, and how the length of the rotated line is computed. These are also contextual factors. Consider Figure 8.4.a in which α and β are the intended and free individuals respectively. For the definition of the transformation, three construction lines are considered. One of these construction lines is parallel to α and intersects the center of rotation of β. The other two construction lines are perpendicular to α and each one of them intersects one of the extreme points of α.

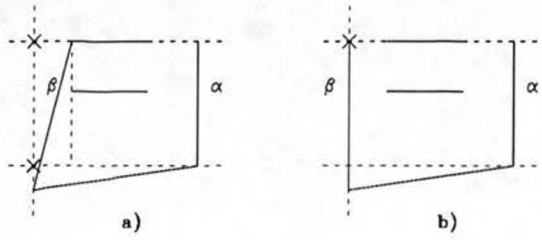

FIGURE 8.4

The construction line that crosses the center of rotation of the free individual determines two intersection points with the other two construction lines, as shown by the marks in Figure 8.4.a. We consider these two points as candidate references for the new position of the point that is altered by the transformation. Here, we make an additional heuristic decision: the new position for the dot to be modified is such that the new length of β is as large as possible as the length of α. When the transformation is applied, the position of the selected dot of β is changed, as shown in Figure 8.4.b

(4) Now, we define an additional heuristic that can be applied in conjunction with all other methods presented above. Consider that the dot that is modified by some transformation function can have other dots attached to it. These attachment relations can be found by examing the sense of the dot to be modified. We are interested, in particular, in the terms of the form *origin_of*(α) and *end_of*(α) as before. If there is some α that is contained in the set of invariants Γ then the transformation is prevented all together, and the corresponding condition is marked as unsatisfiable. On the other hand, if there is not α such that $\alpha \in \Gamma$, and there are other terms of the form *origin_of*(α) and *end_of*(α), where α is not the free individual for the current transformation, then the positions of these terms can be altered in conjunction with the modified dot. That is to say, the dot that is modified by some transformation function of degree of freedom of one 'drags' all its attached dots that belong to free lines in the state. The effect of this rule is equivalent to the unconditional and simultaneous application of several transformation functions. This heuristic has been used in the program SIGRASITT (Santana 1987) in which the satisfaction of design intentions that are manifested upon very complex structures converges very fast --most of the time-- to the solution intended by the human designer. Given these considerations, the satisfaction of the design intention manifested in Figure 7.15 is produced with a significantly lower computational effort, as shown in Figure 8.5.

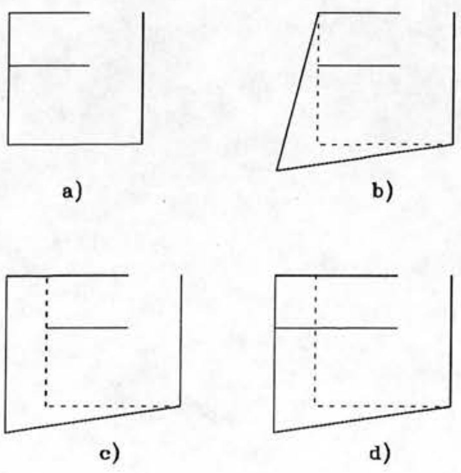

FIGURE 8.5

8.4. Evaluation of the Function Ω^+.

An important consideration that follows from the way the intentional function is expressed is that it can be evaluated in a highly distributed process in a natural way. This feature is important for intelligent CAD systems because the task of finding solutions for interesting real-life problems is likely to demand very large computational power. Furthermore, ICAD systems must support highly interactive environments and they must be able to provide answers fast enough; otherwise, human users might simply abandon them.

8.4.1. Distributed Computation of Ω^+.

Now, consider that an intention to be satisfied in some state in the process is expressed by the pair of local and global intentions. Consider as well that the intention propagation process is defined as,

$$\Omega_{<i,\_>}[P, x, \Phi, Y].$$

We can think of this expression as a message that contains all the information that is relevant for the function computation at every local state in the process. For computing the solution of complex problems, we can consider a highly distributed processing scheme in which the computation is started by sending a message to any available processor. This processor computes the set of local design intentions $\Omega_{<i+1,j>}$ and produces a set of J son states. Then, the parent processor can assemble and send a message for each j son state to some available processor. When this task is accomplished, the father processor is itself available for receiving a message from any other branch or level of the process. Although this issue has not yet been fully explored, a parallel-or distributed processing system for the computation of the intention propagation process could be used.

8.4.2. History of the Problem Solving Task Process.

In solving problem tasks it is usually helpful to be able to give an explanation of the particular solution that is found by the system. In fact, given that the solution of problems is found with the help of heuristic rules in which even random choices are used, an explanation for the problem solution is demanded. A comprehensive explanation mechanism is considered one key feature that an intelligent CAD system should have (Schmitt 1988).

Another very important consideration is that there must be a way to prevent loops over intermediate states in the function computation. Although there have been imposed several constraints for preventing loops in the problem solving process, and although the original design object cannot be reached by a transformation sequence, there might be cyclic processes over non-initial configurations. A way out of this problem could be to determine that the solution space for problems in some specific domain is a partial order. In such a case, an inductive proof for the process of satisfaction of design intentions might be given. However, in the general configurations that are considered here, there are no criteria to establish that such an order exists. In particular, the definition of heuristic rules for producing the transformations make things quite unclear. However, if loops cannot be prevented, they must be detected. Here, if the form of a design object at some state in the intention satisfaction process is as it was in some previous state in the same branch, then there is a cycle, and the corresponding branch of the process has no satisfaction.

For these two reasons we define a history mechanism and extend the definition of a total design intention by including a history record in the argument list of the intention satisfaction procedure. In the same way that the global intention is inherited by all son states in the computation of the function, the history record is incremented with the specification of the intention that was manifested in each state. Here, we add an additional parameter Θ to the intention satisfaction procedure, as follows,

(18) $\Omega_{<i, j>}[P, x, \Phi, Y, \Theta]$.

where Θ contains the specification of the global intention, on the one hand, and the list **H** of index pairs from which the current state is produced. In general, Θ is of the following form

(19) $\Theta = <\Omega_{<global>}, H>$.

For the sequence in Figure 7.15, **H** is:

(20) **H = (<0, 0>, <1, 0>, <2, 0>, <3, 1>, <4, 0>).**

Note that the first element of these ordered pairs is the computation level, and the second represents the *j*-index branch of its corresponding father state. With this information it is possible to reconstruct the whole of the solving problem task.

8.5. Explanation of the Problem Solving Task.

The main design intention expressed in Figure 7.15, produces a sequence of transformations until the state *<4, 0>* in which a new configuration of the design object is reached. This intention is expressed as follows:

(21) **$\Omega_{<4, 0>}$[({origin_of(wall_4), origin_of(wall_5)}, house_1, Φ, Γ, Θ)].**

Now, we define a procedure *explain*(Θ) which produces the detailed history of the solving problem task: the list of triples constituted by the design object at the corresponding state, the intended individual that was taken as a reference for a transformation in that state, and the condition that was satisfied. In the example,

(22) **explain(Θ) =**

[

[house_1$_{<1, 0>}$, wall_2, {e_joins_at(wall_3, wall_2)}],
[house_1$_{<2, 0>}$, wall_1, {parallel(wall_1, wall_3)}],
[house_1$_{<3, 1>}$, wall_3,

{e_joins_at(wall_4, wall_3), e_joins_at(wall_5, wall_3)}],

[house_1$_{<4, 0>}$, none, ∅]

].

The explanation of the problem solving task can be enriched by combining a textual and graphical presentation. Although this issue is fairly complex and it has not been explored yet, the representational environment of GRAFLOG can be used for producing a natural and friendly graphical and natural language explanation of the system behaviour. In general, the history of the computation provides an underlying representation that could be used for producing a natural language explanation of the solving problem task. Furthermore, there is a graphical representation in each state, and the natural language dialogue can make a deictic reference to these pictures. Next, a plausible scenario for this textual and graphical explanation is presented.

In the initial problem solving state, the system produces the expressions in (23) at the time their graphical referents are drawing on the screen, as shown by the corresponding pictures in Figure 8.6. Expressions (a) and (b) introduce the conditions that were satisfied by the transformation that produced the drawings in Figure (c).

(23) (a) *In the initial state these did not join.*
(b) *This was updated.*
(c) *To produce this.*

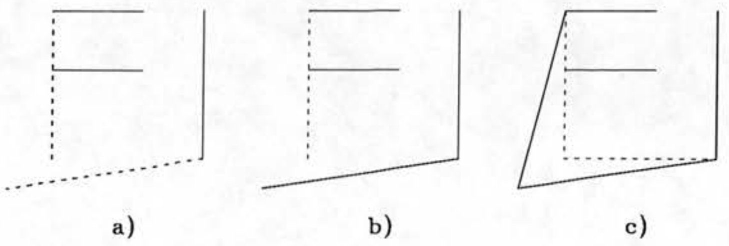

FIGURE 8.6

At the time the word *these* in (23.a) is uttered, *wall_2* and *wall_3* are set to blink as shown by the dotted lines in Figure 8.6.a. For the construction of this linguistic expression, the translation in English of the term *e_join_at(wall_3, wall_2)* is used. Here, some additional pragmatic considerations have to be considered along the lines discussed for the production of linguistic and graphical answers in Section 6.2. The word *this* in (23.b) is supported by setting *wall_2* to blink as shown in Figure 8.6.b. Finally, the graphical support for the word *this* in (23.c) is the change on the drawing itself. The intuition here is that a change rule is better explained by showing the change that it produces, rather than presenting a cumbersome and unnatural explanation of the way such a change is produced.

We can go on developing the explanation along the same lines. For the next transition, the expressions in (24) are supported by their corresponding drawing states and transitions, as shown in Figure 8.7.

(24) (a) *These were not parallel.*
(b) *This was updated.*
(c) *To produce this.*

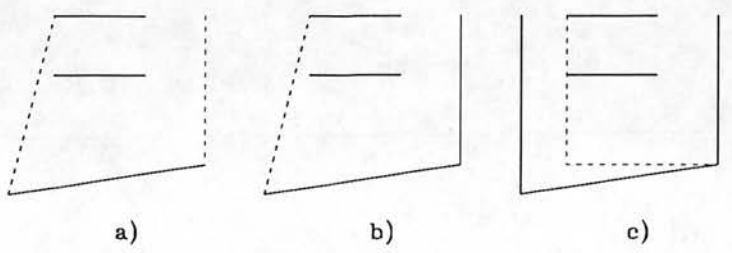

FIGURE 8.7

The last transition is explained by the expressions in (25) and their corresponding graphical support that is shown in Figure 8.8.

(25) (a-b) *These$_a$ did not join this$_b$.*
(c) *They were moved to produce this.*

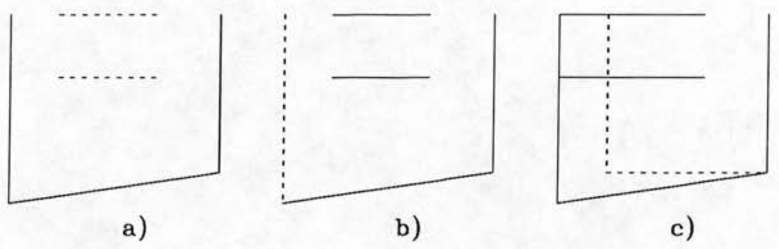

FIGURE 8.8

Here, expression (25.a-b) is supported by two drawings states. The demonstrative *these* refers to the parallel lines that are not joining the left wall, and the demonstrative *this* refers to the *wall_3*. As before, the transition from (b) to (c) is supported by the graphical transformation itself.

An interesting theoretical question that arises from this kind of explanation is to what extent the overt graphical symbols and relations support a simple and natural production of referring expressions in the natural language component. My intuition is that the definition of a pragmatic procedure for producing this kind of discourse could be simple and easy to implement. However, this question requires further research that has not yet been developed.

8.6. The Integrated Design Environment.

Now, we come to a discussion of the integrated design environment that is supported in GRAFLOG.

In this design environment we restrict the system to produce modifications in design objects. The system is not able to produce an original graphical description of the design object, nor to define design concepts.[3] Here, there is no intention to model the kind of inference from which such kind of knowledge is produced. It is not at all clear to what extent such a sort of creativity can be modelled with current computational theory or technology. The original conception of a design object presents itself to the human mind in an immediate manner. The processes that produce such a kind of object cannot be broken into more basic processes or states. The analysis of these holistic acts, that present themselves as finished products of human imagination, in an immediate manner, is very elusive. According to Jones (Jones 1984, p. 19),

> What's striking is that each method begins with a first stage that is
> extremely difficult to do
> Which has no description of how to do it
> Which is intuitive
>
> WHAT EMERGED
> IN WRITING THE BOOK
> WAS THAT TO USE DESIGN METHODS ONE NEEDS TO BE ABLE
> TO IDENTIFY THE RIGHT VARIABLES
> THE IMPORTANT ONES
> AND TO ACCEPT INSTABILITY IN THE DESIGN PROBLEM ITSELF
> ONE HAS TO TRANSFORM THE PROBLEM AND THE SOLUTION
> ALL IN ONE MENTAL ACT OR PROCESS

In GRAFLOG, the task of specifying original design objects and concepts is one that the human designer must perform. In the definition of the design environment, we concentrate rather on showing how the computer system is able to support the human designer in the process of refining the graphical description and conceptual interpretation of the design object.

Next, we discuss the definition and application of design rules. From Chapters 6 to 8 two kinds of design rules have been discussed. The first kind corresponds to user defined rules that are expressed in the course of the graphical and linguistic interaction, and are related to specific design tasks. An instance of this kind of rule is the change rule presented in Section 6.5. The second kind is the set of design heuristics that are used in the intention propagation process.

[3] In a productive environment, however, a rich collection of prototype instances that are recalled from previous design task might be available.

The definition of the first kind of rule is important in several respects. They capture design knowledge of individual human designers which is probably intuitive and acquired through experience; which is meaningful only in particular design tasks. Through the interactive design session, the human designer might realise the need for some particular rule and the design environment must support its definition. However, if the human designer is asked why such a rule is relevant he might not be able to give a consistent and clear explanation. Rules of this kind are relevant in as much as they are used by the interpreter in the solution of design problems. They contain no propositional knowledge. They are rather imperative commands that must be obeyed by the interpreter. And when the human designer realises some common pattern of change in the production of a design object, or maybe in the set of practices of specific design domain, he also realises that a design rule has been learned. However, if these rules are arbitrarily defined then there is no way to predict what kind of object will be produced by their application. In fact, they might turn well-defined objects of some graphical sort into ill-defined structures, and the whole system might be set to a non-equilibrated state.

For this last reason, we need the additional layer of functionality that is provided by the intention propagation process. It is essentially a background process that complements the explicit knowledge expressed by the human designer. Explicit change rules are likely to be focused on particular aspects of drawings, and the user should not be responsible for preventing the global consequences of local transformations. The intention propagation process complements rather than substitutes for the human user. It follows the global consequences of local changes, and produces design objects that are well formed, refer, and are consistent with the explicit knowledge of the design task that is manifested in the original definition of the objects, and in the conditions asserted in the design concept.

In this scheme, we could also consider the definition of an interactive dialogue in which the human designer intervenes in the intention propagation process. For instance, by dictating a change in a drawing by a direct manipulation process, replacing with this action the heuristic rule that otherwise would be applied by the system. Consider again that the intention propagation process determines what has to the computed, and the actual heuristic procedures comprise the knowledge of how such a condition is satisfied. And such 'how' knowledge might be better expressed by the user by himself modifying the drawing, satisfying in this way the condition whose satisfaction is the current task of the intention propagation process. It might be the case that the user is able to express the change directly. Here, the design knowledge might be the change action itself, that can be easily experienced, but that can hardly be explained. We can even consider a set of heuristic rules associated with a specific condition, and define the interaction mode in a way the user selects the rule that he wishes to use in the particular design context.

In summary, in the design cycle human designers express the original graphical description of a design object and its associated design concept. Then, an interactive conversation in which design rules are defined through the graphical and linguistic dialogue is started. In the same dialogue, design intentions that have the purpose of modifying the graphical representation can be expressed. For inferring these intentions, the system can use the rules that are explicitly defined by the user, and the instabilities that they produce can be compensated by means of the intention propagation process. We can envisage different control strategies in which the application of explicit design rules and the intention propagation process are intertwined and co-operate towards the solution of complex problems. In this interdependent control flux, the intended individual for the intention propagation process can come from a change that is directly produced by the human designer, or from a change that is produced by the application of some human defined design rule. However, this issue, like many others in this work, deserves further research. Finally, when the design task has been completed the explanation of the design task must be produced by the system, as was discussed in the previous section.

Here, we conclude this chapter on the solution of design problems. First, the function Ω for the satisfaction of design intentions was defined, and a procedure for its computation was presented. Then, some considerations for the evaluation of the function were discussed. A scenario for the graphical and natural language explanation of the solution of design problems produced by the system was also illustrated. Finally, some considerations for the definition and solution of design problems in the integrated environment defined in GRAFLOG were discussed.

With this chapter, we conclude the second part of this dissertation. In Chapter 5, the role of analogical representations was discussed. In that chapter, a notion of a graphical and logical language was presented, and the notion of graphical and linguistic inference was introduced too. In Chapter 6, a graphical and logical language for our particular design domain was presented. The way this language is used for the definition of design objects was illustrated. We also defined the conditions for a drawing to refer in this restricted domain. On the basis of such a notion, we define the notion of equilibrium between the logical and graphical representational systems in GRAFLOG. Then, a notion of graphical parsing was discussed in detail, and a procedure for producing the interpretation in $\mathbf{L_d}$ of 2-dimensional wire-frame drawings was presented. Finally, an example of the kind of rules that can be defined by the human designer in the course of the graphical and linguistic interaction was presented. In Chapter 7 we introduce the notion of design concept. We showed how these concepts can be expressed in the language $\mathbf{L_d}$ and how they are implemented in the representational structures of GRAFLOG. Then, the notion of design intention was presented, and the intention propagation process was introduced and discussed in detail.

Here, I would like to acknowledge that every individual conceptual constituent of this theory is extremely simple, and that there are many issues that deserve a further and considerable amount of research. The intention here is not to present the solution of specific design problems, but rather to use our very simple example to discuss some of the issues that, in my view, have to be considered in the definition of so-called Intelligent CAD systems. The aim is to present an integrated view of all these issues, and to show that they interact in a complex but systematic way. I would like to highlight as well the interdisciplinary considerations that have to be taken into account for the production of this kind of system. We have considered issues ranging from the highly technical and theoretical disciplines, like formal semantics and linguistics, to the rather pragmatic and intuitive practices and programming techniques that are employed in the definition of CAD systems. It is worth pointing out as well that although many workers and practitioners of the emerging field of intelligent CAD denied the relation between ICAD and psychology, the very fact that we are modelling a human task that is related with the processes of the human imagination make our enterprise, in some sense, a psychological one. Although we are producing 'bad psychology', it is better to be aware of it, rather than believing that we are doing geometry or topology. Finally, there is a philosophical issue that has been involved all along all the second section of this work: this is the question of to what extent, design knowledge can be considered representational. We will face this question directly in Chapter 9, in the conclusions of this dissertation.

Chapter 9

Computer Graphics and Artificial Intelligence

In this concluding chapter I present a brief summary of the most important issues of this thesis. I also shall make some general remarks related to the representation and use of design knowledge in computer systems. I believe that the experiences gained in the integration of computer graphics and natural language processing facilities invite an interesting reflections on the relation between symbolic thought and perceptual processes in the so-called computational metaphor of mind.

GRAFLOG was started with the idea of imposing an overt interpretation upon graphical symbols and relations in interactive computer graphics by means of natural language. For this purpose, we used the most basic mechanism for associating words and pictures: ostension. This exercise led us directly to realise that not only graphical symbols but also graphical relations can be given a linguistic interpretation, as was shown in the interactive dialogue of Chapter 2. A notion of graphical language then emerged in a natural way. This notion is interesting, in my view, because although graphical symbols and relations in a drawing have a geometrical interpretation, this interpretation is neutral in relation to the meaning of the drawing itself. The set of conventions for interpreting a drawing that are expressed in GRAFLOG through the natural language facility, implicitly determines a graphical language (Pineda 1988a, 1988b, 1988c).

The direct utility of this notion of graphical language is that the same graphical interactive environment can be used in different application domains. What has to be given in advance to the program is the geometrical knowledge and a facility for changing the interpretation conventions of drawings in a dynamic way. Then, end-users can customise their own interpretative contexts, developing in this way interactive environments in which the drawings themselves convey meaning, and in which abstract notions can be visualised by means of an appropriate set of conventions. Although the linguistic and graphical knowledge of the current prototype of GRAFLOG is quite limited, simple applications have been developed (Pineda 1988d).

Another important facility that can be developed by integrating computer graphics and natural language is an explanation mechanism through which the system is able to illustrate the methods by which it comes to solve problems. In our design domain, this facility is required given the complexity and magnitude of a representational system for design, in which it might be very difficult to trace the consequences of design decisions in an intuitive way. Additionally, some of the knowledge involved in the solution of problems might not be representational, and it always helps to explain an algorithm in terms of its behaviour, rather than in terms of its structure, as was suggested in Chapter 8. Furthermore, design decisions that are made directly by human designers in the course of the interaction might not be subject to analysis, and the system should reproduce the corresponding human action when an explanation of the problem-solving process is required. In GRAFLOG, the explanation mechanism has been theoretically develop in Chapter 8.

The natural language processing facility of GRAFLOG could not be provided independently of a linguistic theory, and a computational linguistics framework. For this purpose the body of research in so-called unification based theories of grammar (Shieber et al 1983), in particular Unification Categorial Grammar (Zeevat et al 1986a) and its associated semantics, was used. The kind of natural language dialogue that was used in Chapter 2 for the dynamic creation of a graphical language, despite its simplicity, is rich enough to appreciate how complex the structure of natural language can be. However, as was shown in Chapters 3 and 4, and as is shown by other related programs like ACORD (Klein 1987), the grammatical framework that we have adopted is strong enough to handle the spatial oriented dialogues required for our particular goals, and offers a rich environment for further developments in the integration of computer graphics and natural language,

Deixis and ostension have always played an important role in computer graphics interaction. However, the intention that a point action presupposes has always been coded beforehand by programmers. In computer graphics interaction, every meaningful graphical input event, every pointing action, is associated with a particular algorithm whose interpretation is presented to end-users as a compact operation. Users think of these algorithms and their interpretation processes as 'tools' which range from the simple selection of a symbol in a graphical menu to very complex operations that analyse and transform the underlying representation of geometrical objects displayed on the screen. These tools have proved to be very productive in different kinds of academic and professional environments, but they have to be learned, in a passive way, by end-users.

Computer graphics algorithms and their associated data-structures can be thought of as 'representations' of knowledge. Or rather as capsules or packages of knowledge. These algorithms are standing by most of the time, until the input events that prompt their selection are detected at the interface. The result of each of these algorithms can be thought of as a response of the system as a whole to a particular input event. Through these algorithms, the geometry, the programmer's knowledge about computer interfaces, and also some conceptual aspects of the objects that are produced in the course of a graphics interactive session, are modelled. This knowledge, however, is there to be used as a finished product. It cannot be enriched by the end-user. These procedures obey rich but inflexible protocols. Such a kind of knowledge bears no relation to the environment in which it helps to solve problems: it is just like any other tool.

The implicit notion of knowledge representation in computer graphics contrasts with the overt study of knowledge in Artificial Intelligence and Cognitive Science. One important assumption in AI research is that thought can be explained as a process in which symbols are manipulated. The information that is received through perception materialises as symbols that are systematically related. A system of symbols which stand for objects and individuals in the world is a representational system, and knowledge that can be expressed through a system of symbols is *representational.* In representational structures, the meaning of a complex expression is a function of the meaning of its constituent parts.

One pragmatic goal in AI is to develop flexible schemes for representing knowledge, and to provide a bridge between the general and abstract knowledge of an expert, and the domain-specific knowledge of the people that are engaged in the day-to-day problem solving tasks. This goal has been achieved, to a certain extent, with the evolution and the better understanding of so-called expert systems and of knowledge representation schemes, like logical and frame-based representational systems.

Algorithmic representations have also been used in AI, but they are regarded, more or less, as analogical representations. Consider, for instance, the WHISPER program for problem-solving with 'diagrammatic representations' (Funt 1985). This program uses geometrical algorithms for simulating causal relations in a blocks-world subject to gravitational forces. There are also important precedents of co-operation between computer graphics and AI. For instance, the *winged-edge* structure for the representation of solid objects in computer vision (Baumgart 1975) is an important antecedent for so-called *boundary representations* of computer graphics (Requicha 1980) as illustrated by Weiler's polygon comparison algorithm (Weiler 1980).

The important notion that I would like to highlight here is that although there are similarities in the notion of representation in computer graphics and AI, the problems that people deal with in these two computer specialities are certainly **not** the same. AI research has been focused on the structure of representational environments with their associated 'inference mechanisms' and the development of heuristic methods that resemble 'human intelligent faculties'. The work on computer graphics, on the other hand, has concentrated on developing algorithms and their associated data-structures for the production of synthetic pictures. The development of these algorithms is the bread and butter of computer graphics.

One likely point of agreement between practitioners of AI and Computer Graphics is the notion of symbol itself: a symbol is something that stands for something else. However, a single symbol, a picture, is the final result of the work of a computer graphics programmer. Externally, a symbol is a picture on the screen; at the internal level, a symbol is the memory location, the pointer, that addresses the data structure where the geometrical description of the picture is 'represented'. From the point of view of AI, on the other hand, a symbol is the basic conceptual unit in inferential process. Symbols are manipulated in terms of the 'syntactic' structure of the representational system. Whether a symbol happens to have a large or a very small information content does not really matter as long it plays its proper role in the syntactic machinery.

The relationship between high-level AI knowledge representation structures, and the low-level algorithmic knowledge that is used in computer graphics is one of the major topics of this thesis. A particular question that was faced directly is how to explain the representational role of graphics. For this purpose an algebraic structure for capturing the semantics of graphics was developed. As a result of this exercise we showed that a graphical symbol or expression can be treated as expressing a concept, in the same way that a linguistic name does, and that graphical compositions can express propositions.

However, this is not to say that graphics play no important role in reasoning, and can be arbitrarily replaced by some other representational language. In fact, algorithmic knowledge plays an important conceptual and implementational role in the kind of inferences that GRAFLOG's interpreter is able to make. This point is reflected in the dual algorithmic interpretation of the graphical structures **G** in Chapter 5 and **D** in Chapter 6. At a logical or symbolic representational layer, GRAFLOG supports facts and relations, and 'symbolic inferences' are operations on this representational structure. The algorithmic layer, on the other hand, consists of the geometrical information that is required for producing the symbols on the screen, for making graphical compositions, and for computing the set of graphical properties and relations that hold between graphical objects.

If a symbol stored in the symbolic representational structure *NLKB* of GRAFLOG has a graphical realisation, then it is a pointer, or an index, to the geometrical data-base where the information for displaying its image is stored. Consider again the form of an entry in *g_db*:

g_db(name, type, description).

In the *g_db* clause, *name* is the symbol's identifier and is also the object that plays a part in deductive inference processes. The symbol *type* corresponds to a sort in the multi-sorted algebraic system of Chapters 5 and 6. However, it is worth emphasising that *type* is also the name of the algorithm that constructs the corresponding picture and *description* corresponds to the algorithm's parameters. This second role of *type* is abstracted away from in the algebraic analysis. However, the interpretation of the algorithm that is named by *type* is a process that produces the picture on the screen.

The additional information that the graphics provides to the representational system can be appreciated in a different light. According to our theory of semantics for graphics, it makes sense to talk about parsing a drawing in relation to a predefined set of geometrical algorithms that compute properties and relations of well-specified kinds of graphical objects. In computer graphics, the interpretation of a graphical representation depends on the geometrical knowledge of the graphics program. In the same way that a graphical program can only display graphical symbols for which construction algorithms are available, a drawing can only be parsed in terms of the geometrical knowledge or algorithms that the program has been endowed with.

The two levels of representation --*representational* and *algorithmic*-- are reflected in the evaluation procedure of GRAFLOG, which traces the deductive consequences of information asserted through both graphics and language. The 'symbolic' knowledge-base stores symbols and relations in a direct way as PROLOG facts and relations. However, the information that is stored in the geometrical data-base is not used directly by the interpreter, but always in conjunction with one or more geometrical algorithms. The conceptual relevance of the two layers of representation can be appreciated by contrasting the relationship between the graphical input events and the algorithmic layer, on the one hand, and the relationship between GRAFLOG's symbolic evaluation procedure and the algorithms, on the other. The control cycle for handling graphical input events is as follows:

```
loop_for_ever
    wait_for_event(EVENT, INFORMATION);
    case EVENT:
        event_1: algorithm_1(INFORMATION);
        event_2: algorithm_2(INFORMATION);
        ...
        event_n: algorithm_n(INFORMATION);
    end_case;
end_loop;
```

This process is a permanent loop which takes an input event in every interactive cycle. Each event that is meaningful to the system has an identifier and some associated information. A kind of event might indicate, for instance, that the information was collected by means of a pick device in a certain region of the screen, and its associated information might be, for instance, the memory segment in which the selected symbol is stored. These input operations are supported, of course, by algorithms that run at the level of the graphical software. Events in GRAFLOG are, for instance, typing an ostensive definition and selecting a graphical symbol, pointing to a symbol to be translated or rotated, typing a declarative natural language expression, asking a deictic question about a graphical symbol at the time it is pointed out, etc. All these basic operations are interpreted as different events and there is a basic algorithm for interpreting each one of them.

Now consider the relation between the symbolic process and the algorithmic information. For this we show the evaluation of the question of whether a student is clever in the interpretative context of chapter 2. Assume that the state of the *NLKB* is as follows:

```
student(john).
student(pete).
subject(programming).
subject(linguistics).
studies(X, Y) :- student(X), subject(Y), in(X, Y).
clever(X) :- studies(X, linguistics), studies(X, programming).
```

This is a normal PROLOG data-base. When the question

```
? clever(pete)
```

is asked, the answer is found by means of a process that is in part symbolic and in part algorithmic too. The evaluation of this clause proceeds in terms of PROLOG unification which implements the resolution inference rule. The clause for *clever* is evaluated in terms of the clause for *studies*, and this high-level declarative interpretation can be thought of as symbolic. However, this symbolic manipulation process has a floor below which it cannot be further decomposed in any interesting way. Note that the geometrical operator *in* does not have basic facts in the PROLOG data-base to unify with. When such a goal has to be proved, a geometric algorithm is interpreted instead. Every geometrical constant of our graphical languages has an associated computational geometry algorithm. In GRAFLOG's interpreter, there is a basic clause that relates the geometrical constants with their corresponding algorithms. For the constant *in*, for instance, such a clause is of the following form:

```
in(X, Y) :- in_geometrical_algorithm(X, Y).
```

Do we know the answer to the question of whether Pete is clever? Given that we have no picture in front of us, can you tell what would be a good answer for such a question? PROLOG would always respond in this situation: *no*! However, if we know and can compute the algorithm for *in*, and we have a picture in front of us, do we have access to the answer? Some people would say, perhaps, no. I am rather inclined to think that we do.

In the same way that input events select algorithms whose interpretation is embedded in the event-driven control cycle, the interpretation of the algorithm for *in*, and of other geometrical functors, is embedded within the symbolic manipulation process: the algorithm is selected by some appropriate rule. In the analysis for the question *clever(Pete)?* the arguments for *in* are symbols *--pete, programming* and *linguistics--* but the algorithm whose name is also *in* performs the computation in terms of the information that these symbols index in the geometrical data-base. In general, the parameters for these algorithms can be provided either by external events or by the symbolic structure, and the result of these algorithms can be either an external behaviour or a change in the symbolic structure as well.

Here, some declarativist minded critic could say that there is nothing interesting in codifying the *in* operator as an algorithm, and that the facts that it entails can be coded directly in the PROLOG data-base, for instance, by typing *in(pete, programming)* and *in(pete, linguistics)* along with other linguistic facts like *subject(programming)* and *student(john).* In a trivial implementational sense this is in fact the case. However, given the theory of Chapters 5 and 6, there is an important conceptual difference in expressing the same fact as a PROLOG relation or in terms of an algorithm. The algorithm embodies the knowledge that is needed to compute whether two individuals stand in an *in* relation for every possible state of affairs on the screen. The algorithm computes the value of the interpretation function of *in* for every intended model of the representational language. According to the discussion of intensionality of graphical symbols in Chapter 6, the algorithm codifies the knowledge of the intension of a geometrical constant, like *in.* In the same way that **logical constants**, like *and* and *if* have a semantic value that is independent of the current interpretation conventions, **geometrical constants**, like *in* or *int_oo*, always denote the same value. If the set of *in* clauses were given independently of the algorithm, *in* would be a **non-logical constant** of the representational language, and the meaning of *in* would have to be given by the interpretation function of the model. And this function is contingent and varies from model to model and depends on the current set of clauses asserted in the PROLOG data-base. The functions that the constant *in* denotes, in one or the other case, are certainly different.

We can put this in more simple terms: if the algorithm is included, the system can compute all the *in* relations that are required for an inference process in any situation. If the *in* clauses are asserted by overt definitions in the PROLOG data-base, then the algorithm runs in the user's head, because how could he know which *in* clauses hold in a new graphical state? When we are looking to a drawing we can identify the discrete symbols by a basic perceptual ability, and the same can be said of geometrical relations, like *in.* The objects produced by the perceptual process are symbols and relations, but I do not think that it makes any sense to say that the perceptual knowledge itself is representational. And if we believe in the computational metaphor of human mind, then it is plausible to think of this knowledge as algorithmic.

Here, a proceduralist minded person could say that this is nothing new, and given the close relationship between Frame systems and first order logic (Hayes 1985) our structure corresponds to the notion of a frame system with procedural attachment. I think that at an implementational level that is in fact the case. The only comment that I would like to make in relation to this point is that geometrical algorithms attached to the symbolic structure cannot be thought of as representational, and that the function that they compute should be well understood. In the case of GRAFLOG, the functions that are computed by geometrical algorithms can be understood in terms of a background theory, namely geometrical analysis. The inclusion of error elements in the carrier of every sort, as shown in Chapter 5, allows us to think of these functions as total. But the fact that the system makes use of these algorithms does not conflict with the declarative semantics for the symbolic level of representation. Furthermore, these algorithms make a productive contribution to the representational environment, because if they were absent, it would be very difficult to trace which geometrical facts and relations remain true when there is a transition in the state of the knowledge-base. Another way to think of these algorithms in frame-like representational structures is as very well informed default values for geometrical frame-slots (Reiter 1985). An interesting question is, I think, to what extent these algorithms help to solve the so-called frame problem for knowledge-bases representing spatial information.

Should the inference by which we come to know whether *Pete is clever* in the example above count as a formal proof? If the facts *in(pete, linguistics)* and *in(pete, programming)* were asserted directly in the PROLOG representation, the argument counts as a formal proof. In such a case there would be an explicit and valid inference from premises to conclusion showing that *Pete is clever* is in fact the case. The claim that might be controversial is that if we are able to see that *Pete* and *programming*, and *Pete* and *linguistics*, stand in an *in* spatial relation, that should be enough for regarding as valid the rest of the linguistic argument that concludes that *Pete is clever.* The difference in this second form of proof is that we rely not only the 'logical' symbolic manipulation process but also on a perceptual ability. When we have to face the question of whether two objects stand in an *in* relation, we are allowed to verify, visually, that such a relation holds. This resembles the process of GRAFLOG's interpreter: the high-level part of the proof consists in proving the goals by the resolution inference rule in which PROLOG unification is based, but for proving the geometrical premisses of the argument, the resolution process is replaced by the interpretation of a geometrical algorithm. A similar form of inference has been recently advocated by Barwise and Etchemendy (Barwise et al 1988) in relation to a program called Hyperproof. However, they do not emphasise the role of algorithmic knowledge in their notion of proof.

The distinction between these two notions of formal proof is important, in my view, if we regard so-called formal proofs as models for human inferences. I have a strong intuition that the information that is gathered by the visual and other perceptual processes is very important for arriving at conclusions, and for discovering relevant information for the solution of problems. I believe that drawings are indispensable for expressing design knowledge. This position has strong consequences in the definition of a design language, as was shown in Chapter 6, 7 and 8. Conceptual or symbolic design knowledge can be expressed in an expressive enough representational language, but I believe that the epistemological richness and the efficiency factor that perceptual inferences convey plays an important role for modelling design tasks, and that such a kind of information must be processed in an algorithmic way.

However, I do not want to suggest that algorithmic knowledge is the only thing required for modelling design tasks. Symbolic thought is also fundamental and we need a logic of design as well. When we are engaged in a design task we start, very often, with a graphical representation. Next, we can think of the properties and relations that have to be expressed through natural language, or through some sort of design specification language. The analytical part of a design task can be thought of as 'proving' that a certain object has in fact a set of properties and relations that are listed in a design specification. For this proof, the only thing that is required is to take each one of the specified properties and verify, by some mechanical method --normally, an algorithm-- that the design object has in fact such a property. However, in what sense can we say that the synthesis of a design object is a proof of something? In what sense might this kind of process be logical? For answering this question we have to consider logic in its wider conception: logic is the study of the valid arguments in a language. To think of design as an argument may not be a very intuitive notion, but given that design knowledge can be expressed through a design language, a design inference would correspond to a valid argument in such a language.

Of course the definition of a logic of design is quite a demanding task in which we have to take into account relations like causality and change that are very difficult to understand, even in standard uses of logic. There is in addition the complexity that design inferences are 'unconscious', and we only have access to the result of design processes. Although I believe that there is no fundamental difference between inferences of this kind and more familiar linguistic inferences, linguistic arguments can be broken down into their atomic constituents, and their structure can be then examined. For that reason, we can classify a set of structural syntactic patterns that are regarded as valid argument forms. The difference in design, I believe, is not so much in the kind of process involved, but rather in that we lack the magnifying lens through which the atomic constituents of the structure of the argument can be discerned. However, this is a limitation of our analytical tools, rather than of human designers. In the same way that good logic looked first into human language, into the arguments that people take as valid, the logic of design should look into the things that people design.

There is of course the question of the extent to which a given design logic resembles a human design ability. Here, I believe that there is room for setting psychological design experiments in which the design predictions of a theory can be compared with the objects produced by human designers. That would allow us to classify some design theories as 'better' than others.

On the other hand, I believe that psychological problems should not prevent the creation of automatic design systems for productive environments. The objects produced by such a kind of systems would correspond to the abilities of those systems, and they need to bear no resemblance to what human designers would produce in similar circumstances. We would have to learn the design abilities of those systems. Furthermore, we have no way of predicting the result of a human design activity.

A simple theory for producing design arguments in the wire-frame domain was presented in Chapter 8. However, we can think of the notion of design argument in more general terms. Here, I will extend the discussion of a formal proof that was introduced in relation to the geometrical constant *in*. Consider that the purpose of a formal proof is to make evident the structural link between premisses and conclusion of a valid argument. For this reason, it is required that all the information that is relevant for an argument is stated in the same representational language: validity is a notion that depends on the structure of language. Many logicians are used to think in the following way: given an argument, show whether it is valid or not. For this reason, they can play down the role of perception in inference, because whatever symbols are known through vision or touch, for instance, can be translated to an expressive enough representational language.

However, what can we say about the relation of a formal proof of a valid argument and the process by which such an argument is learned? Take, for instance, the Theorem of Pythagoras. Suppose that you know that the sum of the squares of the legs of a right triangle is equal to the square of the hypotenuse. There might be a purely symbolic proof of this theorem, and some skillful logician might know about it. However, I am more concerned here with the kind of process of the mind that would have to be performed for inducing this theorem, and furthermore, for realising its non-trivial theoremhood. We are rather interested in the kind of process that Pythagoras himself used for making such a fundamental discovery. This is, of course, very speculative, but we are very close to the end of this thesis and some speculation should not make much difference.

The best place to start this discussion is the proof that is attributed by tradition to Pythagoras himself (Bronowski 1981). This proof is graphically illustrated in Figure 9.1.

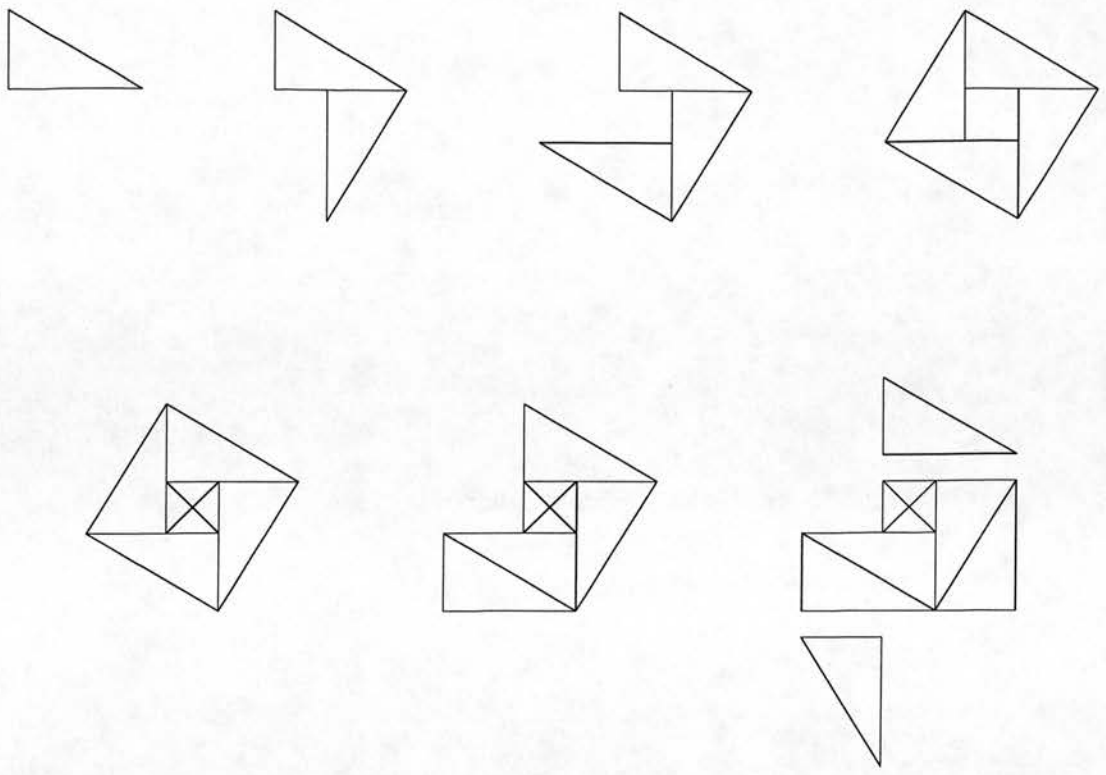

FIGURE 9.1

The proof starts by taking an arbitrary right triangle. The four pictures in the top-row show the sequential addition of one right triangle, each in one step, in the figure. The four triangles have the same dimensions. In the top-right figure, a square on the hypotenuse emerges from the graphical composition. This square is rotated, but it can be appreciated by a basic perceptual ability. There is also in the top-right figure a small square, a hole, in the center of the rotated square. The small square emerges from the graphical composition of the four triangles as well, but in an internal manner. In the bottom-left figure the internal square is marked to highlight its area. Then, we can rotate over one of their vertices two of the basic 'tiles' out of which the square on the hypotenuse was made, one by one, as shown in the last two transformations in the bottom-row. In the last figure, two squares emerge again out of the original triangles and the small marked square. The right triangles above and below in the bottom-right figure have been drawn just to illustrate the fact that each of these two new squares correspond to the smaller and larger legs of the original right triangle. Given that the 'tiles' that conform the square on the hypotenuse are the same as the 'tiles' that conform the squares on the other two legs, the Pythagorean theorem holds.

Here, I would like to highlight the operations that seem to me fundamental not only for realising that the theorem holds, but also for realising its relevance. The first thing to notice is that the proof is made out of a set of two well defined graphical sorts: *triangle* and *square*. These objects are identified by an immediate perceptual process. The second point is that the proof is performed as a process of graphical composition. The operations considered are the addition and rotation of overt basic symbols. For instance, in the first four steps of the graphical composition only one triangle was added in the structure, and only one triangle was rotated in the last two transformations. The third important fact, I believe, is the ability to realise that symbols of one sort emerge in the graphical composition in terms of symbols of other sort. For instance, a square emerges in an appropriate composition of two triangles. These emerging objects can include other emerging objects as well, as the small square in the middle of the hypotenuse square. These emerging patterns are also realised by a basic perceptual ability. Finally, the theorem itself asserts a relation between a property of emerging patterns --the area of the hypotenuse square is the same that the sum of the areas of the squares on the legs of the basic right triangles-- and not between the basic overt context-independent building blocks.

Suppose that we define a graphical structure for capturing the Pythagorean operations along the lines of the graphical languages in Chapter 5 and 6. Each step of the construction process in the Pythagorean proof would correspond to a well-formed expression of such a language. Of course the emergent patterns in the language of Chapter 6 are identified by an explicit ostensive definition, as the polygon standing for the house, and a language for this more ambitious enterprise would have to be given a more powerful perceptual ability for realising the emergent patterns by itself. For instance, we would require a set of algorithms that can recognise right angles and make use of construction lines, as defined in Chapter 6, for determining that one or more objects of sort square emerge from a graphical context. In this way, the squares of the top-right and bottom-right figures could be identified. Additionally, in order to learn the theorem we would require an inductive rule in terms of which the emerging objects in the two crucial stages of the proof could be related.

Although this is an issue for further research, I believe that an example of such an inductive rule is suggested by the Pythagorean proof itself. The crucial thing, it seems to me, is that the theorem is a relation between objects of the same sort that emerge out of the same construction blocks in two different graphical compositions in the same sequence of transformations. Furthermore, one of the tiles that is required for the construction of the configuration of two squares in the bottom-right figure, the inner square, emerged itself from the basic triangles in the same construction stage in which the square on the hypotenuse was produced. Then, it was carried on in the construction process --although this square does not undergo a transformation itself. When these basic perceptual facts have been realised the proof is granted by the causal link in the graphical composition, through which the fact that the area of the two crucial configurations is preserved in the chain of transformations is known in an immediate manner.

Notice that there is no claim that this proof is performed in a purely perceptual manner, nor that one part of the proof is perceptual and the other is symbolic. The two aspects of the proof run together. The proof can be produced in a representational language, but each step of the proof goes in hand with the use of a perceptual ability.

The sort of reasoning by which the Pythagorean proof is developed reflects, it seems to me, the kind of reasoning that goes on in design. In design we have to discover objects, and this discovery process in architectural, mechanical, and many other design fields, depends on the identification of 'interesting' shapes.

A currently popular approach to the definition of design languages for architecture and other design domains is based on so-called Shape Grammars (Krishnamurti et al 1979, Weissman Knight 1981, Stiny 1975, Woodbury 1988). Shape grammars are conventional graphical visualisations of an underlying context-free generative grammar. Each drawing produced by a Shape Grammar corresponds to a sentence that can be generated by the recursive application of the generative rules of the language. Shape grammars are able to produce a rich and useful range of patterns that are productive in the definition and implementation of design systems. However, I believe that the Shape Grammar formalism misses the relevance of emerging objects in graphical context as well. Emerging objects can be realised by human perception in drawings produced by these grammars, but the knowledge of these objects is not captured in the underlying representational language. For instance, Weissman's Shape Grammar defines the Pythagorean operations and has right triangles and squares as primitives. This grammar not only produces some of the Pythagorean patterns, but also some of their interesting variations. But the crucial configuration for the proof of the Pythagorean theorem is never produced by his system. Although the squares that emerge in terms of overt triangles can be perceived, the small square within the square on the hypotenuse cannot be carried on to other compositions -- these squares are not known to the representational language-- and the second pair of squares of the proof does not emerge.

Here I would like to raise the question of whether this proof of the Theorem of Pythagoras should count as formal. I think it should. Furthermore, if we restricted the proof to its symbolic aspect, we would miss, perhaps, the fundamental intuition. One property of this proof that I find very difficult to reflect in a natural way in a standard formal logical proof is the presence of symbols that emerge in the context of others. In a first order logical language, both basic and emergent objects would have to be represented as individual constants. But the relation between these two kinds of entities might have no proper logical analysis. The relation between basic and emergent objects runs at the geometrical level. When these emerging objects are realised, as well as their properties, some interesting processes that can be subject to analysis might be transparent to the facts that can be known by immediate inspection. Because a picture is more than the sum of its parts.

References

. Arbab, F, ''Examples of Geometric Reasoning in OAR,'' in *Intelligent CAD Systems II*, ed. V. Akman, P. J. ten Hagen, P. J. Veerkamp,Springer-Verlag, Berlin (1989).

. Barwise, J and J Etchmendy, *A Situation-theoretic account of reasoning with Hyperproof*, Extended Abstract for the STASS Meeting at Asilomar March, 1988.

. Baumgart, B C, ''A Polygon Representation for Computer Vision,'' *National Computer Conference*, pp. 589 - 595 (1975).

. Beynon, M and A Cartwright, ''A Definitive Programming Approach to the Implementation of CAD Software,'' in *Intelligent CAD Systems II*, ed. V. Akman, P. J. ten Hagen, P. J. Veerkamp,Springer-Verlag, Berlin (1989).

. Beynon, W M and A G Cohn, ''Representing design knowledge in a definitive programming framework,'' in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988).

. Bolt, R A, " ''Put-That-There": Voice and Gesture at the Graphics Interface,'' *ACM Siggraph'80 Conference Proceedings, Computer Graphics* **14**(3)(1980).

. Borning, A, ''The Programming Language Aspects of Thinglab, A Constraint-Oriented Simulation Laboratory,'' *ACM Transactions in Programming Languages and Systems* **3**(4) pp. 353 - 387 (1981).

. Bronowski, J, ''The Music of the Spheres,'' pp. 155 - 187 in *The Ascent of Man*, BBC Corporation, London (1981).

. Brown, D C and B Chandrasekaran, ''Design Considerations for Picture Production in a Natural Language Graphics System,'' *Computer Graphics* **15**(2) pp. 174 - 207 (1981).

. Calder, J, M Moens, and H Zeevat, *A UCG Interpreter*, Centre for Cognitive Science, University of Edinburgh (1986). ESPRIT ACORD (P393) Project Deliverable T2.6

. Calder, J, E Klein, M Moens, and H Reape, *English Parser*, Centre for Cognitive Science, University of Edinburgh (1988). ESPRIT ACORD (P393) Project Deliverable T1.6

. Clocksin, W F and C S Mellish, *Programming in Prolog*, Springer-Verlag, Berlin Heidelberg (1981).

. Cunningham, J J and J R Dixon, ''Designing with Features: Origin of Features,'' in *Proceedings of ASME International Computers in Engineering Conference and Exhibition*, ASME, San Francisco (1988).

. Dale, R, *Cooking up Referring Expressions*, Centre for Cognitive Science, University of Edinburgh (1989). Research Paper

. Dixon, J R and C L Dym, ''Artificial Intelligence and Geometric Reasoning in Manufacturing Technology,'' *Applied Mechanical Review* **39**(9) pp. 1325 - 1330 (1986).

. Dowty, D R, R E Wall, and S Peters, *Introduction to Montague Semantics,* D Reidel Publishing Company, Dordrecht, Holland (1981).

. Doyle, J, ''A Glimpse of Truth Maintenance,'' in *Artificial Intelligence: An MIT Perspective*, MIT Press, Cambridge (1979).

. Flew, Antony editor, *A Dictionary of Philosophy,* Laurence Urgdang Associates, Pan Books Ltd, London (1979).

. Frege, G, ''On Sense and Reference,'' pp. 56 in *Translations from the Philosophical Writings of Gottlob Frege*, ed. P T Geach and M Black,Blackwell, Oxford (1952).

. Funt, B V, ''Problem Solving with Diagrammatic Representations,'' pp. 441 - 456 in *Readings in Knowledge Representation*, ed. R J Brachman & H J Levesque,Morgan Kaufman Publishers Inc., Los Altos, California (1985).

. Gandhi, A, ''A Natural Language Approach to Feature Based Modeling,'' in *Proceedings of International Conference on Computer-Aided Design and Computer Graphics*, International Academic Publishers, Beijing (1989).

. Goguen, J A, J W Thatcher, and E G Wagner, ''An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types,'' pp. 80 - 149 in *Current Trends in Programming Methodology*, ed. edited by R T Yeh,Prentice-Hall (1978).

. Hayes, P J, ''The Logic of Frames,'' pp. 287 - 296 in *Readings in Knowledge Representation*, ed. R J Brachman & H J Levesque,Morgan Kaufman Publishers Inc., Los Altos, California (1985).

. Johnson, M and E Klein, *Discourse, Anaphora and Parsing,* Centre for Cognitive Science, University of Edinburgh (1988).

. Jones, J C, ''How My Thoughts About Design Methods Have Changed During the Years,'' in *Essays in Design*, John Wiley & Son, Chichester (1984).

. Kamp, H, ''A theory of Truth and Semantic Representation,'' *Formal Methods in the Study of Language* **136** pp. 277 - 322 Mathematical Centre Tracts., (1981).

. Karttunen, L, ''D-PATR: A Development Environment for Unification-Based Grammars,'' pp. 744 - 80 in *Proceedings of the 11th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics*, Institut fuer Kommunikationsforschung und Phonetik, Bonn University, Bonn (1986).

. Kimura, F, H Suzuki, and L Wingard, ''A Uniform Approach to Dimensioning and Tolerance in Product Modelling,'' pp. 165 - 171 in *Proceedings of CAPE'86, Second International Conference on Computer Applications in Production and Engineering*, (1986).

. Klein, E, *Grammar Frameworks,* Centre for Cognitive Science, University of Edinburgh (1988a). Research Paper EUCCS/RP-26

. Klein, E, *Topics in Unification Categorial Grammar,* Centre for Cognitive Science, University of Edinburgh (1988b).

. Klein, E H, ''Dialogues with Language, Graphics and Logic (P 393),'' pp. 867 in *Proceedings of ESPRIT '87 Conference*, North-Holland (1987).

. Knight, K, ''Unification: A Multidisciplinary Survey,'' *ACM Computing Surveys* **21**(1) pp. 93 - 124 (1989).

. Knight, T Weissman, ''Languages of designs: from known to new,'' pp. 213 - 238 in *Environment and Planning B*, (1981).

. Kripke, S, *Naming and Necessity*, Basil Blackwell, Oxford (1980).

. Krishnamurti, R and P H O'N Roe, ''On the generation and enumeration of tessellation designs,'' pp. 191 - 260 in *Environment and Planning B*, (1979).

. Leon, G and L A Pineda, *Un Algoritmo para el Cálculo de Intersecciones Vectoriales en Computación Gráfica*, Instituto de Investigaciones Eléctricas (1986). (unpublished memorandum)

. Light, R A and D C Gossard, ''Modification of Geometric Models through Variational Geometry,'' *Computer-Aided Design* **14**(4)(1982).

. Lindsay, R K, ''Images and Inference,'' *Cognition*, (29) pp. 229 - 250 (1988).

. Lyons, J, *Introduction to Theoretical Linguistics*, Cambridge University Press, Cambridge (1968).

. Millington, K, A Robertson, and T Smithers, ''An Architecture for AI-Based Design: A Foundation Made Concrete,'' in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD.*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988).

. Moens, M, *Formalizing Nonmonotonic Inference*, Centre for Cognitive Science, University of Edinburgh (February, 1989). (IDC Lecture Notes).

. Opas, J. and M. Mantyla, ''Manufacturing Knowledge into CAD systems,'' in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988).

. Pineda, L A, *Un Algoritmo General para la Comparación de Polígonos*, Instituto Tecnológico de Monterrey, México (1986). M. Sc. dissertation (in Spanish)

. Pineda, L A, E Klein, and J Lee, ''GRAFLOG: Understanding Graphics Through Natural Language,'' *Computer Graphics Forum* **7**(2)(1988a).

. Pineda, L A, ''GRAFLOG: A Graphical and Logical Programming Language,'' in *Proceedings of the IFIP TC5 International Conference on CAD/CAM. Technology Transfer to Latin America: Mexico City, August 22-26 '88.*, ed. G. Leon Lastra, J. Encarnacao and A G Requicha,Elsevier Science Publishers B. V., North Holland (1988b).

. Pineda, L A, ''A Compositional Semantic for Graphics,'' in *Eurographics'88 Conference Proceedings*, ed. D. Duce and P. Jancene,Elsevier Science Publishers B. V., North-Holland (1988c).

. Pineda, L A and N Chater, ''GRAFLOG: Programming with Interactive Graphics and PROLOG,'' in *New Trends in Computer Graphics, Conference Proceedings of CG International' 88, Geneva,* Springer-Verlag, Heideberg (1988d).

. Pineda, L A, J Lee, and A Bijl, ''GRAFLOG: A Notion of Design for ICAD,'' in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD.*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988e).

. Pineda, L A and E H Klein, ''On The Integration of Graphical and Linguistic Knowledge for CAD Systems,'' in *Pre-prints of the Third Eurographics Workshop on Intelligent CAD*, ed. Ten Hagen, (1989).

. Pollard, C, *Categorial Grammar and Phrase Structure Grammar,* Centre for the Study of Language and Information, Standford University (1985).

. Pratt, M J, ''Synthesis of an Optimal Approach to Form Feature Modeling,'' in *Proceedings of ASME International Computers in Engineering Conferenceand Exhibition,* , San Francisco (1988).

. Preparata, F P and M I Shamos, *Computational Geometry: an Introduction,* Springer-Verlag, Heideberg (1985).

. Reiter, R, ''On Reasoning by Default,'' pp. 402-410 in *Readings in Knowledge Representation,* ed. R J Brachman & H J Levesque,Morgan Kaufman Publishers Inc., Los Altos, California (1985).

. Requicha, A G, ''Representation for Rigid Solids: Theory, Methods and Systems,'' *ACM Computing Surveys* **12**(4)(1980).

. Requicha, A G, ''Geometric Modeling and Programmable Automation,'' in *Proceedings of the IFIP TC5 International Conference on CAD/CAM. Technology Transfer to Latin America: Mexico City, August 22-26 '88.*, ed. G. Leon Lastra,IIE and Conacyt, México (1988).

. Rossignac, J R, ''Interactive Design with Sequences of Parameterized Transfromations,'' in *Intelligent CAD Systems II*, ed. V. Akman, P. J. ten Hagen, P. J. Veerkamp,Springer-Verlag, Berlin (1989).

. Russell, B, ''On Denoting,'' *Mind* **14**(1905).

. Santana, J S, *La Manipulación de un Modelo Geometrico Bidimensional para el Diseño de Torres de Transmición,* Instituto Tecnológico de Monterrey, México (1987). M. Sc. dissertation (in Spanish)

. Saussure, F, *Course In General Linguistics,* Peter Owen Limited, London (1974).

. Schmitt, G, ''IBDE, VIKA, ARCHPLAN: Architectures for Design Knowledge Representation, Acquisition, and Application,'' in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD.*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988).

. Shamos, M I, *Computational Geometry,* University Microfilms International (1978). Ph. D. Thesis, Yale University

. Shieber,, Uszkoreit, Pereira, Robinson, and Tyson, "The Formalism and Implementation of PATR-II," pp. 39 - 79 in *Research on Interactive Acquisition and Use of Knowledge*, ed. Grosz and Stickel,SRI International, Menlo Park (1983).

. Stiny, G, *Pictorical and Formal Aspects of Shape and Shape Grammars,* Basel, Birkhauser Verlag (1975).

. Sutherland, I E, "Sketchpad: A Man-Machine Graphical Communication System," pp. 329 - 346 in *AFIPS SJCC Proceedings,* (1963).

. Suzuki, H, H Ando, and F Kimura, "Synthesizing Product's Shapes with Geometric Design and Reasoning," in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD.*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988).

. Szalapaj, P J, *Logical Graphics: Logical Representation of Drawings to Effect Graphical Tranformations*, Ph. D. thesis, University of Edinburgh 1988.

. Tarski, Alfred, "The Semantic Conception of Truth," *Philosophy and Phenomenological Research* **4** pp. 341-375 (1972).

. Tilove, B, "Set Membership Classification: A Unified Approach to the Geometric Intersection Problem," *IEEE Transactions on Computers* **C-29**(10)(1980).

. Tomiyama, T and H Yoshikawa, "Extended General Design Theory," pp. 95 - 124 in *Proceedings of the IFIP WG 5.2 Working Conference on Design theory for CAD*, ed. Yoshikawa and E.A. Warman,North-Holland, Amsterdam (1987).

. Tweed, C and A Bijl, "MOLE: A Reasonable Logic for Design?," in *Intelligent CAD Systems II*, ed. V. Akman, P. J. ten Hagen, P. J. Veerkamp,Springer-Verlag, Berlin (1989).

. Veerkamp, V, "The Multi-world Mechanism in IDDL," in *Pre-prints of the Second IFIP WG 5.2 Workshop on Intelligent CAD.*, ed. H. Yoshikawa and T. Holden,The University of Tokio, Tokio (1988).

. Weiler, K, "Polygon Comparation using Graph Representations," *ACM Siggraph'77 Conference Proceedings, Computer Graphics* **11**(2)(1977).

. Winograd, T, *Understanding Natural Language,* Edinburgh University Press, Edinburgh (1972).

. Wittgenstein, L, *Philosophical Investigations,* Blackwell and Mott, Guildford (1963).

. Zeevat, H, E Klein, and J Calder, *Unification Categorial Grammar,* Centre for Cognitive Science, University of Edinburgh (1986a).

. Zeevat, H, *A Specification of InL,* Centre for Cognitive Science, University of Edinburgh (1986b). Research Paper EUCCS/RP-26