Reference, Synthesis and Constraint Satisfaction*

Luis A. Pineda

EdCAAD and Centre for Cognitive Science
University of Edinburgh

Abstract

In this paper we discuss two kinds of constraint satisfaction problems that arise in the context of geometric modelling. In particular in the modification of 2-D wire-frame diagrams that are subject to an arbitrary number of geometrical and topological constraints. We argue that problems in this domain can be classified in two categories that we shall call *problems of reference* and *problems of synthesis*. Since Sutherland's Sketchpad program [16], a large number of systems have addressed constraint satisfaction in terms of the representation of constraints sets as equation systems, which in turn are solved by numerical methods like local propagation, relaxation and Gaussian elimination. Here, we present an alternative framework. We argue that conceptualising constraint satisfaction as symbolic rather than "numerical" problems helps to clarify the notion of "constraint", simplify solution methods, and to explain the intuitive inferential processes underlying the modification of drawings in the course of interactive drafting sessions. The theory presented in this paper has been tested with an experimental computer program called Graflog [5, 8, 9, 10, 11, 12]. 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.

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

1 Prescription 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 linguistic 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 Figures 1.1 to 1.6 in which a drawing screen is illustrated. In Figure 1.1 two lines are edited by standard interactive techniques. Suppose that we intend such a pair of lines to

*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 Council Initiative in Cognitive Science/HCI. I am grateful to Ewan Klein and John Lee for their substantial contributions to this research.

stand in a *t_join* relation during the interactive session. A representation of such lines can be

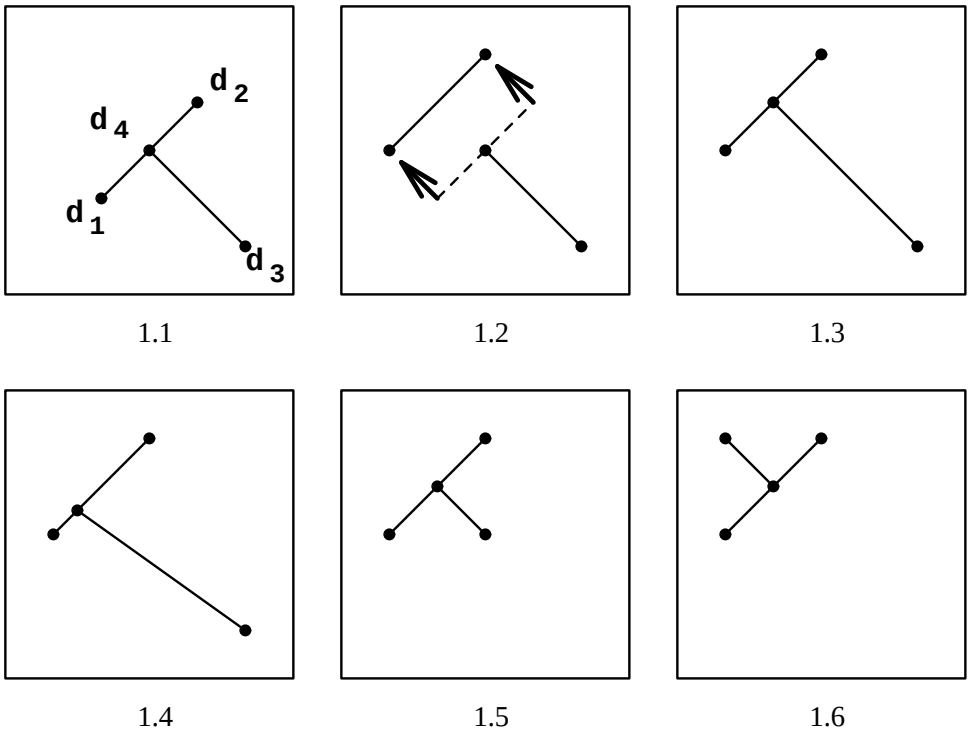

Figure 1: A Simple Constraint Satisfaction Problem

stated by the following symbolic equations:

(1) l_1 = line(d_1,d_2)

(2) l_2 = line(d_3,d_4)

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

(3) t_join(l_2,l_1)

Now consider what happens if one of the lines, l_1, as 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 is to be accomplished. In fact, Figures 1.4 to 1.6 are alternative constructions, less intuitive perhaps, but which nevertheless satisfy the drawing description and constraints.

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 [13]. This has been labeled 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 explict 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 knowledge representation language comes into the picture. A language that is expressive enough will allow us to refer and to describe not only the graphical symbols constituting a drawing but also its associated constraints. The purpose of this paper is to argue that a family of constraint satisfaction problems reduce to problems of naming or reference. However, it will be shown that not all problems belong to this class, and there is a family of problems whose solution will be causally determined by the constraint satisfaction method. We will refer to this latter class as "problems of synthesis".

2 Names and Descriptions

In parametric design we deal very often with problems of reference. These problems have traditionally been solved with the help of constraint satisfaction algorithms. Here, we argue that a proper naming policy would substantially reduce the need for using such problem-solving techniques. In this section we develop the example in Figure 1 to illustrate the consequences of naming graphical symbols and constraints in an integrated fashion.

Consider Figure 2.1 in which two lines have been defined. The lines are represented as follows,

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

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

The left side of each of these equations is a name, and the right side is a description. The equality holds if the name refers to, or names, 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. Note that within the graphical description of a line, names referring to graphical objects of other kinds can be included. The symbols $\mathtt{d}_1$, $\mathtt{d}_2$, $\mathtt{d}_3$ and $\mathtt{d}_4$ are in fact names of the corresponding graphical dots on the screen.

Consider now Figure 2.2 in which we have added two additional lines in the drawing. Suppose, in addition, that we define the new lines, $\mathtt{l}_3$ and $\mathtt{l}_4$, with the intention that a *t_join* is established between each of these lines and both $\mathtt{l}_1$ and $\mathtt{l}_2$. The question we are interested in is how do we refer to these new objects? The most simple way to do it is by specifying the lines and the constraints with independent expressions as we did before. However, if we allow the substitution of names by descriptions when they have the same syntactic type and denote the same individual, a more powerful naming device is at hand. For instance, if the expression intersect($\mathtt{l}_a$,$\mathtt{l}_b$) is of sort *dot*, it can be replaced by the constant $\mathtt{d}_a$ if this constant names in fact the corresponding intersection, and vice versa. With this notion we can define the lines added in Figure 2.2 as follows,

(6) $\mathtt{l}_3$ = line($\mathtt{d}_5$,intersect($\mathtt{l}_1$,$\mathtt{l}_2$))

(7) $\mathtt{l}_4$ = line($\mathtt{d}_6$,intersect($\mathtt{l}_1$,$\mathtt{l}_2$))

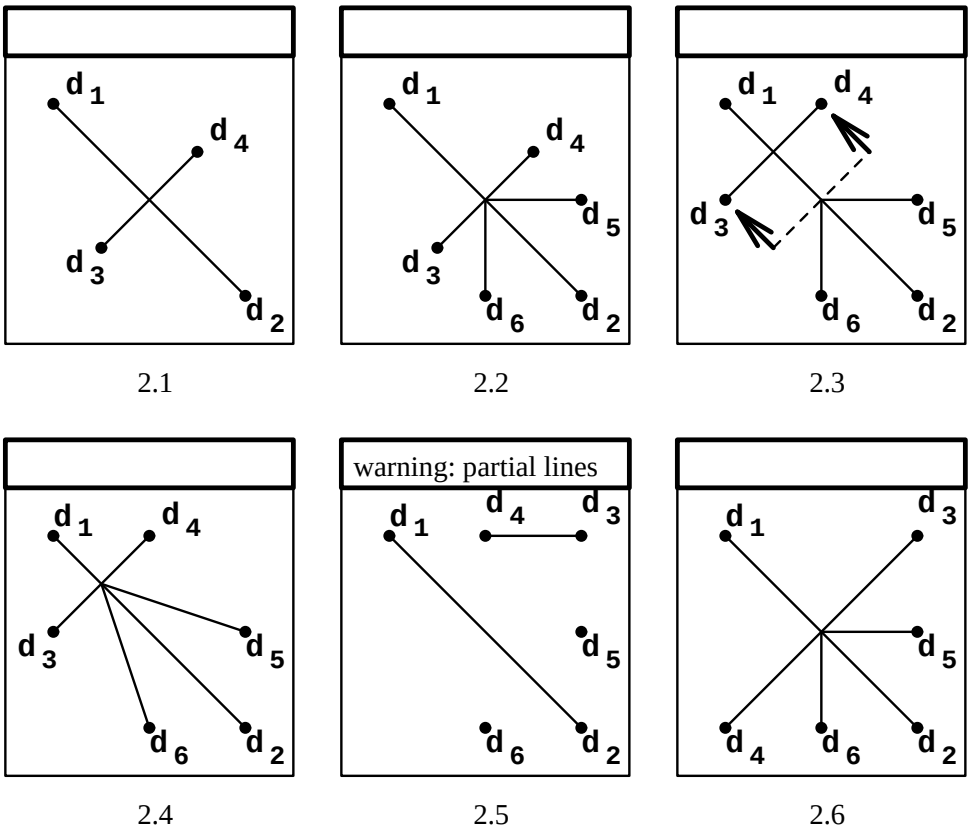

Figure 2: Descriptions and Constraint Satisfaction

The idea in (6)–(7) is to use descriptions that hold in any graphical state during the drafting production process. Note that if we edit Figure 2.2 by dragging $\mathtt{l}_2$ as shown in Figure 2.3, the lines $\mathtt{l}_3$ and $\mathtt{l}_4$ will be updated as shown in Figure 2.4 in virtue of the description only. Here, there is no need to invoke a constraint satisfaction algorithm at all. Furthermore, the description remains constant across graphical change, and the only parameters that need to be updated in the knowledge-base are the values of the most basic constants, namely, the position of the basic dots.

Another way to think of graphical descriptions is as functions from the invariant geometrical objects of a modelling task to drawings. This sheds additional light on problems related to constraint satisfaction in which we are interested in modifying the value of some graphical properties while keeping the values of other properties constant during the modelling process. In the current example, we are controlling the *t_join* relations between the lines $\mathtt{l}_3$ and $\mathtt{l}_4$ to the lines $\mathtt{l}_1$ and $\mathtt{l}_2$ in terms of the basic descriptions of $\mathtt{l}_1$ and $\mathtt{l}_2$.

The use of 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, for instance, the transition from Figure 2.4 to 2.5. Here, the position of $\mathtt{d}_3$, a basic constant, has been changed in such a way that the lines $\mathtt{l}_1$ and $\mathtt{l}_2$ do not intersect each other in the

new state. The question is, what happens with the lines whose descriptions are functionally dependent upon such an intersection? Normally, functions represented by descriptions in an interactive graphics context are partial rather than total, and there are states in which their images are not defined. Note that most drafting systems would not allow the transition from 2.4 to 2.5 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 strategies. 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.

The use of descriptions allows the transition from Figure 2.4 to 2.5. Although the actual value of some graphical properties is undefined in the new state the representation is still consistent. In fact, as was mentioned above, descriptions are invariant. In our current prototype implementation, when the system interpreter finds that there is no value for an expression like `intersect(`$\mathtt{l}_1$`,`$\mathtt{l}_2$`)` in the current state, it allows the change and sends a message warning the user of the irregular condition. Note as well that some graphical objects referred to in the description –the independent dots– have nevertheless a well defined value and can be drawn on the screen.

Consider now the transition from Figure 2.5 to 2.6 in which the other extreme of $\mathtt{l}_2$ is modified in such a way that the intersection between $\mathtt{l}_1$ and $\mathtt{l}_2$ is defined again. Here, the functions represented by the descriptions of $\mathtt{l}_3$ and $\mathtt{l}_4$ are properly defined and the lines can be drawn again. In this fashion 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 a sufficiently expressive representational language descriptions can be defined to be as simple or as complex as desired. Descriptions can be used to refer to any kind of graphical object as long as it is syntactically defined.[1] Here it is worth pointing out that the problem of when and how a human-user or the system specifies a description is an interface problem, which is independent of the semantic problem of how descriptions are interpreted. In our current implementation there is a heuristic function that given a graphical context and a pointing action, returns an expression that refers to the current position of the cursor in an informative and relevant way. In a practical implementation a number of tools for allowing the human-user to define, review and modify descriptions during interactive sessions can be very helpful. The point is that whenever we want the system to behave according to our drafting intentions the more explicit the expression of such intentions, the less room for "system idiosyncrasies". However, there are problems that require additional representational devices and problem-solving methods. In the next section we turn to such "synthesis" problems.

[1] For a detailed account of how to specify the syntactic definition and semantic interpretation of graphical languages see, for instance, [12].

3 Naming and Constraint Satisfaction

In the previous section we have discussed the role of descriptions in constraint satisfaction. Here we investigate how constraints on more general graphical properties and relations can be expressed through a representational language, and how such language can help to solve constraint satisfaction problems.

Consider that descriptions are **terms** of the language that denote, or refer to, graphical objects of a certain graphical sort. The name $\texttt{l}_1$ and the description $\texttt{line(d}_1\texttt{,d}_2\texttt{)}$, for instance, denote an object of sort *line*. 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. The notion of constraint, in addition, 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 an intention: 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, an expression representing a constraint differs from a name or a description in that while the former is a modal sentence, the latter are substantives or noun phrases.[2]

Constraints can be embedded within descriptions too. Think of complex names with a relative clause component like *the line that should be parallel to* $\texttt{l}_1$ *and perpendicular to* $\texttt{l}_2$. For representing this mixture of a description and constraints we need powerful representational devices. For instance, an operator that relates a variable and a conjunction of boolean expressions, forming a syntactic construction that is a term of the same sort as the variable in question. For instance,

(8) $\texttt{description(x}_{line}\texttt{): parallel(x,l}_1\texttt{)} \wedge \texttt{perpendicular(x,l}_2\texttt{)}$

Note that such a description would have to be interpreted very much like a variable whose value is constrained by a number of properties. Here, we do not intend to go into the complexities involved with the actual definition of such a kind of structure, but just to illustrate that the different ways to refer to an object within the language and the external attributions made to the same object, contribute and interact in complex ways to the definition of the global constraints that are placed upon the graphical object in a drafting task. In general, the complete description of an object can be distributed in a large number of expressions, although only some of these are nouns or descriptions. Furthermore, the interaction of how a graphical symbol is described with what else is said in the language about the symbol "demarcates" the design possibilities for the object in the world that is represented through the symbol.

Next, we illustrate how this reflection is realised in practice. Consider the shape in Figure 3 and its associated description in (9),

[2] Here, linguistic terms like *modal sentence* and *substantive* are used in the intuitive sense of elementary grammar: a substantive is a noun, and a modal sentence has a modal auxiliary –like *must* or *should*– in it.

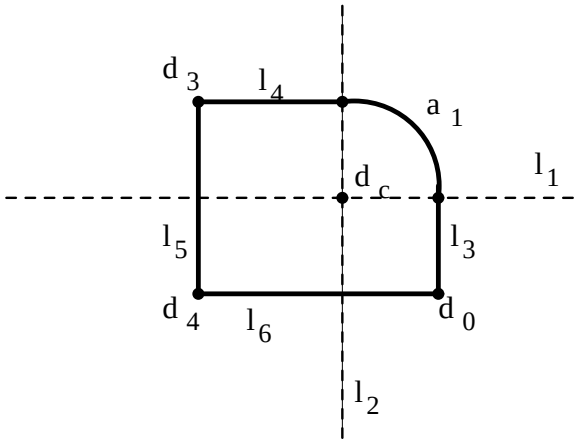

Figure 3: A Constrained Shape

(9) l_1 = c_line(d_c,0)
l_2 = c_line(d_c,1/2π)
l_3 = line(d_0,origin(a_1))
l_4 = line(end(a_1),d_3)
l_5 = line(d_3,d_4)
l_6 = line(d_4,origin(l_3))
a_1 = arc(intersect(l_1,l_2),ρ,0,1/2π)
k_1 = path(l_3,a_1)
k_2 = path(k_1,l_4)
k_3 = path(k_2,l_5)
k_4 = path(k_3,l_6)
$poly_1$ = polygon(k_4)

Note that in (9) there are two different ways of describing a line. While the functor term *line* denotes a function of type dot $\times$ dot $\rightarrow$ line, the functor term *c_line* denotes a function of type dot $\times$ real $\rightarrow$ line. While *line* permits the definition of a line segment in terms of its extreme dots, *c_line* allows us to define a line in terms of a dot and an angle. This latter device is useful for representing construction lines and reference axes. In our example, the intersection between the the axes l_1 and l_2 is the independent parameter that controls the description of the arc a_1, which is defined in turn with the help of the functor term *arc*. This functor term denotes a function of type dot $\times$ real $\times$ real $\times$ real $\rightarrow$ arc. The interpretation of its arguments is as follows: the first is the centre of a circle of which the arc is a segment, the second stands for the radius of the circle, and the third and fourth stand for the angles of two axes –the arc extends from-to– intersecting the centre of the circle. In the definition of a_1 in (9), the first argument is an expression of sort dot, while the other three arguments are real constants. Note that the arc itself can be modified either by changing the parameters of the control axes, or by changing the value of the constant parameters. The constructor term *path*

has nine different homonymous interpretations of the form $\alpha \times \beta \rightarrow$ `path`, where α and β stand for `line`, `arc` or `path`. It permits the recursive specification of a path in terms of its constituent segments, as shown for the definition of $\mathtt{k}_1$ to $\mathtt{k}_4$ in (9). Finally, the polygon's constructor term *poly* denotes a function of type `path` $\rightarrow$ `polygon` and maps simple closed curves to polygons.

In addition to "constructor" functor symbols, like *line*, *c_line*, *arc*, etc., there are a number of "selector" functor symbols, like *origin* and *end*. These functors are interpreted as functions of type `line` $\rightarrow$ `dot`, `path` $\rightarrow$ `dot` or `arc` $\rightarrow$ `dot`. The expression `origin(`$\mathtt{a}_1$`)`, for instance, denotes the origin of $\mathtt{a}_1$, travelling the arc in a counter-clockwise direction.

Consider now that we wish to express that the shape in Figure 4.1 ought to satisfy the following set of constraints across change processes:

(10) {`parallel(`$\mathtt{l}_4$`,`$\mathtt{l}_6$`),vertical(`$\mathtt{l}_3$`)`}

What should be done when the shape is altered in the course of the interactive session as shown in Figure 4?

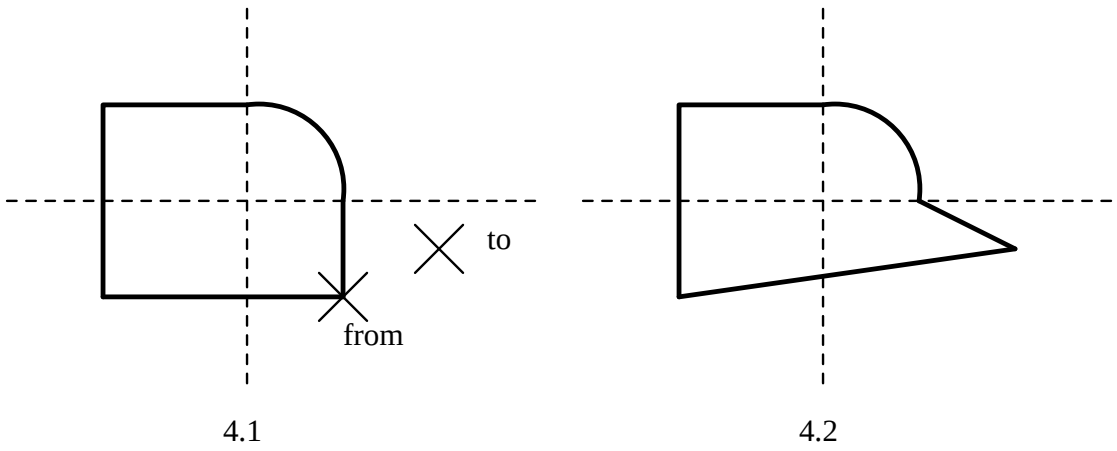

Figure 4: Specification of a Change

A proper policy on naming will not suffice for dealing with the new change problem. The right-end of $\mathtt{l}_6$ and the bottom-end of $\mathtt{l}_3$ are linked to the dot $\mathtt{d}_0$ by their basic descriptions, and these lines are modified by reference. Notice that all expressions in (9) refer to a well-defined graphical object in Figure 4.2, but (10) is not satisfied in the new state. Here we need, indeed, a constraint satisfaction method. However, the change is not fully determined. Description (9) can refer to an infinite number of configurations in which the constraints in (10) are satisfied. One trivial solution is, for instance, to reverse the change to the state in Figure 4.1. For this reason, the constraint satisfaction method will have a semantic contribution to the solution of the problem, and a certain amount of "prescriptiveness" will be inevitable. Next, we consider two options for dealing with the change.

Traditionally the sort of problem exemplified in Figure 4 has been addressed with numerical techniques, like local propagation, relaxation and Gaussian elimination. This tradition goes back to Sketchpad [16] and it is still found in many modern systems, for instance, [1, 3, 4, 6, 14, 15]. However, this approach has the drawback that the actual configuration produced by the system will be 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.

Constraint satisfaction problems of this kind can be solved, alternatively, through a symbolic inference process. This strategy can be implemented with the definition and use of a number of context sensitive drafting rules that modify drawings in specific ways. Our current implementation supports a number of rules for making lines vertical, parallel, horizontal, perpendicular, etc. There are also some rules that can satisfy complex contraints stated as conjunctions of a number of atomic conditions. Consider, for instance, the following two rules in which actual graphical contexts are shown in solid lines, and a number of construction lines are drawn by the rules themselves. The first rule makes a line vertical in a context in which an arc is next to the line in question, as illustrated in Figure 5.1; the second makes two lines parallel, one of which is next to an arc. Although the parallel condition can be satisfied in an infinite number of different modes, our rule proceeds by rotating the line that is next to the arc, as shown in Figure 5.2. Note as well that each rule has a reference dot that acts as a kind of "pivot" for its application. Contexts are defined relative to such reference individuals.

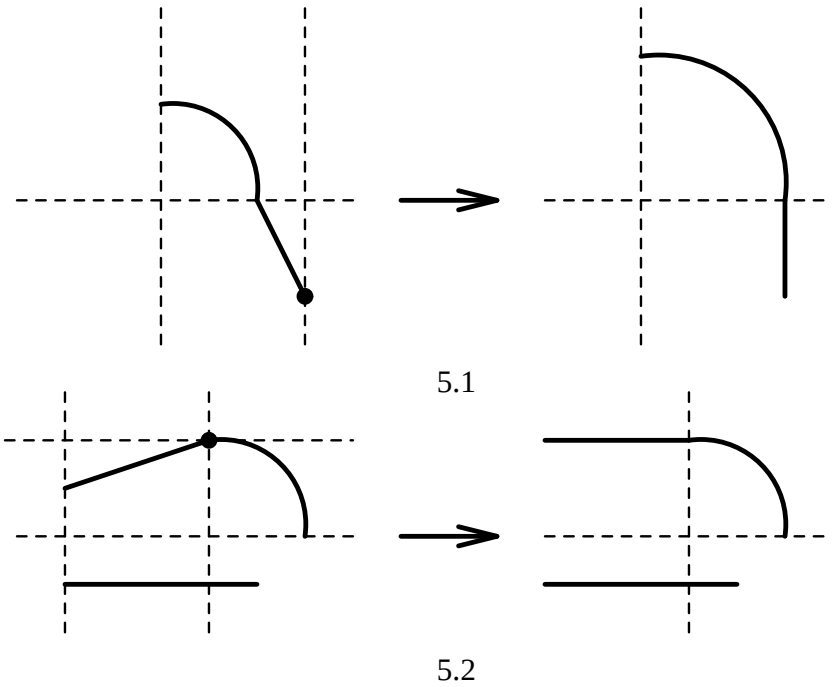

Figure 5: Synthetic Drafting Rules

Now, our change problem can be solved with the sequential application of these rules as shown in Figure 6. Figures 6.1 and 6.2 illustrate the change specification. The transition from Figure 6.2 to 6.3 illustrates the application of rule 5.1. In order to make $\mathtt{l}_3$ vertical, this transformation changes the radius of $\mathtt{a}_1$; however, it preserves the value of the other arc's parameters. The application of rule 5.1 is permitted because there is a topological match between the context of the rule and the context of the drawing, and also because there is a

reference dot available for the application of the rule. This reference is the dot whose position was modified by the user in the original change command. In the transition from Figure 6.3 to 6.4, rule 5.2 satisfies the parallel constraint between $\mathtt{l}_6$ and $\mathtt{l}_4$. The reference for the application of this second transformation is provided by the interpreter in terms of the previous changes made by the system in the current solution path. In the current implementation, a dot whose position is modified by a drafting rule becomes a candidate reference for a further transformation. In order to control cyclic solution paths, whenever a dot is modified by a drafting rule its position is taken to be an invariant property for subsequent modifications of the drawing in the current solution path.

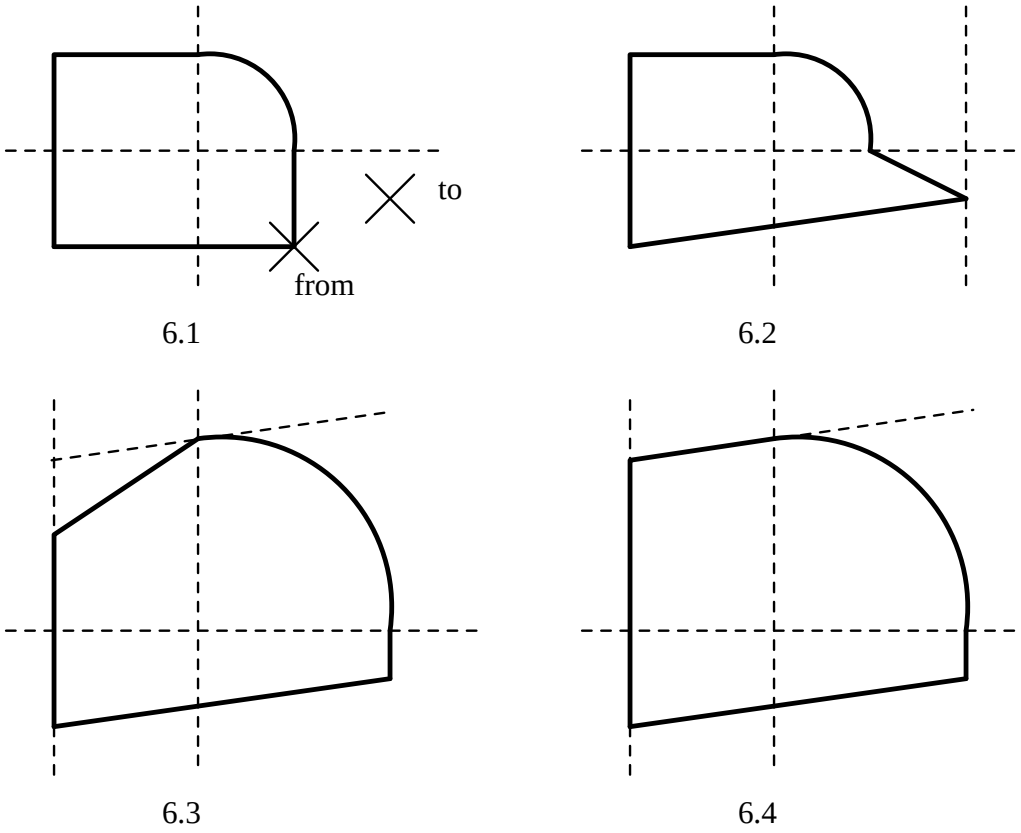

Figure 6: A synthesis Process

Is this the solution intended by the user? If he or she is aware of the drafting rules –in fact, the human-user can define his or her own rules– he or she might foresee the solution. However, this is not in general the case. Knowing a collection of rules, no matter how simple they are, is not equivalent to knowing their logical consequences. In fact, the task is non-deterministic and a given problem can have a large number of solutions. Furthermore, a solution can be found through different search paths. In the current prototype implementation there is an "explain" option through which human-users can inspect how the system has solved a problem. When an explanation is requested, the system produces a sequence of windows illustrating the sequence of rules that have been applied in the solution of the problem. An instance of such an explanation is Figure 6 itself. In addition, the user can inspect all the

solutions that can be found by a given set of drafting rules.

Here, a number of questions in relation to the applicability of the method can be raised. For instance, whether if the system finds a solution it is valid, or whether if there is a solution the method is assured to find it –that is, whether the method is complete. We can confidently say that the answer to the first question is positive; however, the answer to the second is not: to know whether an arbitrary set of geometrical and topological constraints has a solution is an np-complete problem [7]. The most that we can say is that if the descriptions are well-defined and the constraints constitute a consistent set it is possible that there is a solution, but not that there is one necessarily. The solutions accessible to the system will depend on the drafting rules at hand.

In relation to implementational issues we can say that a means-end analysis combined with depth-first search is relatively easy to implement and performs well at least in the experimental setting. The depth-first limitation can be removed in a parallel implementation in which several solution paths can be simultaneously explored. Although the current implementation is sequential, it simulates in many respects a distributed one, which we hope to implement in the future.

4 Conclusions

In this paper we have discussed two problems related to the modification of drawings, or parametric design, in the course of interactive sessions. We have argued that traditional constraint satisfaction approaches based on numerical techniques fail to distinguish some problems of representation and interpretation. We have also argued that a proper naming policy, supported by the use of a sufficiently expressive representational language, helps to solve problems of change by reference, without resorting to constraint satisfaction algorithms. However, naming graphical objects and specifying a number of geometrical and topological constraints upon them are not independent representational problems. For that reason, some change processes are best modelled with the use of symbolic synthetic inference techniques. We have also argued that symbolic solutions to constraint satisfaction problems should be preferred because they can be explained in relation to the semantics of the drafting task, and not in relation to the numerical techniques that are internal to the computer system.

References

[1] Bjorn, N. F. B., Maloney, J., Borning, A., An Incremental Constraint Solver, in: Communications 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] 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).

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

[7] 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.

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

[9] 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).

[10] 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).

[11] 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).

[12] Pineda, L.A., Intensional Representations in Drafting and CAD Systems, in: V Reunión Nacional de CAD/CAM, (Facultad de Ingeniería, UNAM, México, 1991).

[13] 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).

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

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

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