8

A Graphical and Logical Language for a Simple Design Domain

Luis A. Pineda and Ewan H. Klein

EdCAAD and Centre for Cognitive Science
University of Edinburgh
20 Chambers Street, Edinburgh EH1 1JZ, UK

Abstract: In this paper 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 of the sort 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 from which they emerge. For the identification of these emerging graphical objects we introduce the notion of *intension* of a graphical symbol. In addition, 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.
Keywords: Semantics of Graphics, Knowledge Representation, Intelligent CAD.

1. Introduction

In this paper we show how 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 possessing a clear semantic interpretation. The language is specified as an algebraic system. 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 [1]. We also show the implementation of the graphical and logical language in a programming environment called GRAFLOG [6-8]. The language is formally presented in §2. In §3 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 throughout a process of change.

In §4, 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 §5, 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, the definition of some design constraints and modelling rules in the course of graphics and linguistic dialogue is illustrated in §6.

2. The Graphical Structure D and the Language L_d for Design

For the formalisation of the structure of graphics and its integration in the logical representation, the following strategy is adopted. First, an algebraic structure **D** for graphics in 2-dimensional wire-frame domain is developed. This structure has a declarative and an algorithmic interpretation. This duality is at the heart of the theory, and the two interpretations of **D** are illustrated. Second, the structure **D** is embedded within the modal first order logical language $\mathbf{L_d}$ 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.

In the formulation of **D** we follow Goguen et al.'s approach to abstract data types [4]. 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*.

2.1. The graphical structure D

After these preliminaries, we turn to the graphical structure **D** for drawings made out of dots, lines and polygons.

(1) The set $S_{\mathbf{D}}$ of sorts = $\{bool, real, real\_pair, dot, line, polygon\}$.

The individual constants are:

(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 $\{d_0, d_1, ...\}$ of dots.

(5) $\Sigma_{e,\,line}$ is a (non-countably) infinite set $\{l_0, l_1, ...\}$ of lines (construed as vectors).

(6) $\Sigma_{e,\,polygon}$ is a (non-countably) infinite set $\{p_0, p_1, ...\}$ 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*(*true*, *A*, *B*) denotes the dot in which the vectors *A* and *B* intersect each other as shown in Fig. 1. The first argument with the value *true* indicates that such a dot actually exist in the state of affairs that is denoted by the algebraic expression. If the vectors *A* and *B* were parallel, the intersection dot would not exist, the first argument of *int_mm*'s arity would be false, and the whole expression would have no referent in that state. We discuss further the treatment of this kind of irregularities in §2.3.

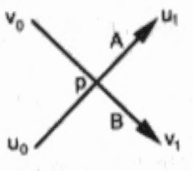

Fig. 1. Two vectors intersect each other

There are also 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) $\Sigma_{bool\ dot\ line,\ bool} = \{on\}$.

The syntactic rules for the definition of well-formed expressions of **D** are given below, when the logical extension of this structure is presented.

2.2. The language $\mathbf{L_d}$

Now, we embed the structure **D** within a modal first order logical language $\mathbf{L_d}$. This will allow us to represent graphical and logical knowledge in an integrated fashion. We define the set of sorts $S_\mathbf{d}$ of $\mathbf{L_d}$ as follows: $S_\mathbf{d} = S_\mathbf{D} \cup \{individuals\}$. These additional 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:

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

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

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

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

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

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

(24) *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 $x_{s,i} \in \Sigma_{e,\ s}$.

(25) *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 $c_{s,i} \in \Sigma_{e,\ s}$.

(26) *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 $p_{s,i}$.

(27) *Auxiliary symbols*: '(' and ')'.

Formation rules are expressed as follows:

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

(29) 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.

(30) 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*.

(31) 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*.

(32) 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*.

(33) 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*.

2.3. Error elements and the closure of $\mathbf{L_d}$

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. Consider the case in Fig. 1 in which vectors A and B intersect each other. The expression *int_mm(true,A, B)* denotes the intersection dot. Notice that the first argument in the operation's rank is a term of sort *bool*. This term is defined to be *true* or *1* if the result of the operation is of a proper sort, as it is the case in Fig. 1. However,

suppose that the arguments of sort *line* of *int_mm* are two vectors that happen to be parallel. Then, the term *int_mm(false, A, B)* denotes no dot at all in relation to such a pair of vectors. In general, an operation of **D** 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.

For handling irregular conditions in drawings we define an infinite number of *error* or ill-formed elements of every sort in every carrier in $S_\mathbf{D}$. 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. For instance, if A and B are parallel then $int\_mm(false, A, B) = e_{<dot,i>}$. With this provision for error handling, every expression of **D** and in which every argument is a symbol of an appropriate sort denotes a total function. Errors are propagated in complex expressions; for instance, $int\_mm(false, e_{line,i}, B) = e_{<dot,j>}$.

2.4. Algebraic interpretation of $\mathbf{L_d}$

A *model M for* $\mathbf{L_d}$ is an ordered tuple $<D, I, F>$, where $D = <\mathrm{D}_s>_{s\in S}$ 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 the set of all non-logical and non-graphical[1] 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 $\mathbf{L_d}$:

(34) 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.

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

(36) 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.

(37) If f is an operation symbol in **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.

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

(39) 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 $[[t_1]]^{\mathbf{M, i, g}},..., [[t_n]]^{\mathbf{M, i, g}}$.

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

(40) 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 $[[\beta]]^{M,i,g}$.

(41) 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.

(42) 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 $\wedge$).

(43) 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.

(44) 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.

(45) 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:

(46) 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 defined below by the system of vectorial equations in (47). 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 (37) and (39).

Functional application in (36) 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 is useful for the definition of design concepts [8].

2.5. 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 again Fig. 1. The vector *A* is defined by the position vectors u_0 and u_1, and the vectorial equation $A = u_1 - u_0$ holds. We have a similar equation for vector *B*. The position vector p of the intersection point between two arbitrarily defined vectors $u_1 - u_0$ and $v_1 - v_0$ in the space is defined in terms of the following equations, where t_1 and t_2 are scalar parameters:

(47) $p = u_0 + t_1 * (u_1 - u_0)$
$p = v_0 + t_2 * (v_1 - v_0)$

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 Fig. 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 *o*rigin of α; if x is m, the intersection point p occurs in the *m*iddle of α; and if this symbol is e then p occurs at the *e*nd 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 Fig. 2. This set of cases is considered in the language for the definition of construction lines. In Fig. 2, the α vector is horizontally oriented, and the β vector is vertically oriented.

Now we consider the relation between the algebraic and algorithmic interpretations of **D** for operator symbols of rank *bool line line, dot*. As was mentioned, the operation *int_mm* denotes 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 Fig. 1 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) = $e_{<dot,i>}$.

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 Fig. 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* [13]. Algorithms for comparing two arbitrary polygons can be efficiently implemented [5, 14]. The output of these operations is illustrated in Fig. 3.

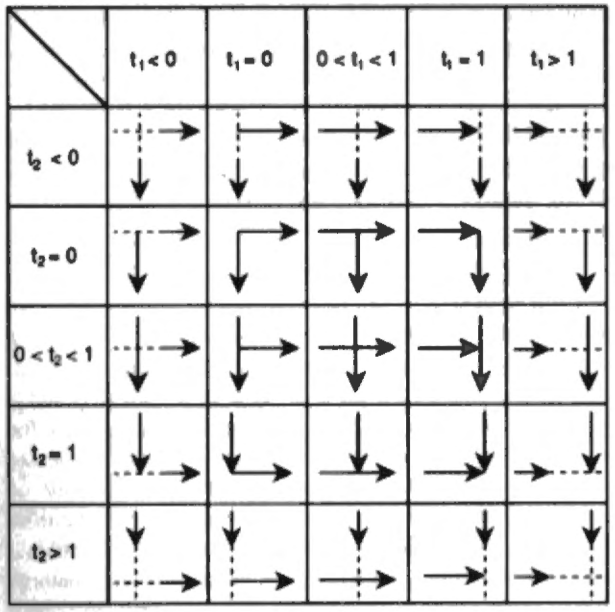

Fig. 2. The 16 cases for the definition of construction lines

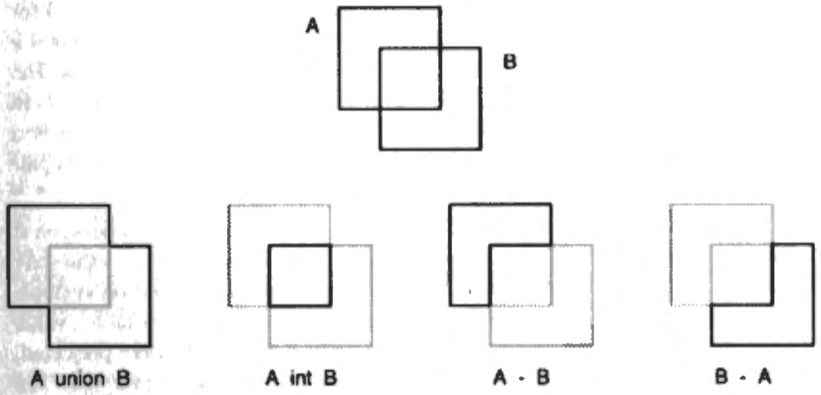

Fig. 3. The output of polygon comparison operations

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 [9, 11].

3. Intensionality in L_d

The language 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, but a 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: [12]

> 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.

3.1. 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 [3]

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 Fig. 4.a.

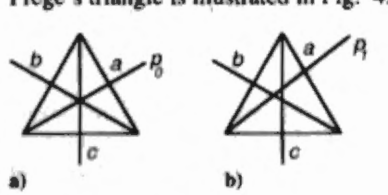

Fig. 4. Frege's triangle

In relation to the triangle in Fig. 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) $\Box\forall x, y_{line}[\text{int\_mm}(x, y) = \text{int\_mm}(x, y)]$

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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 □ of $\mathbf{L_d}$. Note that the truth value of expression (51) is true,

(51) **□on(int_mm(a, b), a)**

but the truth value of (52) is false.

(52) **□on(int_mm(b, c), a)**

Expression (52) is false because there are situations such as the case in Fig. 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'[2] 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 [2] 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 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 Figs. 4.a and 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.

[2] These kinds of context are known as *referentially opaque constructions*. See, for instance, Dowty et al. pp.143 [2].

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 Fig. 4.a is p_0,[3] but the same value of the same expression in Fig. 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 Figs. 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 expression of **D** and 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.[4]

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

(54) $\Box\forall x, y, z\,[x = \text{int\_om}(y, z) \vee x = \text{int\_em}(y, z) \vee x = \text{int\_mo}(y, z) \vee x = \text{int\_me}(y, z)]$
$\equiv x = \text{t\_joins\_at}(y, z)$.

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

[4] 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.

(55) $\Box\forall x, y, z\,[x = \text{int\_oo}(y, z) \vee x = \text{int\_oe}(y, z) \vee x = \text{int\_eo}(y, z) \vee x = \text{int\_ee}(y, z)]$
$\equiv x = \text{e\_join\_at}(y, z)$.

(56) $\Box\forall x, y, z\,[x = \text{e\_join\_at}(y, z) \vee x = \text{t\_joins\_at}(y, z)] \equiv x = \text{join\_at}(y, z)$.

(57) $\Box\forall x, y, z\,[x = \text{cross\_at}(y, z) \vee x = \text{join\_at}(y, z) \vee x = \text{int\_ww}(y, z)]$
$\equiv x = \text{intersect\_at}(y, z)$.

These formulas define a hierarchical categorisation on the 25 intersection cases in Fig. 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 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.

3.2. 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 [6, 8]. 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? 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 denote the same object in every design state a 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 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.

Consider as well that an expression of 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 the identity of the referred object is not fully determined in such a state. We adopt the additional convention that a name in the linguistic domain that is given 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.

Next, we show how the representational environment provided by L_d is implemented and used in GRAFLOG.

4. 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 L_d. For the implementation, two representational structures are used. The linguistic information is represented as Prolog's facts and relations and the graphical information is stored in a set of geometrical clauses g_db.

Here, 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. Now, suppose that the drawing in Fig. 5 is created by editing five lines, at the time the ostensive definition in (58) is typed in by the user.[5]

[5] 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.

(58) *These are walls.*

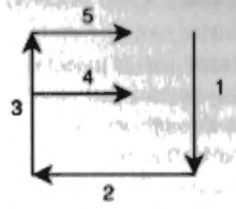

Fig. 5. Drawing of a house

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 the knowledge-base and also the geometrical data-base g_db in an appropriate manner. In the linguistic domain, the following facts will be asserted, 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 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 L_d, and description is a list of terms denoting the points of the space, the parameters, by which the graphical object is defined. 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.[6] 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_{o6}, p_{e6}]).**

[6] Objects of sort dot are not implemented directly, and dots are represented through their corresponding positions.

The identifier of a given object in g_db is also used as the name of the same individual in the domain knowledge-base. 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).

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*

is typed in, a polygon whose vertices are the dots identified in Fig. 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.

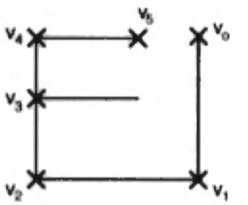

Fig. 6. Dots identifying the vertices of a polygon

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 Fig. 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 the knowledge-base. 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 $\mathbf{L_d}$ 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.[7] Obviously, the 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 $\text{origin\_of}_{\text{line, dot}}$ 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. In order to denote, graphical symbols must 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 Fig. 7.

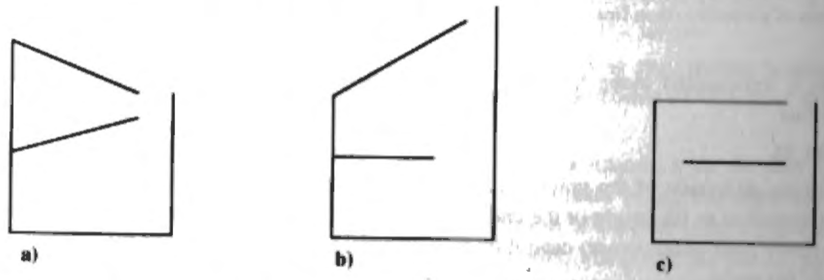

Fig. 7. Three variations of 'house_1'

[7] 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.

Under the current house's definition, Figs. 7.a and 7.b are permissible variations of 'house_1'. However, the drawing in Fig. 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) = $e_{<dot, \perp>}$ As a consequence, house_1 = $e_{<polygon, \perp>}$.

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 Figs. 7.a and 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 Fig. 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 Fig. 8.a, as was shown above for the house.

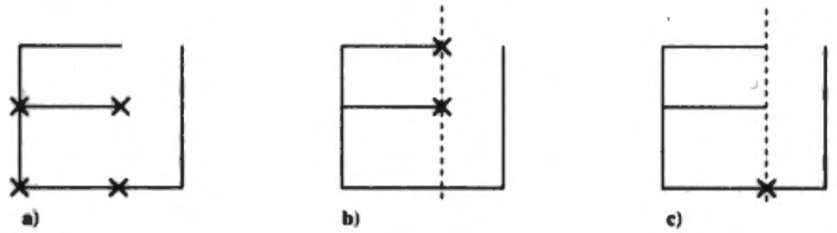

Fig. 8. Definition of a construction line

The room is represented along the lines of other graphical objects named by common nouns, and the fact

(68) **room(room_1).**

is asserted. 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 Fig. 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 Fig. 8.b. Note that this line 'emerges' from the graphical context in which it is defined. This is an important consideration because if the

lines in terms of which the construction line is defined, namely wall_5 and wall_4, are altered, the construction line will be modified accordingly. However, 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 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 Fig. 8.c. Such a dot is denoted by the term intersect_at(c_line_1, wall_2) in $\mathbf{L_d}$. 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 Prolog 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 (59.1) 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.[8] The translation of expressions of $\mathbf{L_d}$ into Prolog is as follows:

[8] 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.

(71) Every clause of the knowledge-base 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, two kind of constants normally distinguished: logical and non-logical. 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 a valuation 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 when the algebraic interpretation of the language was presented. The interpretation of truth-functors, as '$\wedge$', '$\vee$', '$\rightarrow$', etc. was stated in a standard way. Additionally, the interpretation of operation of symbols of $\mathbf{D}$ 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. As is well known, there is an important difference between logical and Prolog negation [10]. However for the purpose of the implementation we adopt the convention that 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 $\cup$ 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.

Next, 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 knowledge-base 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.

4.0.1. 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 Figs. 9.a, 9.b or 9.c, and then types,

(74) *What is this?*

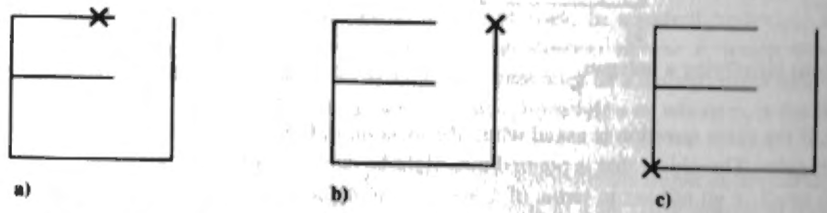

Fig. 9. Locations picked out by the user

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

(75) (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 Fig. 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.

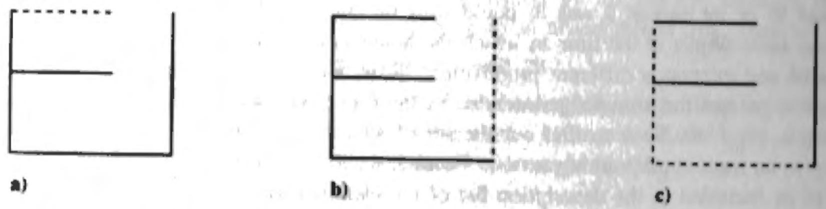

Fig. 10. Graphical feedback by highlighting the relevant components

The expressions *origin of* and *end of* in the natural language answers in (75.a) and (75.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 [8]. We can ask as well for the identity of objects represented by symbols of sort polygon. If the location in Fig. 11.a is pointed out, and the question

(76) *What is this?*

is asked, the answer is *A house*.

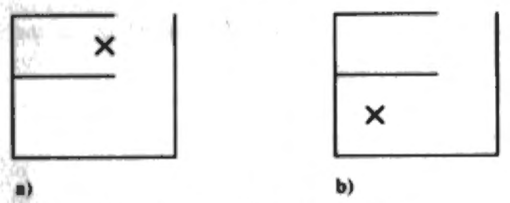

Fig. 11. Locations identifying a polygon

However, if the same question is asked when the location in Fig. 11.b is selected, there is a referential ambiguity. The object that is pointed out might be either the house or the room. Here, the system can produce an answer in terms of a geometrical default value and the answer could be, for instance, *a room*. However, the issue of answering these kind of question is rather complex, and further research is required in this matter [8].

5. The Parsing of Wire-frame Drawings in $\mathbf{L_d}$

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. 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 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 Fig. 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 best expresses 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. Second, we define a criterion for selecting the term that has to be included in the description list of the identified object. We discuss these in turn.

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(α)), 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* of 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*) = m_i where 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 Γ(*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,

(77) Γ_{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 Γ(*i*). We write

(78) $\Gamma(i) \vdash \phi$

to specify that the expression ϕ is derivable, or can be deduced, from the theory Γ(*i*).

If t_1, t_2 and t_3 are terms of $\mathbf{L_d}$, (79.1), (79.2) and (79.3) are valid inference rules:[9]

(79) (1) $\Gamma(i), t_1 = t_2 \vdash f(t_1) = f(t_2)$
(2) $\Gamma(i), t_1 = t_2 \vdash t_2 = t_1$
(3) $\Gamma(i), t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3$

For example, suppose that the drawing represented by the set Γ(*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

(80) Γ(i) ⊢ mark = position_of(e_join_at(wall_1, wall_2))

Let $MARK_{dot}(i)$ be the set {α in $\Gamma_{dot}(i)$ | Γ(i) ⊢ mark = position_of(α)}. This is, every expression α that denotes the same position as 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 $MARK_{dot}(i)$.

[9] (79.1) is not defined if ϕ is opaque.

(81) For every α and β in Γ_{line}(i) which are not parallel or collineal, if Γ(i) ⊢ mark = position_of(int(α, β)) then int(α, β)) is in $MARK_{dot}(i)$, where int is an intersection operator in clause (12) of the definition of **D**.

(82) For every α in Γ_{line}(i) if Γ(i) ⊢ mark = position_of(β(α)) then β(α)) is in $MARK_{dot}(i)$, where β is either end_of or origin_of.

(83) Nothing is in $MARK_{dot}(i)$ except as stipulated by clauses (81) and (82).

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:

(84) $Selected_{line}(i)$ = {α ∈ Γ_{line} | on(mark, α)}.

(85) $Selected_{polygon}(i)$ = {α ∈ Γ_{line} | in(mark, α)}.

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 Fig. 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:

(86) **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 rather they are 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 points to an overt dot on a drawing, he or she 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 havê a larger priority of selection than the 16 intersection cases in Fig. 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:

(87) 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.

(88) 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 Fig. 6 and is included in the description list of *house*, the set $MARK_{dot}(i)$ is

(89) **{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. 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 definition of the house, we define an additional construction line, namely c_line_2,[10] and it extends from the origin of wall_5 to the origin wall_1,

[10] c_line_1 was introduced in the definition of the room.

6. L.A. Pineda, E. Klein, and J. Lee, "GRAFLOG: Understanding Graphics Through Natural Language," *Computer Graphics Forum*, vol. 7, no. 2, 1988.

7. L.A. Pineda, "A. Compositional Semantic for Graphics," in *Eurographics'88 Conference Proceedings*, ed. D. Duce and P. Jancene, North-Holland, Amsterdam, 1988.

8. L.A. Pineda, *GRAFLOG: A Theory of Semantics for Graphics with applications to Human-Computer Interaction and CAD Systems*, 1989. Ph.D. thesis, University of Edinburgh

9. F.P. Preparata and M.I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, Berlin, 1985.

10. R. Reiter, "On Reasoning by Default," in *Readings in Knowledge Representation*, ed. R.J. Brachman & H.J. Levesque, pp. 402-410, Morgan Kaufman Publishers Inc., Los Altos, California, 1985.

11. M.I. Shamos, *Computational Geometry*, University Microfilms International, 1978. Ph.D. Thesis, Yale University

12. I.E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," in *AFIPS SJCC Proceedings*, pp. 329-346, 1963.

13. B. Tilove, "Set Membership Classification: A Unified Approach to the Geometric Intersection Problem," *IEEE Transactions on Computers*, vol. C-29, no. 10, 1980.

14. K. Weiler, "Polygon Comparation using Graph Representations," *ACM Siggraph'80 Conference Proceedings, Computer Graphics*, 1980.

representation Smalltalk and is the tool show a Programming.

[illegible]

fields. have on more technology There are definitely ve phases of design at the

top-down and theoretical AD will be

[illegible]