Logical Representations in Drafting and CAD Systems*

Luis A. Pineda and John R. Lee

EdCAAD and Centre for Cognitive Science
University of Edinburgh

Abstract

In this paper we present a logical language for representing and reasoning about drawings in the context of interactive computer graphics. We discuss the motivation for this kind of representation in graphics systems and its applications to geometric reasoning and constraint satisfaction. A formal definition of the language and its interpretation process is presented and illustrated. The theoretical machinery is then applied to modelling constraint-based drafting tasks. An application in which constraint satisfaction problems are expressed through the language in the course of interactive sesssions, and solved through symbolic inference, is presented and discussed. Finally, we discuss the functional architecture of a drafting system based on logical representations.

Key-words: Knowledge Representation in graphics, Drafting and Problem-solving, Constraint satisfaction, Graphics Semantics, Geometric reasoning.

1 Introduction

In this paper we discuss the definition and use of logical representations in drafting and CAD systems. In Section 2, we motivate the use of logical representations in interactive drafting tasks. We illustrate how a representational language can support the definition of graphical symbols and constraints in an integrated fashion, and can help to clarify the semantics of a drafting task. We also argue that logical representations offer a coherent framework for the solution of constraint satisfaction problems through symbolic inference, as an alternative to traditional techniques in which constraints are interpreted as equation systems that are in turn solved through numerical techniques. In Section 3, we define the syntax of a logical language for representing drawings. The relation between expressions of this language and the drawings that they represent is illustrated through a number of examples. In Section 4, we look at the definition of the interpreter of the language, and discuss the evaluation of graphical expressions in relation to a knowledge-base. In section 5, we review the interpretation of graphical expressions in a more theoretical light. There, we define the interpretation rules for expressions of the language in a declarative fashion,

*This paper has been written under the auspices of the project *Foundations for Intelligent Graphical Interfaces* (FIG), supported by Special Project Grant 8826213 from the Joint Councils Initiative in Cognitive Science/HCI.

and discuss the relation between symbolic and numerical aspects of the interpretation of graphical representations. In Section 6, we discuss the role of logical inference in interactive graphical tasks. We address the issue of "consistency" in the representations of drawings and their associated constraint sets. In Section 7, we show how the representational language can support the definition and interpretation of drafting rules for solving constraint satisfaction problems. In particular, we discuss the relation between symbolic inference and the interpretation of drafting intentions expressed by a human-user in the course of an interactive session. In Section 8, we present an architecture of a CAD system supporting logical representations, and discuss the relation between the representational scheme, the interface component of the program, and some related HCI issues. In Section 9, we conclude this paper with a summary of the main issues arising with the use of logical representations and we also point out some topics for further research.

The theory presented in this paper has been tested with a practical implementation. The architecture presented in Section 8 corresponds closely to an experimental computer program called Graflog [7, 11, 12, 13, 14, 15, 16, 17, 18]. The program has been implemented during the last four years, and has evolved through several stages. The current version is implemented in terms of two unix processes connected by unix-pipes. The first is a "C" program running *X windows*, and handles the external aspects of the interaction. The second is a Prolog program supporting the representational structures and interpreters of the system.

2 Graphical Descriptions and Constraint Satisfaction

In this section we motivate the use of a representational language expressive enough to refer to graphical symbols, relations and constraints in an integrated fashion. We explain that a drawing can be fully described by a set of terms that act like names or descriptions of the graphical symbols constituting a drawing. Constraints, on the other hand, are represented by sentences of such a language.

Consider Figure 1.1 in which we intend to "snap" $\mathtt{l}_2$ to $\mathtt{l}_1$ in such a way that a *t_join* is established. For making this definition we can place one extreme of $\mathtt{l}_2$ along $\mathtt{l}_1$ with the help of an interactive pointing device, and we have to express through a comand, or an alternative interactive facility, that we intend that constraint to hold. A representation of such a pair of lines can be stated by the following symbolic equations:

(1) $\mathtt{l}_1$ = line($\mathtt{d}_1$,$\mathtt{d}_2$)

(2) $\mathtt{l}_2$ = line($\mathtt{d}_3$,$\mathtt{d}_4$)

The constraint, in turn, can be represented by the expression,

(3) t_join($\mathtt{l}_2$,$\mathtt{l}_1$)

Now consider what happens if one extreme of line $\mathtt{l}_1$ is dragged to a new position as shown in Figure 1.2. In this state the *t_join* constraint does not hold, and a constraint satisfaction algorithm can be invoked to make an appropriate change, and produce the drawing in Figure 1.3.

Note, however, that the proposed solution for the change problem has a certain amount of indeterminacy. Expressions (1)–(3) tell us what the constraint satisfaction algorithm should do, but they do not specify how the task has to be accomplished. In fact,

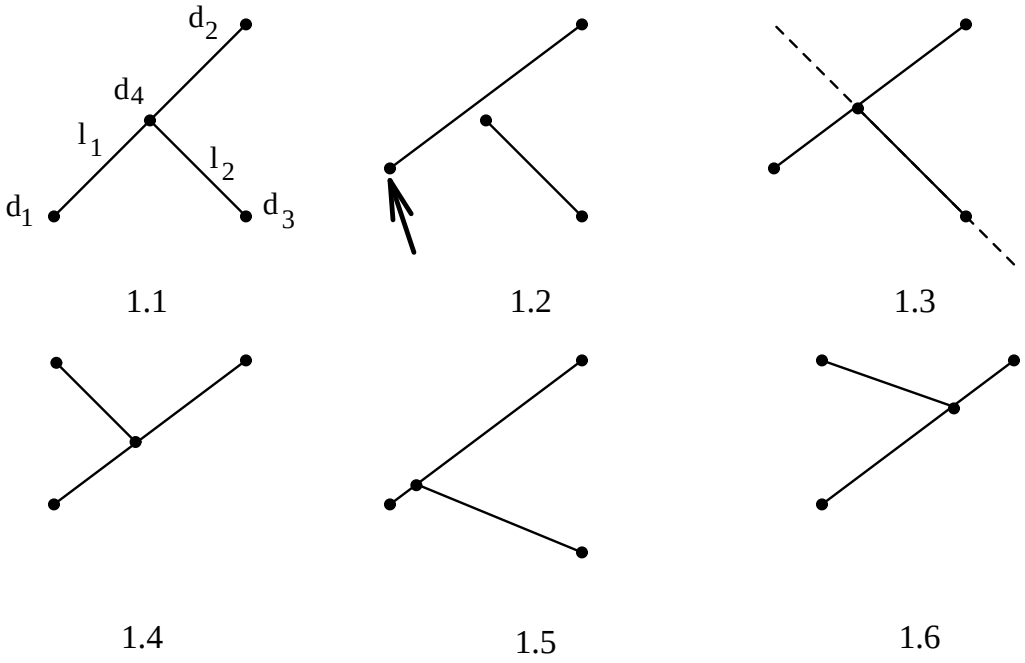


Figure 1: Definition of a "t_join"

it is possible to think of many alternative constructions, less intuitive perhaps, but which nevertheless satisfy the drawing description and constraints. Some such drawings are illustrated from Figure 1.4 to 1.6. Furthermore, one trivial solution is to reverse the change to the original configuration in Figure 1.1. Such a change would most probably be counter to the intention of the user who performed the original change comand, but of course, not necessarily. There might be an application in which such a cyclic behaviour is indeed expected by the human user.

This simple example illustrates a problem that has long undermined the use of geometric modellers. These systems tend to behave well when the expectations of the human-user match the expectations with which the program was specified and implemented. However, when the demands of the modelling task differ from the original model, systems tend to behave in erratic and idiosyncratic ways [19]. This has been labelled as the problem of "prescription" and it is argued that one goal for the next generation of drafting and CAD systems is to overcome this limitation [2].

It is clear that the more explicit the representation the less the "semantic contribution" made by the constraint satisfaction algorithm. To a certain extent we face a problem of naming. Here is where a logical representation comes into the picture: a language that is expressive enough will allow us to refer to and to describe not only the graphical symbols constituting a drawing but also its associated constraints in an integrated fashion. Consider the following alternative representation for the drawing in Figure 1.

(4) l_1 = line(d_1,d_2)
(5) l_2 = line(d_3,t_join(l_2,l_1))

The left side of expressions (4) and (5) is a name, and the right side is a description. The equality holds if the name refers to, or denotes, the same object in the drawing that is

referred to by the associated description. The name and the description must be of the same syntactic type for the equality relation to be meaningful. Here, unlike the assigment operator of many imperative languages, the symbol "=" means equal. Note that within the graphical description of a line, names referring to graphical objects of other kinds can be included. The symbols $\mathtt{d}_1$, $\mathtt{d}_2$ and $\mathtt{d}_3$ are in fact names of the corresponding graphical dots on the screen. The *t_join* symbol stands for a function of type $\mathtt{line} \times \mathtt{line} \rightarrow \mathtt{dot}$. The value of the *t_join* term in (5) in relation to the drawing in Figure 1.1 is of course the point in which $\mathtt{l}_2$ and $\mathtt{l}_1$ form a *t_join*. Suppose now that $\mathtt{l}_1$ is edited as shown in the transition from Figure 1.1 to 1.2. With our new representation, the transition from Figure 1.2 to 1.3 is accomplished implicitly: only the basic description of $\mathtt{d}_1$ –the dot moved– has to be updated, because the constraint is expressed as an invariant expression which is embedded in the definition of $\mathtt{l}_2$.

The use of symbolic descriptions can help us to visualise problems that might be overlooked if numerical representations were used. Definition (5) is, for instance, wrong. For suppose that we want to know what is the description of $\mathtt{l}_2$ in terms of its constituent parts. The first parameter of the *line* functor in (5) is a constant term, and as such, its value is immediately accessible to the interpreter; however, the value of the *t_join* term must be computed in terms of the values of $\mathtt{l}_1$ and $\mathtt{l}_2$. But, the value of $\mathtt{l}_2$ is what we are computing in the first place, so we come to an infinite regression. The problem is that in (5) we are defining a geometrical object in terms of itself.

Logical representations can help us to understand how "robust" a geometrical definition allowed by a graphical input device can be. Think again of the interactive definition illustrated in Figure 1.1. The question is this: where on $\mathtt{l}_1$ did we point to to get the *t_join* condition? Is it not the case that in such pointing action part of the target was defined on the drawing, but also part of the target was defined with the pointing device? Consider that it is not the same to point to a line with a dot –the graphics cursor– with the intention to select a line, as to perform the same action with the intention to select a dot on the line. This last pointing action is geometrically ambiguous because in the real space there is a set with an infinite number of dots lying in the vecinity of the position where the pointing device is placed, and we intend to pick up one of the dots, rather than the set. If the graphical pointing device is a dot, then we want the target of the pointing action to be a dot too, because in a successful pointing action there must be a match between the parameter of the pointing device and a parameter of the drawing.[1]

Next, we show one way of making a sound interactive definition of our *t_join* condition. In Figure 2.1, we define in addition to $\mathtt{l}_1$ the construction line $\mathtt{l}_c$. In Figure 2.2 we add the joining line $\mathtt{l}_2$ in such a way that $\mathtt{l}_2$ is defined in terms of an unconstrained dot and a function that depends on $\mathtt{l}_1$ and $\mathtt{l}_c$. The state of the representation of the drawing could then be expressed as follows,

(6) $\mathtt{l}_1$ = line($\mathtt{d}_1$,$\mathtt{d}_2$)

(7) $\mathtt{l}_c$ = line($\mathtt{d}_3$,$\mathtt{d}_4$)

[1] More generally, in order to pick unambiguously a graphical object of a certain sort in terms of its geometry, that is matching a geometrical property of the drawing with a geometrical property of the pointing device, we have to point it out with a graphical cursor of the same sort. Otherwise, conventions that are not always clear might have been embedded in the program's code. Furthermore, a pointing action does not depend only on the geometry, but also on the conceptual interpretation of the drawing and the user intentions. For a discussion of these issues see, for instance, [11, 12, 14, 16, 7].

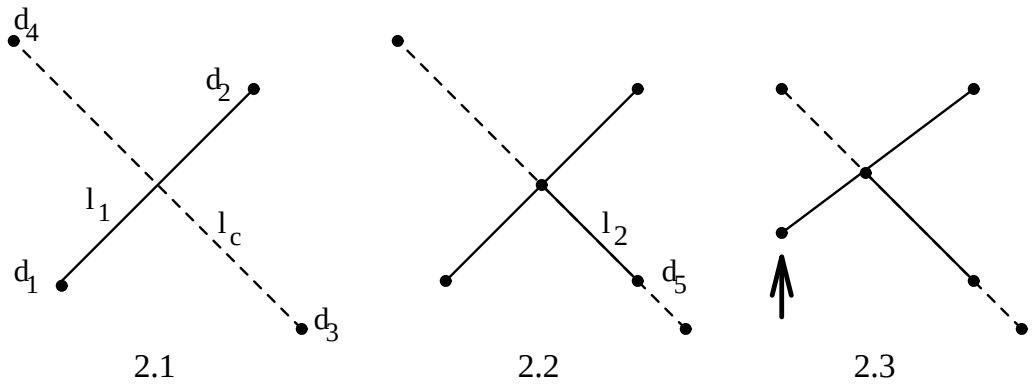


Figure 2: A sound definition of a "t_join"

(8) $\texttt{l}_2$ = line($\texttt{d}_5$,intersect($\texttt{l}_1$,$\texttt{l}_c$))

where every term is either basic or can be computed in terms of its constituent parts. Now we are in a position to illustrate the expressive power of the new definition. If we edit $\texttt{l}_1$ as shown in Figure 2.3, only the value of the basic constant $\texttt{d}_1$ has to be changed in the representation. The expression through which the attachment condition is referred to remains the same. Here, the satisfaction of the *t_join* constraint is not mediated by an idiosyncratic process.

The use of logical descriptions gives additional expressive power over alternative parametric design techniques in which the "parameters" are restricted to being constant values. However, there is no freedom without responsibility. This is so because a well-formed description can have a proper referent in some interactive states while lacking such a referent in others. Consider now the representation in (6)–(8) in relation to the change from Figure 3.1 to 3.2. Here, one extreme of $\texttt{l}_c$ has been moved so that there is no longer

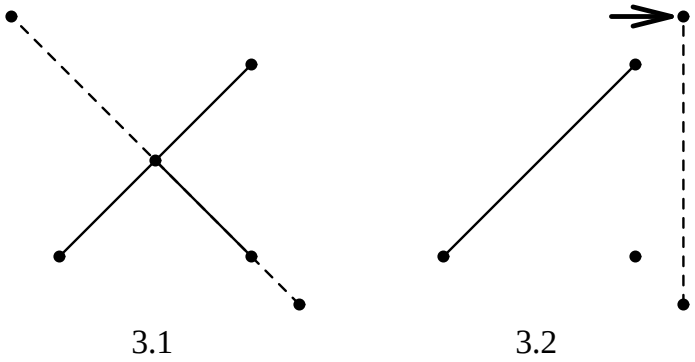


Figure 3: Partial Interpretation

an intersection between $\texttt{l}_1$ and $\texttt{l}_c$. The question is then what happens with $\texttt{l}_2$? Notice that the definition of $\texttt{l}_2$ is invariant in the change but the term intersect($\texttt{l}_1$,$\texttt{l}_c$) has no value in the drawing in Figure 3.2; subsequently, neither has $\texttt{l}_2$. More precisely, the range of the function denoted by *intersect* is not defined for every member of its domain: it is a partial function. However, the free extreme of $\texttt{l}_2$ –$\texttt{d}_5$– is well-defined in Figure 3.2, and we can depict it on the screen if we wish to do so. Consider that most drafting systems

would not allow the transition in Figure 3 because that would lead to inconsistencies of the data-structures representing the drawings. In such a situation we would probably get a message saying "invalid operation: try again". However, such a strategy is prescriptive because the system is forcing predefined drafting norms. Such conventions can be proper in some situations, but there might be contexts in which global drafting intentions of the human-user are more relevant to the drafting task than a local constraint that can be provisionally left unsatisfied. In a system supporting logical representations the interactive change illustrated in Figure 3 can be allowed, provided the user is properly notified of the irregular condition. In fact, in our implementation the system can move in and out of partially defined states of the interaction while preserving semantic consistency. This is another point worth thinking about. In traditional constraint satisfaction the notions of "inconsistency" and "partiality" are not often distinguished.

In summary, logical representations can be thought of as functions from drawing states to drawings: a given set terms is an abstraction over the geometrical and topological properties of a set of symbols, and for every drafting state such properties are assigned specific values, and the symbols can be depicted. Alternatively, logical representations can be thought of as functions from basic graphical symbols to composite graphical structures. This view of drawings provides a general way to look at problems related to constraint satisfaction. Since Sutherland's Sketchpad program [21] the problem of satisfying a number of geometrical and topological constraints on drawings has been thought of in terms of representing constraints as equational systems. Following Sutherland's line, many drafting and CAD programs solve constraint systems through local propagation, relaxation and Gaussian elimination.[2] However, the literature in constraint satisfaction is often unclear in distinguishing the problem's definition from its solution method. Constraints need not be conceptualised necessarily as equation systems which are solved through numerical techniques. Constraints can alternatively be thought of as expressions of a representational language, and constraint satisfaction problems can be solved through inferential techniques supported by a clear naming policy. In our view, in constraint satisfaction in drafting and CAD systems we are interested in constructing drawings through the definition of a number of graphical symbols, the specification of a set of **invariant** graphical properties and relations that ought to hold in a design task, and an interactive drafting process in which the specification of drawing and constraints is refined. This is precisely what logical representations can help us to do. In the rest of this paper we address the definition, interpretation and use of logical representations in the process of drafting.

3 Definition of an Functional Language

One important feature of graphical representations is that they are composed of a finite number of symbols. In fact, for every finite collection of graphical symbols on the screen there is a graphical language implicitly defined. Interestingly enough, such a language can be made explicit and generalised through techniques employed in the formal specification of programming languages.

In the formulation of a language we first define all symbols that can be a part of

[2] For an introduction to constraint satisfaction and constraint programming languages see, for instance, Leler[9].

an expression. We then specify the rules for the construction of well-formed expressions. Finally, we define the rules for interpreting the expressions of the language. Next we address the definition of a graphical language, then we come to its interpretation process.

3.1 Definition of a graphical language

Following Goguen et. al. [6] and Wang [22] we define every constant symbol of the language by specifying the symbol's *rank, arity* and *sort*. The arity is a string of symbols stating the number, order and sort of the arguments that an operator symbol has. If a functor denotes a function, the arity specifies the type of the function's domain and the sort the type of the function's range. For instance, the symbol *t_join* shown above has arity *line line* and sort *dot*. The rank is the specification of the arity and sort taken together; i.e. the rank of *t_join* is the string *line line, dot*. For basic constants, the arity is the empty string ϵ.

A language can have a large number of symbols of the same rank and we define $\Sigma_{w,\ s}$ to be the set of all symbols of rank w and sort s. The family of such sets contains all symbols of the language and will be referred to as the S-sorted signature Σ.

For example, we might take the set $S = \{integer\}$, and $\Sigma_{w,\ s}$ to be empty except for $\Sigma_{\epsilon,\ integer} = \{0\}$ and $\Sigma_{integer,\ integer} = \{\mathbf{succ}\}$. That is, **succ** is to be interpreted as the unary operator "succesor" 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. This system can graphically depicted as shown in Figure 4.

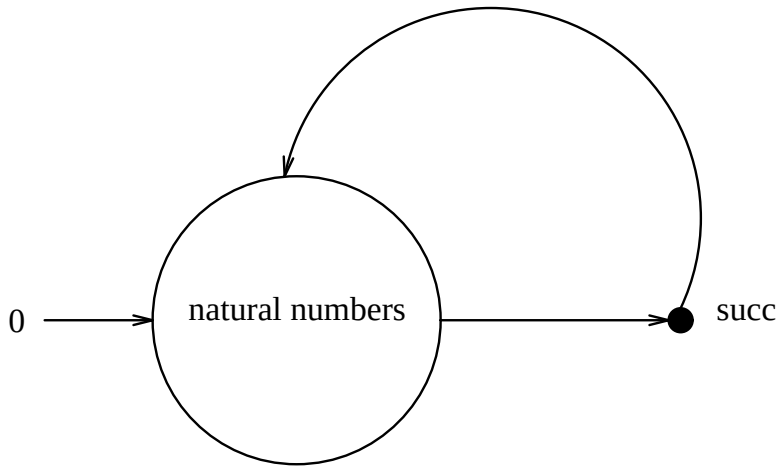


Figure 4: Graphical Representation of a language

Next, we define the language **G** for modelling the graphics symbols and operations illustrated in the previous section. We include a number of additional symbols for illustrating the expressive power of the language.

The language **G** is defined as follows:

1. The set of sorts $S_G = \{bool,\ real,\ real\_pair,\ dot,\ line,\ arc,\ circle,\ path,\ polygon\}$.

2. The set of *error* sorts $S_e = \{bool_e,\ real_e,\ real\_pair_e,\ dot_e,\ line_e,\ arc_e,\ circle_e,\ path_e,\ polygon_e\}$. For every sort there is a corresponding set of abstract error elements

of that sort. In the interpretation, such elements correspond to the denotation of graphical expressions in states in which the normal denotations are not defined.

3. An orderering relation $\preceq$ on the sorts is defined as follows:

   (a) For every pair of types s and s_e, $s_e \preceq s$.

   (b) *line* $\preceq$ *path, arc* $\preceq$ *path.*

Intuitively, $\alpha \preceq \beta$ expresses that the set of objects denoted by expressions of sort α is a subset of the set of objects denoted by expressions of sort β. Every error element of a sort s_e is also of sort s, but no normal element of sort s is in s_e. Similarly, lines and arcs are subsets –disjoint– of the set of paths.

Next, we define the basic constants of the language **G**.

1. The logical constants $\Sigma_{\epsilon,\ bool} = \{\text{false, true}\}$.
2. $\Sigma_{\epsilon,\ real}$ is an infinite set of numerals
3. $\Sigma_{\epsilon,\ real\_pair}$ is an infinite set of ordered pairs of numerals.
4. A finite set of dots $\Sigma_{\epsilon,\ dot} = \{\mathtt{d}_1, \mathtt{d}_2, ...\}$.
5. A finite set of lines $\Sigma_{\epsilon,\ line} = \{\mathtt{l}_1, \mathtt{l}_2, ...\}$.
6. A finite set of arcs $\Sigma_{\epsilon,\ arc} = \{\mathtt{a}_1, \mathtt{a}_2, ...\}$.
7. A finite set of paths made out of sequences of lines and arcs: $\Sigma_{\epsilon,\ path} = \{\mathtt{k}_1, \mathtt{k}_2, ...\}$.
8. A finite set of circles $\Sigma_{\epsilon,\ circle} = \{\mathtt{c}_1, \mathtt{c}_2, ...\}$.
9. A finite set of polygons $\Sigma_{\epsilon,\ polygon} = \{\mathtt{poly}_1, \mathtt{poly}_2, ...\}$.

Error elements will be named by a constant of the form e_{s_i}, where s is a basic type and i an integer.

We come now to the definition of the functor symbols of **G**. First, we define a basic constructor for every graphical sort as follows:

1. Dot: $\Sigma_{real\_pair,\ dot} = \{\mathbf{dot}\}$.
2. Line: $\Sigma_{dot\ dot,\ line} \cup \Sigma_{dot\ real,\ line} = \{\mathbf{line}\}$.
3. Arc (centre, radius, from-angle, to-angle): $\Sigma_{dot\ real\ real\ real,\ arc} = \{\mathbf{arc}\}$.
4. Circle: $\Sigma_{dot\ dot,\ circle} \cup \Sigma_{dot\ real,\ circle} = \{\mathbf{circle}\}$.
5. Path: $\Sigma_{path\ path,\ path} = \{\mathbf{path}\}$
6. Polygon: $\Sigma_{path,\ polygon} = \{\mathbf{polygon}\}$.

We proceed now to define a number of other constructors and selectors of the language:

1. Position of a dot: $\Sigma_{dot,\ real\_pair} = \{\mathbf{position}\}$.

2. Construction dot: $\Sigma_{dot\ dot,\ dot} = \{\textbf{cons\_dot}\}$.

3. Real properties of lines: $\Sigma_{line,\ real} = \{\textbf{angle, length}\}$.

4. Extreme dots of lines: $\Sigma_{line,\ dot} = \{\textbf{origin, end}\}$.

5. Angle of line: $\Sigma_{line\ line,\ real} = \{\textbf{angle}\}$.

6. Intersections: $\Sigma_{line\ line,\ dot} = \{\textbf{t\_join, e\_join, intersect, project}\}$.

7. Dot on line: $\Sigma_{dot\ line,\ bool} = \{\textbf{on}\}$.

8. Boolean properties of lines: $\Sigma_{line,\ bool} = \{\textbf{horizontal, vertical}\}$.

9. Boolean relations: $\Sigma_{line\ line,\ bool} = \{\textbf{parallel, perpendicular}\}$.

10. Arc selectors: $\Sigma_{arc,\ dot} = \{\textbf{origin, end, centre}\}$.

11. Circle centre selector: $\Sigma_{circle,\ dot} = \{\textbf{centre}\}$.

12. Circle radius selector: $\Sigma_{circle,\ real} = \{\textbf{radius}\}$.

13. Tangent of a circle at a given angle: $\Sigma_{circle\ real,\ line} = \{\textbf{tangent}\}$

14. Equality: For every sort S, $\Sigma_{s\ s,\ bool} = \{=\}$

The language can, of course, be augmented with other geometrical types, additional constructors and selectors, arithmetic functor terms, and other utilities. In addition, we include the following definitions to embed the language **G** in a first-order predicate logical language.

1. Variables: For every sort S, there is a countable infinite set of variables $V_s = \{x_{s_0},\ x_{s_1}, ...\}$.

2. Quantifiers: For every sort s, $\forall_s$, $\exists_s$, each of rank $\Sigma_{bool\ s,\ bool}$.

3. The logical symbols $\Sigma_{bool\ bool,\ bool} = \{\textbf{and, or,} \rightarrow\}$.

4. The negation symbol $\Sigma_{bool,\ bool} = \{\textbf{not}\}$.

5. The auxiliary symbols "(" and ")".

Next, we state the formation rules of this term language:

1. Every constant of sort s is a well-formed term of **G**.

2. 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 well-formed term of **G**.

3. If $\forall_s$ is a quantifier of sort s, u a variable of sort s, and ϕ a term of sort *bool*, then $\forall_s u\phi$ is a term of sort *bool*. The same for the existential quantifier $\exists_s$.

4. Nothing else is a well-formed term of **G**.

We can now formulate a large number of expressions referring to graphical symbols and relations that arise in graphical configurations. The constructors and selectors defined in the language allow us to define geometrical objects in terms of other objects in a flexible fashion. For the definition of expressions, we just have to verify that the term's arity and sort are of consistent types. In Figure 5.1 to 5.6, a number of constructions are shown to illustrate the expressivity of the language. In each drawing, basic symbols are drawn with light lines, while the composite symbol denoted by the corresponding expressions in (9)–(14) are depicted in bold. In Figure 5.1, the dot in the intersection between the two

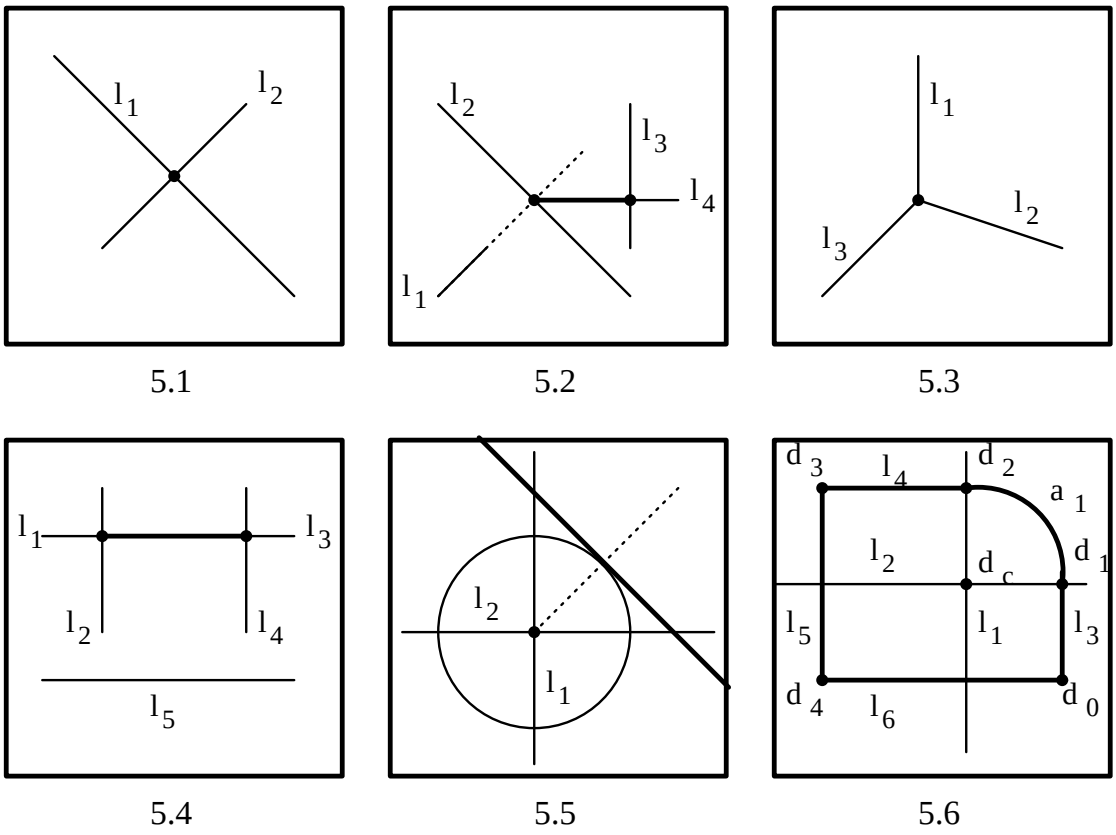


Figure 5: Drawings denoted by expressions of **G**

lines is denoted by (9).

(9) `intersect(l1,l2)`

We can also denote the position of the dot through the expression

`position(intersect(l1,l2))`

which is of sort *real_pair*. Note that the expression denoting the dot can be depicted, but the dot's position is an abstract property.

Figure 5.2 illustrates the construction of a line in terms of four basic lines. The construction is denoted by the term (10) of sort *line*. The new line is defined from the intersection of the "projection" of $\mathtt{l}_1$ into $\mathtt{l}_2$, on the one hand, and the "t_join" intersection

between $\mathtt{l}_3$ and $\mathtt{l}_4$.

(10) $\mathtt{line(project(l_1,l_2),t\_join(l_3,l_4))}$

Figure 5.3 illustrates the definition of a dot as a function of more than two lines through the *cons_dot* functor term. Expression (11) takes two dots as its arguments; if the position of both dots is the same, the value of the term is also a dot in the same position; otherwise, the "cons_dot" term has no interpretation in the state.

(11) $\mathtt{cons\_dot(e\_join(l_1,l_2),e\_join(l_2,l_3))}$

Expression (12) and Figure 5.4 illustrate a boolean relation between a basic and a composite graphical object. The composite line is defined from the "t_join" between $\mathtt{l}_1$ and $\mathtt{l}_2$ to the "t_join" of $\mathtt{l}_3$ and $\mathtt{l}_4$. The expression as a whole states that such a line and $\mathtt{l}_5$ are parallel.

(12) $\mathtt{parallel(line(t\_join(l_1,l_2),t\_join(l_3,l_4)),l_5)}$

Figure 5.5 and expression (13) illustrate the construction of a line that is tangent to a circle at some specified angle. The centre of the circle is in turn defined as the intersection of lines $\mathtt{l}_1$ and $\mathtt{l}_2$, and a given radius.

(13) $\mathtt{tangent(circle(intersect(l_1,l_2),}\rho\mathtt{),}\theta\mathtt{)}$

Finally, (13) denotes a composite polygon, as illustrated in figure 5.6. The polygon is constituted by a sequence of paths, one of which is the arc $\mathtt{a}_1$. The center of $\mathtt{a}_1$ is itself defined as a function of the intersection of the axis $\mathtt{l}_1$ and $\mathtt{l}_2$, and the angles in relation to the horizontal that determine the origin and end extremes of the arc. The definition of the polygon is as follows:

(14) $\mathtt{polygon(k_4)}$

where,

$\mathtt{a_1 = arc(intersect(l_1,l_2),}\rho\mathtt{,}\theta_1\mathtt{,}\theta_2\mathtt{)}$

$\mathtt{k_1 = path(l_3,a_1)}$

$\mathtt{k_2 = path(k_1,l_4)}$

$\mathtt{k_3 = path(k_2,l_5)}$

$\mathtt{k_4 = path(k_3,l_6)}$

Note that the functor *path* denotes a function of type **path** $\times$ **path** $\rightarrow$ **path** but some of the arguments of the *path* constructors in (14) are of sorts *line* and *arc* yielding the corresponding terms syntactically ill-formed. However, recall the ordering relation on the sorts *line* $\preceq$ *path* and *arc* $\preceq$ *path*. That is to say that lines and arcs are restricted kinds of paths, and if two paths can combine to form a path, then lines and arcs can combine to form paths too. The ordering relation $\preceq$ is a device to increase the expressive power of the system, allowing us to define a small set of basic constants of differents sorts explicitly, while constants of other sorts are implictly defined. The explicit definition of the *path* constructor of rank **path** $\times$ **path** $\rightarrow$ **path** and the relation $\preceq$ implicitly define nine different constants of type $\alpha \times \beta \rightarrow$ **path** in which α and β are either *line*, *arc* or *path*.

With this example we finalise the definition of a graphical language, and the discussion of its expressive power. Next, we come to its interpretation process.

4 Interpretation of the language G

There are two alternative ways of looking at the interpretation of expressions of **G**. In this section, we direct our attention to the computer program that can take an expression of **G** as an input and compute its value in terms of the representation of the current drawing on the screen. In Section 5, we focus on an alternative declarative formulation of the interpretation process. There, we introduce the notion of semantic algebra, and the notion of interpretation in relation to a theoretical model.

In Figure 6, the interpreter of the language G as a computer program is illustrated. First, note that there is a geometrical knowledge-base (KB_G) that holds the logical representation of the drawing at any state of the interactive session. In the modelling process, basic constants correspond to the graphical symbols to be considered as "the independent variables of the modelling process". A composite term, on the other hand, captures one or more functional relations and the term's value depends on its constituent parts. The

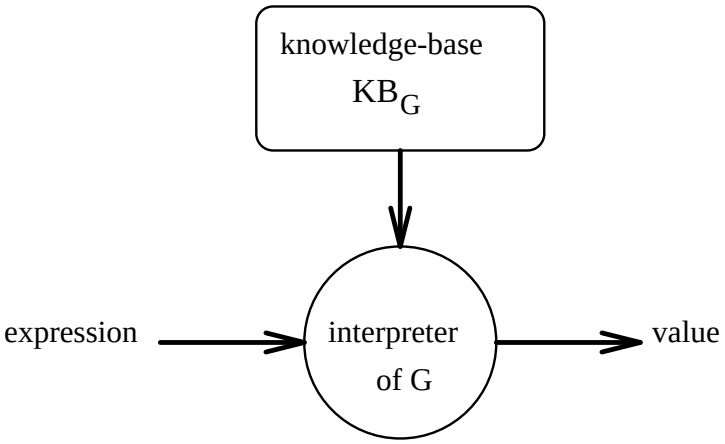


Figure 6: The interpreter of expressions of **G**

process of interpretation can be thought of as the substitution of a term by its value in the evaluation state. Intuitively, the output of the interpretation process is a "drafting command", an initialised description of the graphical object to be depicted. We illustrate this notion with the help of an example. Suppose that the representation in KB_G of the drawing in Figure 5.4 is as follows,

(15) l_1 = line(d_1,d_2)

l_2 = line(d_3,d_4)

l_3 = line(d_5,d_6)

l_4 = line(d_7,d_8)

l_5 = line(d_9,d_{10})

l_a = line(t_join(l_1,l_2),t_join(l_3,l_4))

As can be seen in (15), every entry in KB_G consists of a definition in which a constant symbol of a graphical sort is associated to a graphical constructor term of the same sort

through an equality relation. Note that some parameters of the graphical constructors are basic constants while others are composite terms. If all parameters of a constructor term are basic constants the term will be referred to as a basic description or *normal form*. If a graphical constructor has as parameters only basic descriptions, such a constructor term will be a normal form too. In general, a basic description of a graphical sort cannot be substituted by a more simple description and the corresponding graphical symbol can be depicted directly on the screen. With this new definition we can reformulate the process of interpretation of an expression as follows: the interpretation of a term consists in replacing the term by its basic description in relation to the current state of KB_G. Evaluating an expression consists in replacing it by its normal form.

The value of $\mathtt{l}_1$ in relation to (15) is given through a basic definition.[3] However, the value of $\mathtt{l}_a$ can be reduced in relation to KB_G as shown in (16).

(16) $\mathtt{line(t\_join(l_1,l_2),t\_join(l_3,l_4))}$

$\mathtt{line(t\_join(line(d_1,d_2),line(d_3,d_4)),t\_join(line(d_5,d_6),line(d_7,d_8)))}$

$\mathtt{line(d_a,d_b)}$

where $\mathtt{d}_a$ and $\mathtt{d}_b$ are computed through geometry.

Next, we define the interpretation of a term ϕ of the language **G** as follows:

1. If ϕ is a basic description the value of ϕ is ϕ itself.
2. If ϕ is a basic constant of a graphical sort the value of ϕ is the value of its associated basic description.
3. If ϕ is a graphical description –not basic– **reduce** the parameters of the description and return its basic description.
4. If ϕ is any other term, **reduce** the term's parameters and return the term's value.

Note that the evaluation of a well-formed expression not only depends on its syntactic definition and interpretation rules. It also depends on the semantic consistency of the representation. It is indispensable that every expression of a graphical sort has a basic description; however, to assess whether all expressions satisfy this requirement for an arbitrary geometrical knowledge-base is by no means a trivial problem. Here, we assume that the knowledge-base is well-defined, and that there are no implict cyclic definitions of graphical objects.

4.1 Partial interpretation

In this section we explain the interpretation of expressions in graphical states in which denotation is not defined. Consider again Figure 3, and its representation as follows,

(17) $\mathtt{l_1 = line(d_1,d_2)}$

$\mathtt{l_c = line(d_3,d_4)}$

$\mathtt{l_2 = line(d_5,intersect(l_1,l_c))}$

[3] For simplicty, we consider terms of sort dot as basic constants in all examples.

If all arguments of a term have a value in a given graphical state, the term as a whole might or might not have a value. If it has a value, the expression denotes a normal element of sort s. However, if the term has no normal denotation, then its value is an object of sort s_e.

Consider the evaluation of (18) in relation to Figure 3.2.

(18) $\texttt{line(d}_5\texttt{,intersect(l}_1\texttt{,l}_c\texttt{))}$

The functor *line* is a basic constructor term, and according to the interpretation rule 3 above all its arguments have to be reduced to a normal form. The term $\texttt{d}_5$ is itself in normal form, but the *intersect* term has to be reduced. The value of $\texttt{intersect(l}_1\texttt{,l}_c\texttt{)}$ depends on whether the equations representing the intersection condition (i.e the parametric equations of two vectors in the space) have a solution or not in such a state. In this particular case there is no such value, and the term has an object of sort dot_e as its value. Expression (18) is then reduced to,

(19) $\texttt{line(d}_5\texttt{,}e_{dot_i}\texttt{)}$

Expression (19) is syntactically well-formed. Note that the functor *line* is not the basic line constructor, but a constant of rank *dot* dot_e, *line*, which is implied in the language due to the ordering relation on the sorts ($dot_e \preceq \texttt{dot}$). The value of (19) is an object in the set $line_e$ $-e_{line_j})-$ which is in turn the value of $\texttt{l}_2$ in relation to Figure 3.2.

Next, we postulate the rules for handling partial interpretation of expressions of **G**. These are,

1. If all arguments of a functor term are normal elements of their corresponding sorts, the value of the term is a normal element of its sort.

2. If all arguments of a functor term are normal elements of their corresponding sorts, but the value of the term is itself an error element of the term's sort, we place the term in the error-stack for further inspection and the production of an error message.

3. If an argument of a term is an error element of its corresponding error sort the value of the term is an error element of the term's sort.

These rules complement the definition of the interpreter as a computer program. Next, we look at the interpretation process in a more abstract and theoretical fashion.

5 Declarative specification of the interpretation process

The view of the interpretation process shown above emphasises the "symbolic" aspect of graphical representations, in which complex expressions are partitioned into its basic constituent symbols. However, we have yet to explain how the geometrical algorithms associated to geometrical computations are defined, and how these algorithms are systematically used in the interpretation of expressions of **G**. We assumed above, for instance, that an expression of the form $\texttt{intersect(l}_1\texttt{,l}_2\texttt{)}$ can be substituted by a basic description like $\texttt{d}_{int}$, as if they were equivalent constants. However, such substitution has to be mediated by the process of computing a geometrical algorithm. More generally, the question that we focus on in this section is how to relate in a systematic manner the finite world

of graphical symbols and relations that constitute a given drawing, and that we use when we think of drawings as "symbol systems", with the infinite domain of real numbers for representing the space and the numerical values of geometrical properties and relations. The relation between finite and infinite domains can be better understood in terms of a *declarative specification* of the interpretation process.

We start this discussion by emphasising the distinction between a graphical symbol and its name or its description. We define the symbol itself as the semantic value, or denotation, of the symbol's name or description. The former is an object of the world, the thing to be represented, while the latter is a linguistic object, the thing doing the representation. To capture the systematic relations between representing and represented objects we use the semantic framework introduced by Montague, as presented by Dowty et al. [5]. We define the *semantic interpretation* of the language **G** in relation to a *model*.

A *model* M for **G** is an order tuple $\langle G, I, F \rangle$, where $G = < G_s >_{s \ \varepsilon \ S}$ is an S-indexed family of non-empty sets (i.e. the sets of graphical symbols of different kinds belong to this family), I is a set of states $i_1, ..., i_n$ (the states of the interactive session), and F is an interpretation function whose domain is the set of non-logical constant symbols of **G**, (the numerals and basic constants of the graphical sorts), and whose range is in G as defined below.

For the interpretation of variable symbols, we define the function g which has as its domain the set of all variables, and as its range a member of $< G_s >$ for every variable of sort s.

Now, we introduce a notational convention: The semantic value of an expression α with respect to a model M, a state $i \, \varepsilon \, I$, and a value assignment g is expressed as,

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

Next, we define the interpretation of expressions of **G**:

1. If α is a constant or a basic description of sort s, then $[[\alpha]]^{M,i,g} = [F(\alpha)](i)$.[4]

2. If f is an operation symbol of rank $w \, s$, where $w = s_1, ..., s_n$, then $[[f]]^{M,g}$ is a function with domain in $G_{s_1} \times ... \times G_{s_n}$ and range in G_s. Note that the interpretation of f does not depend on the state i, because the functions denoted by logical and geometrical operation symbols are the same in every state. For a geometrical functor term, this function is the function computed by a computational geometry algorithm associated to the functor term. For the basic constructors of every graphical sort, this function verifies that the geometry of a symbol is properly defined.[5] For a logical constant, this function is stated in its associated truth table.

3. If ϕ is a term of the form $f(t_1, ..., t_n)$, where f is an operation symbol of rank $w \, s$, where $w = s_1, ..., s_n$, and $t_1, ..., t_n$ are terms of sorts $s_1, ..., s_n$ then

$$[[f(t_1, ..., t_n)]]^{M,i,g} = [[f]]^{M,g}([[t_1]]^{M,i,g}, ..., [[t_n]]^{M,i,g}]).$$

4. If ϕ and ψ are terms of sort s the $[[\phi = \psi]]^{M,i,g}$ *is true* if and only if $[[\phi]]^{M,i,g}$ is the same as $[[\psi]]^{M,i,g}$.

[4] The value of $[F(\alpha)]$ in the state i.

[5] For instance, the polygon constructor verifies that the argument of sort path forms a closed curve in which the constituent segments do not intersect each other.

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

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

The semantic rules (1)–(6) state precisely the interpretation of expressions of **G**, and they correspond to the more intuitive procedural specification presented in the previous section. Rule 1 simply states that the referent of a graphical symbol's name or its basic description is the graphical symbol itself. Definition (2) states the interpretation of functor or operation symbols of the language. Every operation symbol is interpreted as a function, which of course can be further specified in formal terms. But given that geometrical properties can be computed by different computational geometry algorithms, we just make the abstraction that the meaning of an operation symbol is some function. The definition of the actual functions for a particular implementation would depend, of course, on the nature of the application, and the available computational geometry algorithms. Rule (3) is the semantic counterpart for the syntactic rule through which well-formed terms are defined, and it simply states that the interpretation of a functor term is the application of the function denoted by the operator symbol of the term, to the objects denoted by the argument terms. Rule (4) specifies the interpretation of the equality relation, and it is the device that we use for interpreting basic constants of graphical sorts. For instance, the interpretation of a constant like $\texttt{poly}_1$ is a symbol on the screen, but such a polygon can only be depicted in terms of its associated description; then, the interpreter is implemented in such a way that when the interpretation of a symbol's name is required, its associated description is evaluated, using (4) and (3). Rules (5) and (6) specify the interpretation of quantified terms. Rule (5) specifies that a universally quantified formula is true if it is satisfied by all the possible different instantiations of the bound variable in relation to the model M in the current state. Rule (6) requires that at least one such instantiation must hold. In our model, the universe of discourse consists of the symbols that are named in the knowledge-base either by a basic constant or a description.

The semantic rules cover as well the partial interpretation of expressions of **G**. Given that the addition of error elements in every sort allows us to think of the functions associated with every functor term as total functions, the present formulation is general enough: in every situation the interpretation of an expression is a normal or an error element of its sort.

Now, we turn to the relation between the theoretical notion of *Model* and its implementation in terms of the objects in the logical representation. In the same way that expressions of **G** are evaluated in relation to the representation in the data-base (the set of expressions denoting the drawing's constituents), they have a semantic interpretation in relation to the model M. Here, we make the abstraction that the finite set of graphical symbols of every sort s in a particular state of the data-base is the set G_s of the model M at that state. On the other hand, the computational geometry functions associated with the operation symbols of **G** have domains and ranges of infinite cardinality (the set of real and order pair of reals) and such functions are the same for every drafting state.

Next, we illustrate with the help of an example how the interpretation rules are used. Consider the expression (5) in relation to the drawing in Figure 3.1,

$$\texttt{line(d}_5\texttt{,intersect(l}_1\texttt{,l}_c\texttt{))}$$

The interpretation proceeds as follows:

1. Rule 3: Application of `line`,

   $$[\![\texttt{line(d}_5\texttt{,intersect(l}_1\texttt{,l}_c\texttt{))}]\!]^{M,i,g} = [\![\texttt{line}]\!]^{M,g}([\![\texttt{d}_5]\!]^{M,i,g}, [\![\texttt{intersect(l}_1\texttt{,l}_c\texttt{)}]\!]^{M,i,g})$$

2. Rule 1: Eval $\texttt{d}_5$,

   $$[\![\texttt{d}_5]\!]^{M,i,g}$$

   is given by the interpretation function F of the model M.[6]

3. Rule 3: Application of `intersect`,

   $$[\![\texttt{intersect(l}_1\texttt{,l}_c\texttt{)}]\!]^{M,i,g} = [\![\texttt{intersect}]\!]^{M,g}([\![\texttt{l}_1]\!]^{M,i,g}, [\![\texttt{l}_c]\!]^{M,i,g})$$

   (a) Rule 4: Substitution of constant $\texttt{l}_1$ by its basic description, (similarly for $\texttt{l}_c$)
   $[\![\texttt{l}_1]\!]^{M,i,g} = [\![\texttt{line(d}_1\texttt{,d}_2\texttt{)}]\!]^{M,i,g}$

   (b) Rule 3: Application of `line`,
   $[\![\texttt{line(d}_1\texttt{,d}_2\texttt{)}]\!]^{M,i,g} = [\![\texttt{line}]\!]^{M,g}([\![\texttt{d}_1]\!]^{M,i,g}, [\![\texttt{d}_2]\!]^{M,i,g})$
   where the dots are basic constants and $\texttt{line(d}_1\texttt{,d}_2\texttt{)}$ is a well-defined basic description.

   (c) Rule 3: Application of geometrical algorithm associated to *intersect*:
   $[\![\texttt{intersect}]\!]^{M,g}([\![\texttt{line(d}_1\texttt{,d}_2\texttt{)}]\!]^{M,i,g}, [\![\texttt{line(d}_3\texttt{,d}_4\texttt{)}]\!]^{M,i,g}) = [\![\texttt{d}_{int}]\!]^{M,i,g}$
   where $[\![\texttt{d}_{int}]\!]^{M,i,g}$ has a normal denotation.

4. Rule 3: Substitution (3) in (1) and application of geometrical algorithm associated to *line* (which verifies that the two dots are well-defined and whose positions are not the same):

   $$[\![\texttt{line}]\!]^{M,g}([\![\texttt{d}_5]\!]^{M,i,g}, [\![\texttt{d}_{int}]\!]^{M,i,g})$$

   which is a well-formed basic description.

Consequently, (5) is a well-defined line from $\texttt{d}_5$ to $\texttt{d}_{int}$ in relation to Figure 3.1. Notice that the expression is recursively partitioned into its constituent parts: a symbolic operation. This reduction is governed by the interpretation rules specified in the semantic algebra. However, the substitution in 3.c is a numerical process which is embedded within the symbolic interpretation. Note that the application of the function denoted by a basic

[6] Although in our grammar, dots have basic descriptions in terms of order-pairs of reals, for clarity we consider dots as primitive constants in this example.

geometric functor, –which is computed by the geometrical algorithm associated to the functor symbol– is preceded by the reduction of the arguments to their corresponding normal forms. Such reduction might itself involve the application of other embedded geometrical computations. It is important to notice that from a declarative point of view, the substitution of a geometric description by its value corresponds to a substitution of equals by equals. However, from a computational perspective this substitution is only made possible through a sub-symbolic level of information processing. Only the input and output of such a process can be thought of as symbols; what happens within the process is a continuous, indivisible flow of information. The symbols defined within the algorithm (i.e constants and variables) function at a lower level of abstraction. All expressions of **G** can be interpreted in the same combined symbolic and numerical fashion, but the semantic theory itself is compositional. Although the domains of the functions computed by the primitive geometrical algorithms are sets of infinite cardinality, the actual objects denoted by the expressions of the representational language at a given state of the knowledge-base –the model– is a finite set.

We conclude this section by highlighting that the definition of the language's interpreter as a computer program as shown in the previous section and the semantic rules presented here are two alternative views of conceptualising the interpretation process. Neither view is more complete or more formal than the other, but each view highlights better a particular aspect of the process.

6 Constraint Satisfaction and Layout Design

In this section we investigate how the theoretical apparatus that has been developed can be applied to a particular problem commonly found in computer graphics and CAD applications. The problem is how to satisfy an arbitrary set of geometrical and topological constraints that have been expressed by the human-user in the course of an interactive graphics session. Problems of this kind range from the simple modification of the properties of a graphical object on the screen to the synthesis of complete layout drawings, in which symbols can be added, changed or even deleted in the drawing construction process. Constraint satisfaction problems in this context have traditionally been addressed by interpreting constraints as equation systems, which in turn are solved through numerical techniques, like local propagation, relaxation and Gaussian elimination. This tradition goes back to Sketchpad [21] and is still found in many modern systems, for instance, [1, 3, 4, 9, 20, 23]. However, such approaches have the limitation that configurations produced by graphics systems are causally determined by a number of factors that are contingent to the drafting task: how the constraints are translated into algebraic equations, what initial conditions are selected for the numerical equation-solving method, etc. In short, the solution will come out of the blue for the human-designer and there is no easy way to justify a design decision made by the system. These problems are particularly acute if we wish the constraints to be expressed interactively through linguistic expressions supported by pointing actions, and rely in a pre-defined translation process for converting external expressions into equations systems. In this section we investigate how constraint satisfaction and layout design problems can be represented and solved through symbolic inference techniques.

In order to start this discussion, we emphasise once more that graphical symbols in an application domain are named by *terms* of the representational language. Terms are either basic constants or functional descriptions which refer to graphical objects of a certain graphical sort. Constraints, on the other hand, can be thought of as properties that **must** or **should** hold of a drawing, and they are usually expressed through boolean expressions. That is to say, through expressions that refer to truth values. It is important to emphasise that the notion of constraint presupposes an intention. Through the representational language we can name or refer to accidental features of a drawing without intending to state a constraint; but when we constrain a drawing we express how a drawing should be. A constraint refers to a goal to be achieved either by the user or by the system. We express, for instance, that two lines ought to be parallel, or that a line ought to have a certain length. Constraints do not necessarily have a definite value, as when we assert that the area of a polygon should be larger than a certain minimum and smaller than a specified maximum value. Constraints can also be negative propositions, as when we express that two polygons should not intersect each other. In summary, constraints are expressed as boolean expressions, and we can reason about them through the logical machinery associated to the representational language. Next, we show some situations in which reasoning about constraints is useful.

The application of logic to constraint satisfaction that first come to mind is to verify whether a set of constraints is consistent. This is particularly important in interactive applications in which the human-user is free to add, modify or delete the constraints involved in a drafting task during the course of interactive sessions. He or she might not be fully aware that some constraints are logically incompatible. Consider, for instance, that the user accidentally defines the following constraint set, [7]

(20) $\{\texttt{parallel(l}_1\texttt{,l}_2\texttt{),perpendicular(l}_1\texttt{,l}_2\texttt{)}\}$

Consider that this set is not logically inconsistent, but there is no graphical situation which satisfies these constraints under the normal geometrical interpretation of the predicates involved. Consider also that a given object can be referred to through different terms of the representational language, and that a contradictory situation might not be explicit. For instance, suppose that we substitute $\texttt{l}_1$ by its basic description in (20) as follows,

(21) $\{\texttt{parallel(line(d}_1\texttt{,d}_2\texttt{),l}_2\texttt{),perpendicular(l}_1\texttt{,l}_2\texttt{)}\}$

In order to realise the inconsistent nature of (20) we need not only to know the meaning of the predicates *parallel* and *perpendicular*, but also the fact that $\texttt{l}_1$ and $\texttt{line(d}_1\texttt{,d}_2\texttt{)}$ are co-referential, which is a contingent situation of the knowledge-base. However, the contradiction would not scape to a logical proof procedure. Suppose that we add the following geometrical axiom

(22) $\{\forall x_{line} \forall y_{line}\ \texttt{parallel(x,y)} \rightarrow \texttt{not perpendicular(x,y)}\}$

The union of (20) and (22) is an inconsistent set, and that can be expressed in our representational language. In general, a number of geometric axioms representing necessary

[7] Suppose that the parallel constraint is stated firts, and the perpendicular contraint is stated latter after several interactive transactions have taken place, remembering the identity of symbols is nos straightforward.

constraints that every drawing must satisfy can be permanently enforced in the implementation.

Consider now that *parallel* and *perpendicular* are basic functor operators in our representational language, and they have an associated geometrical algorithm, which is their denotation as explained in the previous section. Then, an alternative way to realise that a number of constraints are inconsistent consists in evaluating the conjunction of the constraints in terms of the semantic rules: inconsistencies can be realised through a process of geometrical verification. Informally we could say that in the former procedure we realise the inconsistency by "thinking" while in the present case the inconsistency is realised by "seeing". The decision as to which strategy is better to use would depend on the kind of application, and the expected computational load of one or the other procedure. Geometrical verification is very efficient, and it is the strategy employed in our current implementation in the Graflog system. However, it is important to keep in mind that inconsistencies realised through this algorithmic method are a matter of contingency, and depend on the state of the data-base. Logical proofs, on the other hand, follow from the structure of the representational language.

Logical inference is also useful in the satisfaction of constraints in which the task consists in mapping a given graphical state to a state in which the constraint set is satisfied. Suppose, for instance, that we have a method for making two lines parallel. Suppose too that such a method is only applicable if a number of conditions hold in the application state. Such a condition migh be, for instance, that the lines involved must not be perpendicular. Once again, we can either prove or verify that the applicability conditions for a method hold. We will make use of this facility when we come to the actual definition of constraint satisfaction methods below in this paper.

Finally, our graphical and logical language supports the definition and interpretation of drafting rules that are useful in the solution of constraint satisfaction based drafting tasks. In the next section, we illustrate the definition of such a kind of drafting rules, and explain their interpretation process.

7 Drafting Intentions and Drafting Rules

The representational language allows us to describe graphical objects and relations in the different states of an interactive session, but transitions from state to state are produced either by the human-user or by the system – by the application of a number of drafting rules. In this section we discuss the definition and interpretation of such a kind of rules. We motivate this discussion with the help of a simple example in which a drawing is modified by a draftsman in a normal drawing activity. Consider Figure 7.1 in which the layout of a house has been sketched. Suppose that at some point during the design process the draftsman realises that the size of the house is too small, and wishes to make it larger. Here, we highlight the fact that design and drafting intentions are commonly expressed through underspecified and vague statements. We could make this task more precise by stating the actual area of the house, but use of the vague statement will emphasise, hopefully, the substance of our argument. Notice that there is an infinite number of ways of making a change producing the desired result, but suppose that we decide to make the house wider in relation to the north-south orientation by dragging the bottom line down. To gain an

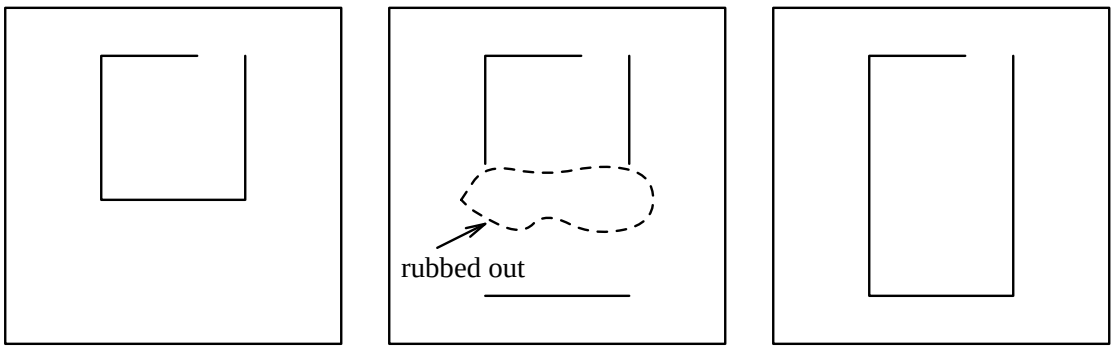


Figure 7: Making larger the area of a house

intuitive grasp of the task we wish to carry out, you could try to do the exercise yourself paying attention to the way your hand proceeds in the modification of the sketch. Working with traditional drafting tools, like pencil and rubber, we could modify our sketch by rubbing out the offending line, and then drawing it below its original position, as shown in Figure 7.2. Suppose now that there is an external observer who knows that we intend to make the house larger –perhaps because we have told him so– and who has observed the modification of the sketch from Figure 7.1 to 7.2. Would he be able to predict that we intend to produce the sketch in Figure 7.3? Presumably, he would. Unless, of course, he knows some additional facts that prevent him from drawing such a conclusion. We say that the inference through which he is able to make this prediction is the interpretation of our drafting intention, or simply *intention interpretation*. Furthermore, if the observer infers the intention correctly, he or she could presumably complete the drafting change him or herself, and we call this latter action *intention satisfaction*. We emphasise that the expression and interpretation of a drafting intention are different notions. Here, the intention has been expressed partly through language –"the area of the house should be larger"– and partly through graphics –drawing the line down. The interpretation of the intention, on the other hand, consists in inferring the course of action that has to be taken in order to achieve the goal implied by the expression of the intention.

To clarify further our notion of drafting intention we emphasise the distinction between individual entities, the properties of these individuals, and the values of these properties. In our example there are five individuals: a house and four walls. The house is represented by a polygon, and the walls are represented by lines. Note that although the lines do not form a closed path, we think of the area of the house in terms of the virtual polygon that is enclosed by the projection of the lines representing the walls. Of course, we assume that the sketch is a faithful representation of the house, and that there are no strange walls, not depicted, which close the explicit gap in an irregular fashion, for instance, by a curve. Although this is not a valid prediction –in a logical sense– familiarity with architectural or cultural conventions would, presumably, facilitate this default assumption. Consider as well that the polygon is a structured object that is defined in terms of its constituent lines, which in turn are defined as a function of their extreme-dots. So it can be seen as a unit, or as an aggregation of parts, depending on the level of granularity at which the drawing is interpreted.

An important consideration in our concept of drafting intention is the destructive character of changes. A drafting process is non-monotonic in relation to the set of constraints holding on the drawing being changed. When a rule is applied with the intention to satisfy a constraint, another constraint might be undone as a side-effect. The modification from Figure 7.1 to 7.2, for instance, produces an unstable structure in which some constraints of the original configuration are violated. In particular, the implicit *e_joint* relations between lateral and bottom walls are broken down by this change, and these constraints must be satisfied again by the next transformation. In general, when we reach a state in which all constraints associated with a task are satisfied, the problem at hand is solved; however, the satisfaction of constraints does not proceed in an incremental –monotonic– fashion. For that reason, the blind application of drafting rules can lead to cycles in the problem solving path, and provision for preventing this behaviour has to be made. Using AI planning terminology, we could say that the sub-goals involved in solving a drafting problem are not necessarily *serialisible* [8].

Another important consideration is that the expression and interpretation of a drafting intention is centered around one or more specific entities. Consider again the change from Figure 7.1 to 7.2. Although our main intention was to make the area of the house larger, what we actually did was to modify the properties of the lines representing the walls. Furthermore, we expressed the intention by focusing on a particular line whose position was explicitly modified. Although the intention was global to the house, it was expressed locally in relation to the bottom wall. In our terminology, the local object around which the expression or interpretation of an intention is centered is called *the focus*: the rule that permits the transformation from Figure 7.1 to 7.2 has the bottom wall as its focus. It is important to highlight that there is no need to think that modifications always proceed by having in focus the more basic elements of a structured object. A human draftsman could change the area of the sketch in Figure 7 in an atomic process, that is to say, by applying one single rule, or in two sequential stages as in the example. This would depend not only on the level of structure of the rules that he or she is able to represent, but also on the limitations imposed by his or her drafting devices. In general, an action that can be performed by a single highly structured rule would have to be accomplished by the application of a number of rules that act on less structured entities. The level of structure of the individual upon which the reasoning process is focused is a paramount parameter in the efficiency of a reasoning process, and it is consistent and desirable to support rules that act at different degrees of granularity in an integrated fashion in a drafting system.

A drafting change can be performed in different ways depending on how drafting rules are defined, and how the focus for the application of each rule is selected. We can think, for instance, of the second transformation in Figure 7 in terms of the sequential application of two rules that have as a focus an individual of sort line. Alternativately, we can think of the same change as produced by the parallel –simultaneous– application of the same rule, but with different focus lines. Furthermore, we can think that the transition is produced by the application of a single rule that has two focus individuals of sort line. All options are consistent, although they have different computational costs. The third view illustrates that the focus needs not be a unique individual, nor a number of individuals topologically connected. Furthermore, a rule can be defined in relation to a set of individuals that have different degrees of structure. Our notion of focus is functional: a rule is focused on a set of individuals that are relative parameters of the intention that the rule aims to satisfy, and

in order to apply a drafting rule the focus must be defined. It is important to highlight that when a problem is solved by applying several rules that act upon basic individuals –like dots– or upon individuals of an intermediate degree of structure, the focus changes during the inference process. Furthermore, which focus individuals are selected in the course of solving a problem determines, to a very large extent, how the user-intention is "understood" by the system.

Our intuition is that the selection of the focus precedes and determines, to a great extent, what is the form of the drawing to be produced. Consider again the interpretation of the drafting intention in Figure 7. Suppose that the original intention is expressed as,

(23) `Make the area of the house larger`

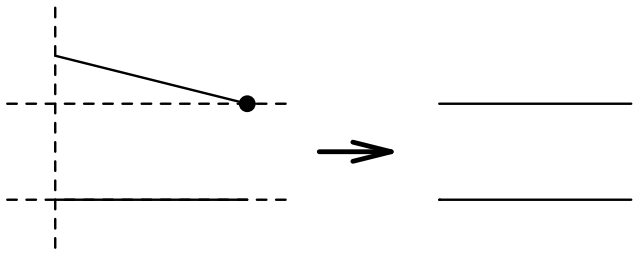
at the time the polygon representing the house is pointed out. One way to interpreting such a drafting intention would be to inferring and executing a plan. One such plan could be as follows,

(24) `Move line x down. Move dots y and z down`

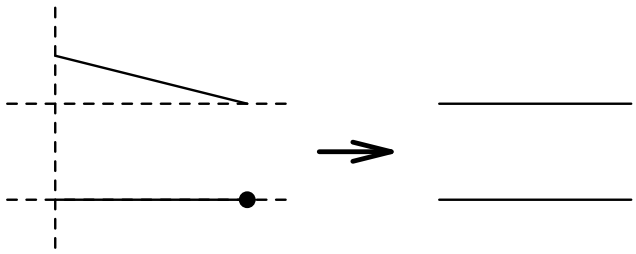
The interpretation of the drafting intention consists in inferring the graphical statements in (24). Satisfying the intention consists in "uttering" such a piece of "discourse" –of course through graphics and in a way such that the constraints are satisfied, as will be shown below. Notice that the production of a sequence of statments in (24) can be partioned into two different –but related– conceptual problems: the first is to identify what is the actual command sequence that solves the problem, and the second is to indentify what are the individuals referred to by the variables in the command sentences. We consider that the identification of the referent of these variables depends on contextual and pragmatic considerations, and an intuitive idea is to search for the focus of a change command in the previous graphical sentences. Our suggestion is that in our example **x** must be related somehow to the polygon representing the house, and that **y** and **z** must be related to **x**.

Notice that the level of structure of graphical focus can vary. In our example, the type of *the house* is `polygon`, but the type of **x** is `line`, and the type of **y** and **z** is `dot`. Then, the polygon cannot be the referent –or value– of **x**, which in turn cannot be the value of neither **y** or **z**. However, the polygon is related to the line –it is the whole of which the line is a part– and the line is similarly related to the dots in the state in Figure 7.1. Generalising this intuition, we could resolve the graphical variables to an individual that has been previously referred to either directly, or alternatively in an indirect fashion by referring to one of its subordinated –or super-ordinated– constituents. If a line is defined in terms of its constituent extreme dots, for instance, and it is the focus of a graphical command, so are its constituent dots.

After this motivation we introduce some instances of our drafting rules and illustrate how the solution of a constraint satisfaction drafting task is found. In Figure 8, two rules for making two lines parallel are illustrated. The left-side of the rules specifies the graphical context in which the rule can be applied, and the right-side illustrates the graphical state after the application of the rule. The solid lines in Figure 8 denote actual graphical symbols of the drawing that is being produced, while the construction lines are defined and "drawn" by the rules themselves. The dark dot-labels indicate the focus of the rule, and all references and operations of a rule are defined relative to such a focus. So, in the satisfaction of a parallel contraint between two lines, each of the four extreme-dots of the two lines involved



parallel 1



parallel 2

Figure 8: Drafting Rules

can be selected as the focus, and the construction lines would be drawn relative to such a reference. Subsequently, the transformation made by the rule is relative to the focus. Another way to think of the rules is in terms of a function from dots to dots in which the domain is the focus, and the range is the dot –or set dots– whose position is changed in the satisfaction of the rule's constraint. Note that the two rules produce a similar graphical configuration as their output. However, they are "pivoted" on a different reference, and they execute a different procedure: while the focus is placed on the line that is fixed in the satisfaction of the constraint in the first rule, the focus is an extreme-dot of the line that is rotated in the second.

Next, we illustrate the application of drafting rules as currently implemented in the Graflog system. Consider Figure 9 in which the human-user has entered the statement *These are walls* at the time a number of symbols of graphical type *line* are selected and placed in the drafting space. Next, the user types the statement *This is a house* as the vertices of a polygon that stands for the house are pointed out on the screen. In Figure 10 we identify the graphical entities that have been introduced through these interactive transactions. These references will be used below when we explain in detail the interpretation of drafting intentions. This graphical and linguistic input is interpreted by a heuristic algorithm ("the graphical parser") that produces the expressions in (25) which are asserted in the KB_G as follows,

(25) $\mathtt{l}_1$ = line($\mathtt{d}_1$,$\mathtt{d}_2$)

$\mathtt{l}_2$ = line(end($\mathtt{l}_1$),$\mathtt{d}_3$)

$\mathtt{l}_3$ = line(end($\mathtt{l}_2$),$\mathtt{d}_4$)

$\mathtt{l}_4$ = line(end($\mathtt{l}_3$),$\mathtt{d}_5$)

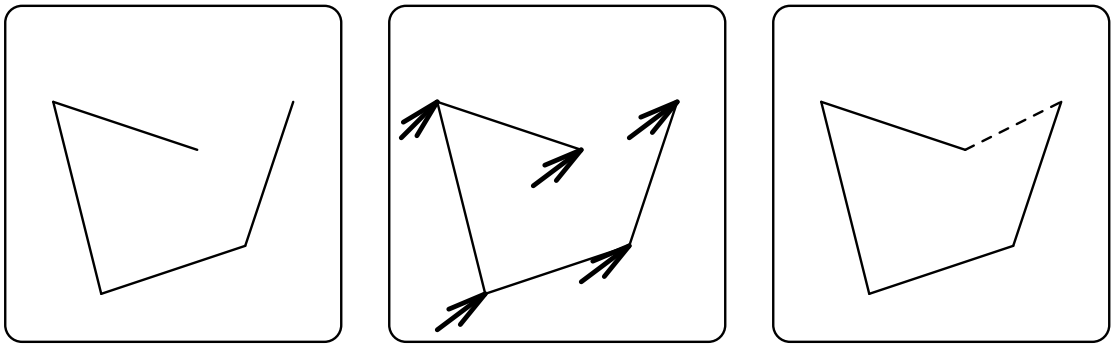


Figure 9: Definition of graphical objects

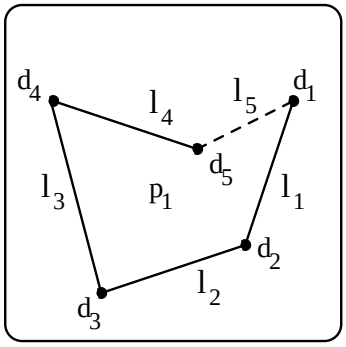


Figure 10: Definition of graphical objects

```
l5 = line(end(l4),origin(l1)
k1 = path(l1,l2)
k2 = path(k1,l3)
k3 = path(k2,l4)
k4 = path(k3,l5)
poly1 = polygon(k4)
```

Now, we come to the definition of constraints. In Figure 11 the user types the statement *These are parallel* as two lines are pointed out on the screen. This statement expresses that the the user wishes that such a pair of lines should stand in a parallel relation. The constraint is represented in the knowledge-base as follows,

(26) `parallel(l2,l4)`

Next, in Figure 12 the user selects a dot and moves it into a lower-right position. We take this information as the specification of a drafting intention expressed by the user. The result of interpreting this intention according to our drafting rules is also shown in

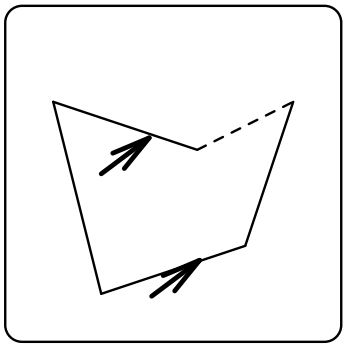


Figure 11: Definition of a geometrical constraint

Figure 12. Note that the dot that is modified by the user in the change command is the focus for the application of the drafting rule. The application of the rule corresponds to the formulation and execution of the following plan,

(27) `parallel(l1,l3),focus(x)`
`move(f(x))`

where the variable values are `x = d2` and `f(x) = d4`. The value of **x** is directly determined in terms of the interactive input, and **f** is a function determined by the drafting rule.

Whenever more than one constraint is involved and the problem must be solved by the application of more than one drafting rule, a procedure for shifting the focus during the problem-solving process must be defined. Consider the situation illustrated in Figure 13 in which we have two parallel constraints directly stated through the interaction and one additional constraint that is implied by the transitivity of the parallel relation. The constraints of this problem are,

(28) `parallel(l1,l3)`
`parallel(l1,l5)`

The implied constraint is

`parallel(l3,l5)`

In Figure 13 a change similar to our previous example is indicated. The question is how the problem should be solved. If the problem can be partitioned into a number of sub-problems that can be solved by our drafting rules, the solution would depend on the choice of foci in the different solution steps. The constraints that we impose for the focus selection problem in our current implementation are:

1. Focus shifting rule: if the solution of a drafting problem is produced by a sequence of drafting rules $R_1, ..., R_n$, the dot modified by rule R_i is an admissible focus of the rule R_{i+1}. The transformation R_1 is the interactive input made by the human-user, and the focus of R_2 is the dot explicitly modified in the interaction.

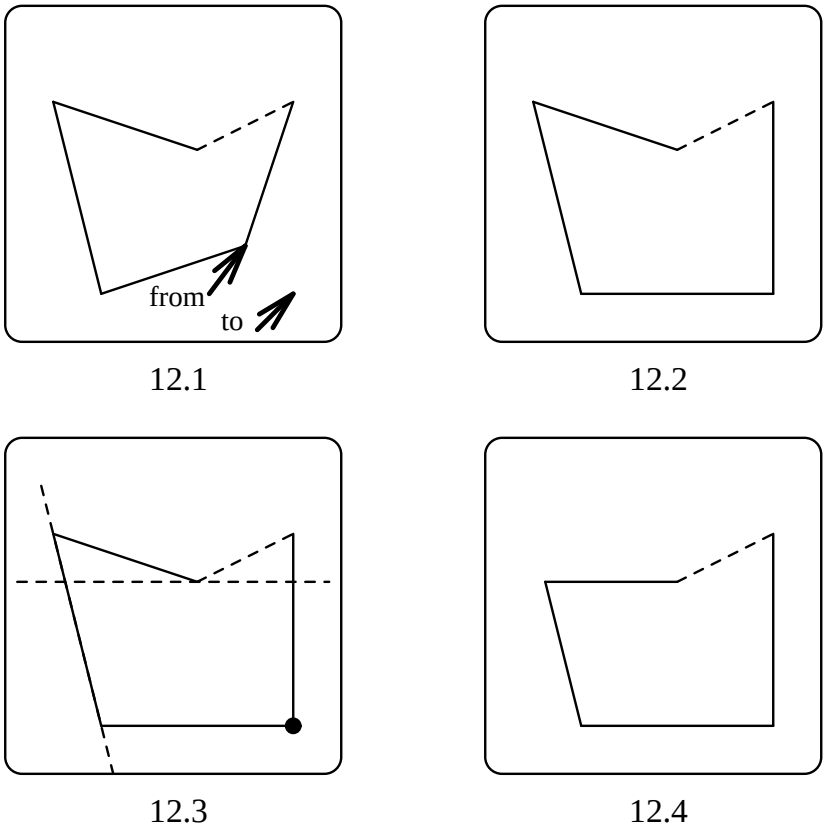


Figure 12: Application of Parallel Rule

2. Focus embedding rule: If $\mathtt{d}_i$ is an admssible focus for rule R_i, then it is also an admissible focus of rule R_j for all $j > i$. The intuition behind this rule is that the main focus of a change –the one that is indicated externally– "controls" subordinated changes.

3. Focus neighbouring rule: If a dot $\mathtt{d}_i$ is a candidate focus, so all $\mathtt{d}_j$ whose distance from $\mathtt{d}_i$ is one line segment are admissible foci too (i.e. $\mathtt{d}_i$ and $\mathtt{d}_j$ are the extremes of a line or an arc). This extension provides additional "local" references for a drafting task because under the focus shifting and focus embedding rules the system might fail to solve a problem for lack of a plausible focus even if the task can be solved by the current set of drafting rules. Consider that in realistic drafting tasks, foci can be graphical objects of any degree of structure, but in the current version of Graflog only dots can be foci of transformations, limiting the power of the problem-solving mechanism. We hope to overcome this limitation in a further implementation of the system.

4. Cyclicity constraint: No dot can be modified twice in the the same transformation sequence. This constraint prevents the problem-solving process from going into infinite drafting loops.

We come now to the solution of our drafting task. Consider that the proliferation of

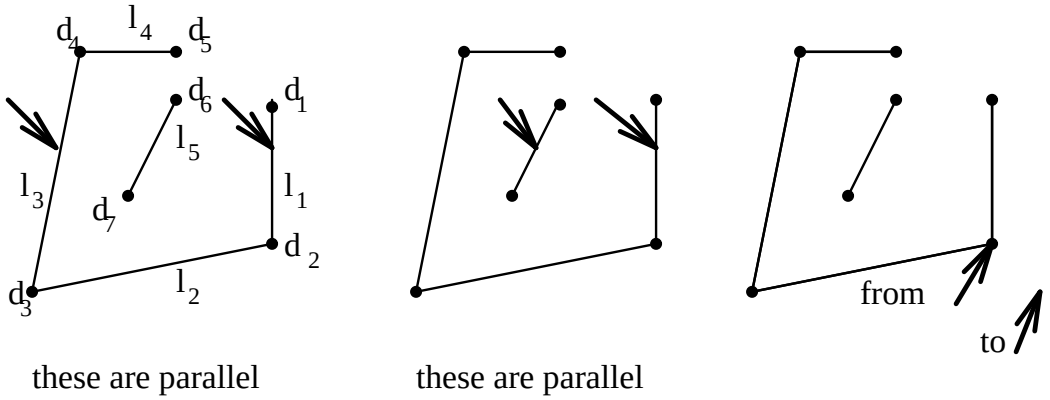


Figure 13: Application of Parallel Rule

admissible foci in combination with the set of drafting rules define a search space in which a given problem can have several solutions and a given solution can be found in terms of two or more different search paths, if it has a solution at all. This conforms to our intuition that drafting problems are highly non-deterministic.

The drafting inference is a forward search process. In each state of the problem-solving task we compute the product set of the power set of atomic constraints that are not satisfied by the drawing in the state with the set of plausible foci in the state. This set is called the *applicability matrix*. Every member of the applicability matrix is an ordered-pair of a set of constraints and a focus dot. Consider now that each drafting rule is defined to satisfy a set of constraints in relation to a focus individual. We define the cross-product of the rules-set and the applicability matrix as the *weights-matrix*. Let i be an entry in the applicability matrix and j a rule. If j cannot satisfy i then `weights-matrix(i,j) = 0`; otherwise, the rule is applicable and the value of `weights-matrix(i,j)` is the number of constraints in the goal of rule j multiplied by an arbitrary preference factor assigned to every drafting rule. That is, the higher the degree of structure and the rule's preference factor, the larger its weight in the weights-matrix in that state. The entries in the weights-matrix are ordered according to the weight's value, and this list is explored in a depth-first search strategy. However, the number of solutions is independent of the search strategy, and the full set could be computed by exploring the full search space. All solutions could be found more efficiently in a parallel implementation of the problem-solving mechanism, which is already being considered.

Next, we show three alternative solutions for the problem in Figure 13. In the first case the intention is interpreted by the graphical discourse in (29). The corresponding graphical explanation as produced by Graflog is shown in Figure 15 at the end of the paper.

(29) `parallel(`l_1`,`l_3`),focus(`d_2`)`

`move(`d_4`)`

`parallel(`l_1`,`l_5`),focus(`d_1`)`

`move(`d_7`)`

The second inference is illustrated in Figure 16, and is as follows,

(30) parallel(l_1,l_3),focus(d_2)

move(d_4)

parallel(l_1,l_5),focus(d_2)

move(d_6)

Note that the only difference between (29) and (30) is that in the former, the focus of the second transformation is d_1, while d_2 in the latter. The final configurations in Figure 15 and 16 are alike, except that the middle detached line is displaced rightwards in the former, and leftwards in the latter.

The third inference in (31) illustrates the non-serialisible nature of the task in relation to the constraint set, and also the use of the second drafting rule in which the focus is placed on the line that is rotated. The graphical sequence in Figure 17 is the graphical translation of the following set of sentences,

(31) parallel(l_1,l_3),focus(d_2)

move(d_4)

parallel(l_1,l_5),focus(d_2)

move(d_1)

parallel(l_1,l_3),focus(d_1)

move(d_3)

Notice that the constraint that is satisfied by the first modification is undone by the second. However, the last transformation re-enforces the satisfaction of the first constraint. Notice also that in the first and third transformations, the focus lies on the line that is fixed in the application of the rule; however, in the second transformation, the alternative parallel rule in which the focus lies on the line that is rotated is considered.

As can be seen, the motivation for this method is based on the view that drafting task are rule-based. The notion of focus reflects what is the set of geometrical objects that the human draftsman is attending to when he or she is engaged in a drafting task, and how his attention varies during the solution of a problem. In summary, we view constraint satisfaction drafting tasks as symbolic inferences supported by a highly expressive representational language, in which graphical symbols, relations, constraints and drafting rules can be expressed.

8 On the Architecture of an Intelligent Drafting System

We are now in a position to discuss some elements of an architecture for a CAD system supporting logical representations. This architecture reflects closely the structure of the Graflog system, and is centered around the definition and use of expressions denoting graphical objects and constraints. Then, the backbone of the system is constituted by the

geometrical knowledge-base KB_G, the constraints set, and the drafting-rules knowledge-base. Interactive transactions have the purpose of defining instance expressions of these representational structures, presenting queries about the objects and relations denoted by them, and specifying changes on these structures.

Figure 14 illustrates the main components of the functional architecture. As can be seen, there are four functional levels: the human-user, the interface, and interpretation and representation levels. The representational structures are the backbone of the scheme. The arrows abstract the main control and information paths. The "internal" aspects of this

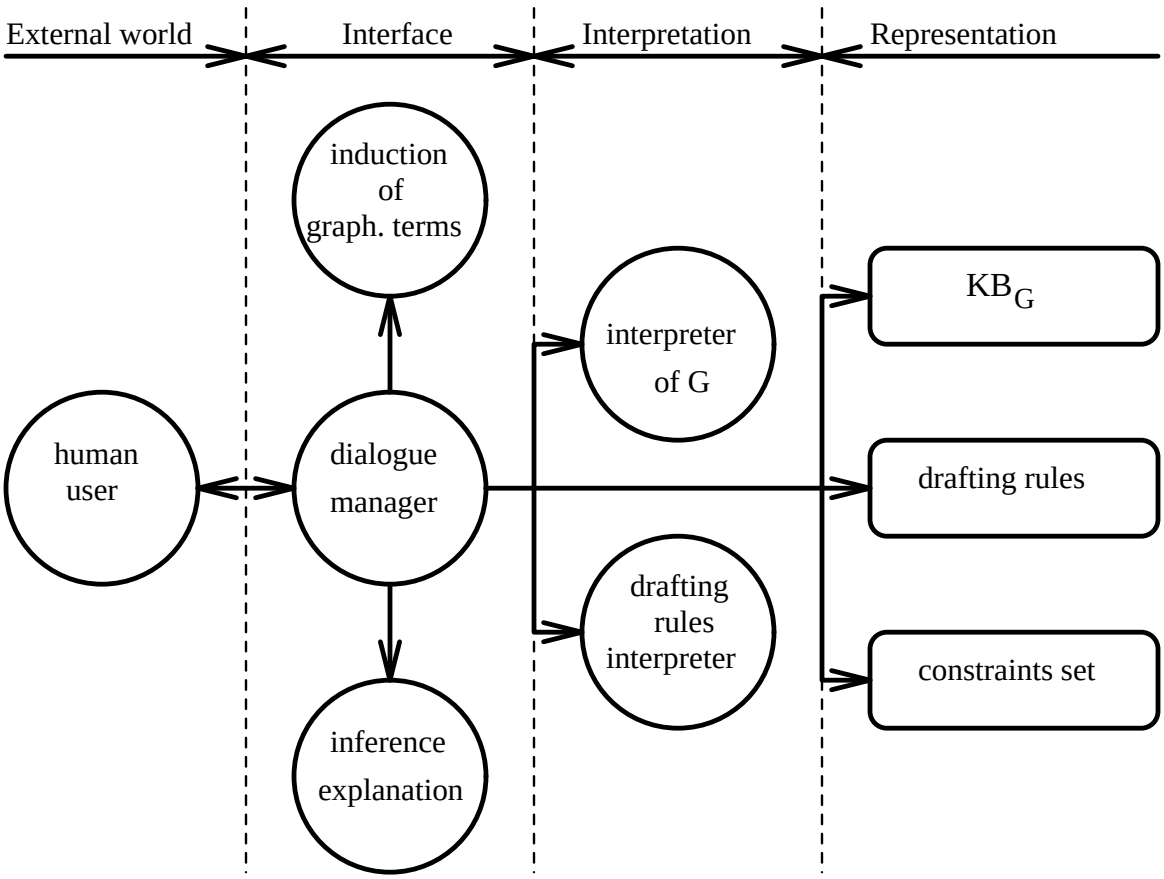


Figure 14: Architecture of an Intelligent CAD system

architecture have been discussed in detail in the previous sections. Here, we concentrate on its front-end functionality. The interface component is constituted by three main functional modules: the dialogue manager, the induction of representations and the explanation of the inferences performed by the system.

The dialogue manager is a complex process that takes the external information, and handles all related processes. This process has access to the internal representational structures and interpreters of the system. In addition, it contains a knowledge-base of its own for handling input strings. The application of the rules in the interface knowledge-base is constrained by the context, and the dialogue manager state. Here, we do not address in

detail the definition and interpretation of the interface rules. However, it is worth pointing out that the theoretical framework used above for the definition and interpretation of graphics can also be applied here. Next, we give a general idea of the objects and processes that the dialogue manager should be able to handle.

The first issue to address is how the expressions in KB_G and the constraints set are defined. The most basic approach is to let the user type directly the definition of all representational objects. To support this strategy, a number of interactive concepts and tools can be developed for allowing the user to define and verify the graphical properties of the drawing as well as the consistency of the representation.

A more developed system can have a number of built-in strategies for inducing representations from the graphical input. Graflog, for instance, uses a heuristic algorithm called *refers_to_dot* that given a position on the screen (provided through a pointing device), produces an expression that refers to a dot in such a position. This expression depends in turn on the current state of the drawing, and captures a number of geometrical and topological properties that permit reference to the dot in an informative and relevant way. The current algorithm produces an expression that "links" all lines that intersect at a selected position. We refer to this strategy as the "strong attachment condition". We have to keep in mind, however, that definitions constituting the representation of a drawing depend on the drafting and design intentions of human-users. In general, these intentions cannot be predicted, and algorithms for inducing graphical representations should be considered as default strategies, and interactive tools for reviewing the representation must be available. For instance, an interface facility for pointing to things to be attached or detached as required should be available. A "detach" command, in particular, would simply replace a complex definition for its basic description.

The partial interpretation of function terms raises another set of HCI issues. Most traditional drafting systems implement extensional representations in which parameters are limited to be basic constants. Such systems do not allow interactive states in which graphical objects lack a well defined referent. In systems based on logical representations a better compromise between user intentions and representations can be found. In particular, the explicit definition of error conditions of different sorts allows us to define error messages according to the kinds of expressions having a partial interpretation or inconsistent constraint sets.

The interactive definition of the constraints set opens another set of interesting HCI issues. Interactive tools for exploiting reasoning with combined logical deduction and geometrical verification can help the user to clarify and synthesise his or her own design intentions. In complex drawings he or she can, for instance, select a subset of constraints in the constraint set as the immediate goal for the system. Although it is perhaps possible for a system to compute a design solution in a one-off fashion, the partition of a global task into a number of local tasks can perhaps be a better interactive strategy. Furthermore, a number of different, but not necessarily mutually exclusive, constraint sets can be defined. Interactive facilities should allow the user to create, modify or select a given contraint set, according to the state of the drafting construction process.

The definition of drafting-rules for solving problems can also be subject to interface research. The system can be provided with a number of rules that are as context free as possible. Such rules might be useful for solving simple drafting problems. However, if there is a substantial amount of knowledge about the application domain, the system could be

customised with a small number of powerful rules able to produce complex modifications in composite graphical objects in a modular fashion. Interface facilities can be developed for the interactive definition and simulation test of such a kind of rules.

In an intelligent CAD system, a facility for explaining the inferences made by the system should also be available. Even with very small knowledge-bases the consequences of the assertions made by the user might be difficult to trace. Many HCI issues arise in relation to the kind of dialogue that can support such a task.

Finally, the explanation of drafting inferences that solve constraint satisfaction problems is an indispensable interface facility. This is so because in this sort of inference it is not only relevant to know the geometrical properties of a graphical configuration satisfying the design constraints, but also how such a configuration is related to the original drawing and the design intention. If the system lacks such an explanation facility, the human-user might not be able to identify whether a solution to a constraint satisfaction problem corresponds to his or her design intentions. Furthermore, if the design concept is consistent but the system finds no configuration satisfying a drafting intention, then the user should help to complement the system's knowledge.

Before concluding this section, it is worth pointing out that we do not intend to specify particular HCI strategies for a practical implementation. However, we emphasise that the properties of logical representations permit us to formulate precise questions about why an interactive concept is required, and also impose strong constraints on how it can be implemented. We hope to address the definition of specific interactive concepts and tools in a further work.

9 Conclusions and Further Research

We conclude this paper with a brief reflection on the nature and applications of logical representations in drafting and CAD systems. First of all, we highlight the fact that this discussion has been motivated by a semantic concern. The syntactic definition and semantic interpretation of the representational language allows us to distinguish the representational scheme from the represented object in a rather objective fashion. In addition, the semantic theory helps us to clarify a number of issues that arise in computer interaction, and also to extend the power of current drafting and CAD systems. Particularly, the notions of partial interpretation and error-conditions, and constraints inconsistency are recurring problems underlying interactive systems in which the input is not only passive data but has a functional significance too. In many traditional systems semantic problems are not realised either by human-users or by programmers both because of the lack of a well-founded theoretical framework, and also due to the unflexible nature of such programs.

Next, we state a number of concluding remarks about the main notions discussed in this paper, as well as some potential issues for further research:

- Representation: Graphical objects can be defined in terms of other symbols in a fully compositional fashion. Compositions are quite flexible, and the geometry is used in terms of the form of expressions, rather than in a predefined and fixed set of strategies. There are as many graphical compositions as well-formed expressions of the graphical language.

- Reasoning: The ability to reason about drawings and drafting problems expressed

by human-users during interactive sessions is greatly enhanced by the use of logical representations and its associated semantic theory. Logical inferences allow us to review the consequences of a given set of sentences. Pragmatic inferences are helpful to synthesise drawings and design objects.

- Traditional techniques for approaching constraint satisfaction focus on the properties of methods for solving equations, rather than on the clarification of the notion of "constraint". The use of logical representations helps us not only to clarify the notion of constraint, but also to simplify and explain solutions to so-called constraint satisfaction problems.

- The semantic theory helps us to clarify the relation between the symbolic and numerical levels of computation. In particular, simple strategies for handling numerical errors can be used. Graflog, for instance, is fairly robust using single precision arithmetic.

- The explicit linguistic view of representation can help to define precise HCI research issues. Although interface design must consider social and psychological aspects of human-users that are external to the computer program, the internal aspects of interactive processes can be subject to objective research. From the system's point of view, only what can be expressed in the representational language can have a functional role in the interface. HCI concepts should be constrained by the kind of expressions to which they map in internal representations. Important HCI issues are consistency and interactive problem-solving, supported by logical and geometrical inference processes.

- The close similarity between the definition of representational languages and the methods for formal specification of programming languages encourages a sound programming methodology. In particular, the notion of abstract data-type in object-oriented systems, corresponds closely to the sorts and operation symbols of our representational language.

There are several issues for further research. Of particular interest is the development of drafting applications supporting a non-trivial set of drafting rules, in which the focus is not limited to a single object of a basic sort, allowing the definition of inferences that focus on lines, polygons or even graphical objects of a composite geometrical structure.

Another promising line of research is exploring the potential of parallel architectures for the solution of complex drafting problems. Parallelism can be exploited at several levels: from the use of general parallel logic programming environments, to specific uses of parallel architecture to deal with the huge volume of computation due to the non-determinism of drafting tasks.

Finally, an important issue for further research is the extension of the formalism for the representation of abstract objects and non-graphical concepts that arise in drafting and design tasks. Such concepts are usually expressed through natural language and other symbolic languages and notations. This extension would allow us to represent not only the geometrical shape of a design object, but also aspects of its functionality, the properties of the materials from which the object is made, aspects of the machinery needed for its manufacture, etc. Some progress in this line has already been made [7].

References

[1] Bjorn, N. F. B., Maloney, J., Borning, A., An Incremental Constraint Solver, in: Communication of the ACM 33, No 1. (1990) 54–63.

[2] Bijl, A., Computer Discipline and Design Practice, Edinburgh University Press (1988).

[3] Borning, A., The Programming Language Aspects of Thinglab, A Constraint-Oriented Simulation Laboratory, in: ACM Transactions in Programming Languages and Systems 3, No. 4. (1981) 353–387.

[4] Borning, A., Maher, M., Martindale, A., Wilson, M. Constraint Hierarchies and Logic Programming, Computer Science Department FR-35, University of Washington.

[5] Dowty, A. D. R., Wall, A. R. E., Peters, P. S., Introduction to Montague Semantics (Reidel Publishing Company, 1981, Dordrecht, Holland).

[6] Goguen, J., Thatcher, J. W., Wagner, E. G., An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types, R. T. Yeh (ed.), (Current Trends in Programming Methodology, 1978, Prentice-Hall). 80–149

[7] Klein, E. H., Pineda, L.A., Semantics and Graphical Information, in: Human-Computer Interaction -INTERACT'90 D. Diaper et. al. (Eds.). (Elsevier Science Publishers B. V. North Holland, 1990).

[8] Korf. R. E., Learning to Solve Problems by Searching for Macro-Operators, Research Notes in Artificial Intelligence 5, (Pitman Advanced Publishing Program, Boston, 1985).

[9] Leler, W., Constraint Programming Languages: Their Specification and Generation (Addison-Wesley, Reading, Mass. 1988).

[10] Liu, Y., Popplestone, R. J., Symmetry constraint inference in assembly planning: Automatic assembly configuration specification, in: Proceedings of the National Conference in AI. (Boston, Mass. 1990) 1038–1044.

[11] Pineda, L.A., Klein, E.H., Lee, J., Graflog: Understanding Graphics Through Natural Language, Computer Graphics Forum 7 (1988) 97–103.

[12] Pineda, L.A., A Compositional Semantics for Graphics, in: D. Duce and P. Jancene (eds.), Eurographics'88 Conference Proceedings (Elsevier Science Publishers B.V., North-Holland, 1988).

[13] Pineda, L.A., Chater, N., GRAFLOG: Programming with Interactive Graphics and PROLOG, in: New Trends in Computer Graphics, Conference Proceedings of CG International'88, Geneva (Springer-Verlag, Heideberg, 1988).

[14] Pineda, L.A., GRAFLOG: A Graphical and Logical Programming Language, in: Proceedings of the IFIP TC5 International Conference on CAD/CAM, G. Leon Lastra, J. Encarnaçao and A. G. Requicha (eds.) (Elsevier Science Publishers B. V. North Holland, 1988).

[15] Pineda, L.A., Klein, E.H., A Graphical and Logical Language for a Simple Design Domain, in: P. ten Hagen and P. Veerkamp (eds.) Intelligent CAD Systems III (Springer-Verlag, Berlin, 1991).

[16] Pineda, L.A., GRAFLOG: A Theory of Semantics for Graphics with applications to Human-Computer Interaction and CAD Systems (PhD thesis, University of Edinburgh, 1989).

[17] Pineda, L.A., On Computational Models of Drafting and Design, in: Edinburgh Architecture Research (18), (Department of Architecture, University of Edinburgh, 1991). To be publish in Design Studies.

[18] Pineda, L.A., Reference, Synthesis and Constraint Satisfaction (To be published in Eurographics'92 Conference Proceedings, Cambridge, U. K., 1992).

[19] 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. G. Leon Lastra, J. Encarnacao, G. Requicha (Eds.), (Elsevier Science Publishers B. V. North Holland, 1989).

[20] Steele, G.L., The Definition and Implementation of a Computer Programming Language Based on Constraints, (Technical Report AI-TR-595, MIT, Cambridge, Mass, 1980).

[21] Sutherland, I., Sketchpad: A Man-Machine Graphical Communication System, in: AFIPS SJCC Proceedings (1963) 329–346.

[22] Wang, D., Lee, J., Graphics–Assisted Reasoning, Internal Report, (EdCAAD, Dept. of Architecture, University of Edinburgh, 1991).

[23] Gusgen, H.W., CONSTAT, A System for Constraint Satisfaction, Research Notes in Artificial Intelligence, (Morgan Kaufmann Publishers, Inc. San Mateo, California, 1989).