Programming Principles for Visualization in Mathematical Research

W. M. BEYNON†, Y. P. YUNG†, M. D. ATKINSON*, S. R. BIRD†

† Computer Science Department, University of Warwick, Coventry CV4 7AL, UK

* School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

Abstract

We describe programming principles for visualization under development at the University of Warwick. These exploit systems of definitions - resembling the formulae that define the values of cells in a spreadsheet - to specify the structure and interrelationship between mathematical images such as Hasse and Cayley diagrams. The use of a prototype visualization system incorporating the definitive (ie "definition-based") notations ARCA and DoNaLD - for graphics - and SCOUT - for screen layout - is illustrated using a specific mathematical research problem involving combinatorial analysis of planar arrangements of lines. We briefly discuss the practical potential of our methods and possible directions for further development.

1. Background and Motivation

1.1. The role of Visualization in Mathematical Research

Identifying relationships between abstract objects is a common feature of mathematical research. Such relationships are often first discovered by experiment, and subsequently proved to be valid. In the initial phases of mathematical research, it is useful to be able to examine many particular instances of associated objects. This involves developing algorithms to construct one object from another, and representation techniques to make the relevant features of these objects apparent to the mathematician. Graphical representations of objects typically play a crucial role in enabling a mathematician to comprehend relationships between objects. This paper describes programming principles for developing graphical representations of objects - a process of visualization.

1.2. An Illustrative Example

The illustrative example used in this paper is motivated by mathematical research currently in progress [1]. It concerns combinatorial characteristics of simple arrangements of lines, in the sense of Grunbaum [7]. The mathematical concepts and relationships to be discussed below apply to arrangements of arbitrary size, but for our purpose it will suffice to consider arrangements of just 4 lines, such as are depicted in Figures 1 and 2. These arrangements are *simple* because the 6 points of intersection of the lines are distinct.

With each simple arrangement of 4 lines, there is an associated family of representations of the permutation $\rho = (14)(23)$ – the permutation that maps the sequence of indices 1,2,3,4 to its reversal – as a product of adjacent transpositions. Informally, to derive such a representation of

ρ from an arrangement, it suffices to scan the arrangement from left to right (by introducing a scanning line) and record which pair of adjacent lines is transposed as each point of intersection is encountered. Such a scan generally leads to an unambiguous representation for ρ, unless it should happen that two or more intersection points are encountered simultaneously in the scanning process. Note that there can be no ambiguity about the order in which points of intersection that lie on the same line of the arrangement are encountered on scanning.

A partial rather than a total order of the intersection points aptly represents ambiguity in ordering the intersection points when constructing a representation of ρ from an arrangement. We associate with an arrangement A of n lines a poset P of size $N=n(n-1)/2$. The covering relation in P is defined by pairs of intersection points (p,q) and (q,r) where p, q and r are lines and, in the scanning process, the intersection point (p,r) is not encountered between (p,q) and (q,r). There is then a natural correspondence between a family of products of adjacent transpositions representing ρ and the linear extensions of the poset P.

In studying the relationship between arrangements of lines, families of representations of ρ and posets, we use geometric representations for each mathematical object. No loss of combinatorial generality results from normalising the arrangement of lines so that the line i connects the points $[0,a_i]$ and $[b_i,0]$ for $1\le i\le 4$, where $0=a_1\le a_2\le a_3\le a_4=1$ and $1=b_1\ge b_2\ge b_3\ge b_4=0$. Each poset arising from an arrangement of lines can be represented by a Hasse diagram. Each representation of ρ can be represented by a geodesic in a standard Cayley diagram for the symmetric group S_4.

1.3. Characteristics of an Effective Visualization System

Figure 1 depicts a specific instance of the relationship between the three mathematical objects under consideration. It is defined by an arrangement of lines A, a family F of representations of ρ, and a poset P. Merely constructing a diagram to statically display A, F and P is a relatively complex process. When exploring the relationship between A, F and P, many instances must be generated and displayed, such as Figure 2 depicts, for example. Effective exploitation of such a geometric representation demands a convenient method of modelling the relationship between A, F and P by computer. Programming principles appropriate for developing such a "visualization system" are very different from those traditionally used to define graphical displays.

An effective visualisation system must be more than a sophisticated toolkit for modelling. Practising mathematicians who cannot employ full-time computer programmers must themselves specify and implement the visualization process. The relationship between A, F and P above is typical of the kind of relationship that a mathematician might investigate, but it may only have ephemeral interest in the subsequent development of the mathematical research. Modification or refinement of the relationship under investigation is frequently necessary, and radical changes may be required if a new geometric representation of an abstract object suggests itself. The

highly problem-specific nature of the visualization process in this context argues against a comprehensive preprogrammed system built on a conventional programming model.

What then are the requirements for an effective visualization system? It must be

- sufficiently easy for a mathematician who is not a specialist programmer to use
- programmable in such a way that incremental modification is possible
- versatile enough to give the user control over all relevant aspects of the display.

None of these criteria is easily met by conventional approaches to programming.

2. Principles behind our Approach

2.1. Requirements for User-Computer Interaction

Our computational paradigm for interaction rests on a basic premise: if the user is to have control over some aspect of the system behaviour, the mode of interaction must be such that the user acts within a formal framework in which this aspect of the state is faithfully modelled with respect to

- what privileges the user has to change the state
- what consequent change of state occurs when a privilege is exercised.

The primary modelling technique used to ensure this is the principle underlying the spreadsheet, viz the representation of state by sets of variables whose values are specified by defining formulae ("scripts"). It should be emphasised that this technique addresses a concern entirely separate from the development of efficient or ingenious algorithms to evaluate defining formulae.

General principles have been developed to formulate state information using sets of definitions. These are illustrated in their simplest form in the abstract operation of a spreadsheet. Each cell of the spreadsheet can be viewed as a variable whose value is either explicitly defined or specified by a formula expressing its value in terms of that of other variables. The data dependency between variable values established by the set of variable definitions will be assumed to be acyclic. The nature of the defining formulae is determined by an underlying algebra of values and operators. For a conventional spreadsheet, this might be a 2-sorted algebra including scalars and arithmetic operators together with alphanumeric strings and associated operators on strings.

2.2. Definitive Notations

A definitive (definition-based) notation is a simple programming medium in which scripts resembling those underlying a spreadsheet can be formulated. The precise syntax of the notation is influenced by the values and operators that appear in defining formulae, as determined by the underlying algebra. The choice of underlying algebra in turn reflects the nature of the interactive application. Definitive notations developed in our work include ARCA – for displaying combinatorial diagrams, DoNaLD – for line drawing, and SCOUT – for screen layout.

ARCA was the first definitive notation to be developed. The data types in its underlying algebra were designed to represent n-dimensional realisations of combinatorial graphs. Because the model adopted for such graphs is based upon the Cayley diagram, the edges may have associated colours and directions. ARCA is well-adapted for expressing the kind of information that arise in research areas such as automata theory and combinatorial group theory. For instance, it can be used both to specify abstract finite automata and to describe a layout for its realisation as a graph. Defining formulae in ARCA can be used to establish rich data dependencies such as are needed to express symmetries between component parts of a Cayley diagram. For instance, if x and y generate a finite group and f and g are elements such that $f = g.x.y.x^{-1}$, we can assert that the node of the associated Cayley diagram representing the element f is defined by rotating the node representing g through $2\pi/3$ about the origin. For further details of ARCA, see references in [4].

The edges of a combinatorial graph such as a Cayley diagram are abstractly defined by adjacency relationships between vertices. In ARCA, the directed edges of a particular colour within such a diagram are specified by a (partial) permutation of the indices of its vertices. DoNaLD – a definitive notation for line drawing, is complementary to ARCA. Its basic data types are points and lines in the plane. Whereas the use of ARCA is appropriate when the abstract vertices and edges of a graph have an interpretation independent of their geometric realisation, DoNaLD is adapted for describing aggregates of points and lines whose interpretation is rooted in their geometry alone. Further details of DoNaLD, and examples of its use, can be found in [2,3].

SCOUT is designed for describing screen layout [4]. A SCOUT script abstractly specifies the nature, content and location of a set of windows and how they are composed to make up a display. The specification is abstract in the same sense that a window manager delegates control of particular windows to other application programs rather than specifying their content directly. For instance, the SCOUT specification of Figure 1 is made up of windows containing ARCA and DoNaLD pictures and textual commentaries. Much of the expressive power of this method of representing the state of the screen display derives from being able to describe relationships between information represented in many different forms. This is illustrated in [4], in which the role of SCOUT in specifying interfaces is discussed in detail.

3. Technical Details

3.1. A Script to Define Figures 1 and 2

The script that defines Figure 1 comprises about 380 definitions: these include

- SCOUT definitions to lay out windows and supply a textual commentary
- DoNaLD definitions to define the arrangement A and poset P'
- ARCA definitions to specify the Cayley diagram S4 and poset P.

Defining the poset P associated with the configuration A so that the data dependency is modelled correctly is technically difficult, but can be done in DoNaLD. It involves translating the definition of the poset as described in §1 into a geometric dependency relation. The realisation of the poset most easily generated is P' as in Figure 1. A typical element of P' represents the point of intersection of a pair of lines i and j in A; if this intersection point has coordinates (X,Y) – as coordinatised in §1, then its coordinates relative to a suitable origin for the Hasse diagram of P' are (X',Y'), where X' is determined by the number of lines that pass above the point (X,Y) on the line x=X in arrangement A, and Y' = X. To be more explicit, the x-coordinates of points in the posets P and P', modulo a scale factor, can be defined by integers representing the index of the adjacent transposition associated with the intersection of lines i and j. The y-coordinate of an element in P would normally also be represented, modulo a scale factor, by an integer value - its rank as an element the poset. In practice, this rank can be approximated simply by recording when a scan line parallel to the y-axis encounters the associated intersection in A.

These observations determine appropriate DoNaLD definitions. The arrangement A is specified in terms of real parameters L12, L23, L34 and R12, R23, R34 that respectively determine the distances between the left- and right-hand endpoints of lines in the arrangement. These parameters in turn fix the relative size of the six ratios:

r12 = L12 / R12, r23 = L23 / R23, r34 = L34 / R34

r13 = (L12+L23) / (R12+R23), r24 = (L23+L34) / (R23+R34)

r14 = (L12+L23+L34) / (R12+R23+R34)

which define the y-coordinates for points in P'. From these ratios, it is also possible to infer whether or not – in the L-R scanning order – the intersection of the line k with the line i is preceded by the intersection of the line j for each index j≠k. This knowledge in turn determines the x-coordinates of points in P'. For instance, it may be seen that the x-coordinate x23 of the point at which lines 2 and 3 intersect is the number of lines passing above the point of intersection of the lines 2 and 3, as defined by the formula:

1 - **int** (r12<r23) + **int** (r14<r24),

where **int** designates conventional coercion of boolean to integer values.

Having determined the positions of the points in P', it remains to specify the covering edges of the poset. The form of the boolean condition required to specify whether the line joining a pair of intersection points involving a common line is inferred from the definition of the poset P in §1. As an example: the line joining the points p13 and p23 that respectively depict the points of intersection of lines 1 and 3 and lines 2 and 3 is a covering edge subject to the condition

d1323 = (x13≠x23) **and** ((r34-r13)*(r34-r23) > 0).

The poset P is defined from P' using ARCA. The definition of P in ARCA imitates the way in which a mathematician interprets P' as a Hasse diagram, converting perceived geometrical

relationships into an abstract order relation. The points of P are vertices of an ARCA diagram whose incidence relationship is determined by geometric properties of P' that can be defined in DoNaLD. The edges of P are directed to reflect the order relation in P and coloured according to their L-R orientation. For instance, we need to know if the edge (p13,p23) is present, and if so, how it is oriented L-R and up-down. The incidence structure of P is defined in ARCA in terms of the values of appropriate DoNaLD variables such as d1323. The vertices of P are given integral coordinates respectively defined by the x-coordinate of P' and the length of a shortest combinatorial path from a minimal element of the poset P. Distance in a combinatorial graph is used in a similar way in conjunction with projection of a truncated cube to define the Cayley diagram S_4 in ARCA. The definition of the path to represent geodesics is discussed in §4.2.

3.2. Scripts as the Basis of a Visualisation System

The use of scripts to generate displays such as Figure 1 and 2 has many attractive qualities. The user can build up the components of a picture incrementally and independently and introduce links between picture elements that are described in conceptually quite different ways. Because the script of definitions represents a particular state of the final picture, rather than the cumulative effect of a sequence of procedural actions, the way in which the user can interact with the visual image is very open-ended. Changing a parameter transforms Figure 1 to Figure 2 for instance. Subject to a good choice of script, the scope for manipulating a picture compares favourably with what can be achieved by providing the user with a customised menu-driven interface where the privileges to amend the state of the picture are already wired-in. This emphasises the difference between our approach, directed at the informal use of visualisation techniques, and approaches that involve encapsulating knowledge about how the user may wish to experiment with a visual image and precompiling a program to allow these specific experiments.

Some specific features of typical interaction with Figures 1 and 2 will illustrate this point:

- exceptions are a common problem in geometric display. In our example, some of the ratios r12, r23 etc may be undefined, so that some points of P' cannot be displayed. In our interpreters, variables whose values are currently undefined are handled routinely.
- the DoNaLD script defines P' as a continuous transformation of the arrangement A. As A is deformed continuously between combinatorial forms, the poset P' passes through a singular states in which covering edges coincide. Near such singularities, the user can compensate for loss of definition in P' by changing the vertical scale.
- a user experimenting with mathematical objects typically wishes to make minor changes e.g. introducing pointers or modifying the presentation or annotations of figures to reflect different features of the image. Interactively revising a script is much more convenient than the tiresome process of modifying a conventional program for geometric realisation.
- definitions have powerful latent effects e.g. the script for Figure 1 *defines* the number of minimal triangular regions. This is 2 as ρ is currently set, but applies to any choice of ρ.

3.3. Implementation Issues

The DoNaLD, SCOUT and ARCA prototypes are based upon the programming language EDEN[3] that was primarily designed to implement definitive notations in a UNIX/C environment. The EDEN interpreter automatically monitors data dependency and updates variable values efficiently through selective re-evaluation. Its features include definitive variables of basic C types (viz. lists, strings and scalars), conventional procedures, and actions in the form of procedures whose execution is triggered by changes to the values of specified variables. Generic ways of interfacing with system utilities such as X-windows have also been developed.

Each definitive notation is implemented by devising EDEN representations for the data types and operators in the underlying algebra. Input definitions can then be translated into EDEN definitions at a lower-level of abstraction. Where the value of a variable represents the state of a displayed entity, such as a geometric point or shape, EDEN actions to maintain the display must also be generated by the translator. This implementation technique can be adapted to permit the definition of predicates to act as monitors – displaying messages as and when particular boolean conditions prevail, or as imposed constraints – revoking a user definition leading to a violation.

At present, the practical integration of several definitive notations within a single SCOUT interface, as in [4], uses EDEN as a common form of intermediate code in conjunction with an interface to X-Windows. EDEN then provides a uniform internal representation of definitions allowing data dependencies between different types of windows to be established.

4. Evaluation and Future Directions for Research

4.1. Definitive State Representations and Informal Semantics

Experiments with our present prototypes clearly show how in principle scripts of definitions can be exploited in constructing a visualisation system in which

- a user can exercise the privileges that normally belong only to the programmer
- programming and reprogramming can be carried out incrementally
- a user can adapt the visualisation process to meet unforeseen requirements

To understand the full significance and potential for these methods, it is helpful to contrast environments based upon sophisticated extension of *calculator* vs *spreadsheet* models.

Using a computing environment for mathematical research involves manipulating many different kinds of information. Knowledge of mathematics, of system response, and even about the operation of the computer itself may be required. A calculator relieves a mathematician of few problems in organising and recording this knowledge. Intermediate results can be stored electronically, but this does not assist their interpretation, upon which the computational strategies adopted by the user crucially depend. To be precise, the calculator only encapsulates

generic knowledge (e.g. about arithmetic relationships and operators, such as "what is the result of multiplying x by y"): in no way can it be adapted by the user to reflect problem-specific knowledge (e.g. "what it means to say that p denotes the profit from a transaction").

The spreadsheet has greater potential in this respect. By introducing a formulae that defines the profit (p) in terms of manufacturing cost (mc) and selling price (sp), a spreadsheet user creates a simple state-based model of a perceived relationship between values. The claim that this mode of representation assists the user in interpreting the computations performed in application-oriented terms can be justified: a third-party, having no knowledge of the intended meaning of the spreadsheet cells that display the values of p, mc and sc can verify that this is an appropriate interpretation either by referring to the defining formula for p, or – with less confidence – by observing the results of experimenting with the values assigned to mc and sc.

Definitive scripts derive their power from the simple principle illustrated in the spreadsheet. A script can be used to represent knowledge about functional relationships between data that persist throughout transition from one state to another. Subject to the usual limitations of the experimental method, this knowledge can be gained through experiment within the application (e.g. as when a mathematician studies many configurations of lines in Figure 1 to discover how to construct a configuration of 4 lines to realise any given shortest path in the weak ordering of S_4), then encapsulated through the formulation of appropriate sets of definitions (e.g. formulae to express the parameters of a configuration of lines in terms of a choice of shortest path).

Our underlying thesis is that a system of variables represented in the computer is most easily interpreted by the user if the data dependencies associated with interpretable transformations of their values are also specifed. Definitive representations of state play a fundamental part in this specification. This link between the interpretation of variables and definitive state-transition models explains the special qualities of the spreadsheet paradigm. In such environments the user can dynamically construct faithful models of external state-transition systems, representing knowledge that is specific to the problem and the context, rather than generic and preconceived. These external systems may include abstract objects (e.g. the relationships between mathematical entities as in Figure 1), physical objects (e.g. the relationships between objects in a room, as modelled in [6]), or reflect the computer environment itself (e.g. the current state of the screen display). The applicability of this technique is determined solely by whether the underlying data types and operators can conveniently represent the perceived state of an external system.

4.2 Limitations of our Present Prototypes

We have identified two factors that determine the expressive power of a definitive script: the richness of the underlying algebra, and the clarity of its interpretation in state-transition terms. The definitive notations DoNaLD, ARCA and SCOUT have underlying algebras with relatively

simple operators: the values defined by scripts in these notations are components of visual images with state-transition interpretations readily accessible to the user. As Figure 1 illustrates, SCOUT links together scripts based on diverse underlying algebras in a single context. In effect, this unification is achieved by exploiting the fact that all the state changes being represented to the user are ultimately registered as actual physical changes to the display. This meets the needs of visualisation, but another kind of unification of definitive notations is ideally required. The specification of the path representing the family of geodesics F in Figure 1 illustrates this point: this path is defined by the linear extensions of the poset P (e.g. 1,6,5,2,3,4) that are first transformed into a sequence of x-coordinates (e.g. 1,2,3,1,2,1) and then interpreted as a product of generators to obtain a path through the Cayley diagram S_4. This process cannot be represented directly by an ARCA definition; in our specification for Figure 1, this data dependency is maintained by introducing EDEN functions and an EDEN action to manipulate the intermediate representation of an ARCA variable.

In their present form, our prototypes are not yet appropriate for informal use. This is illustrated, for instance, by the technical difficulty of expressing the data dependency between an arrangement A and its associated poset P in DoNaLD, as outlined in §2 above. Of course, some conceptual difficulty in understanding the relationship between P and A is to be expected – describing this relationship formally is an essential part of the mathematician's task. Yet even when the relationship is understood, it is tedious to express it in DoNaLD definitions.

There are two aspects to this problem: we ideally require

- more powerful ways of constructing scripts e.g. including convenient methods of specifying families of similar definitions
- a richer underlying algebra e.g. including sorting operators for lists of scalars.

For consistency, we would also wish to exploit definitive principles in introducing these features. How should this be done?

Experimental work suggests solutions to these problems. A functional programming language such as Miranda [8] has an extraordinarily rich underlying algebra of values and operators. Typical Miranda scripts have characteristics complementary to DoNaLD scripts: they describe sophisticated functional relationships, but it is difficult for the user to attach a state interpretation to the script. It is relatively easy to develop a Miranda script that can generate DoNaLD scripts through evaluation of variables: in this way, we have generated DoNaLD scripts to describe A and P in Figure 1 for larger values of n for instance. By editing the Miranda script, the user can exercise privileges more powerful than those permitted by the use of a DoNaLD script alone, e.g. changing the number of points in the arrangement, or formulating different dependency relationships (as when either generating a script in which the appropriate dependencies between the components of Figure 1 are described, or simply generating a script to display Figure 1).

How should we interpret this use of Miranda and DoNaLD scripts in combination? Miranda and DoNaLD scripts can be integrated within a unifying definitive paradigm, as follows:

- introduce admira, a definitive interface to Miranda
- interpret DoNaLD scripts associated with Miranda variables as values that can be given interpretations by the user analogous to the values of DoNaLD variables
- conceive an underlying algebra appropriate for the generation of scripts as values.

The admira prototype has already been developed.The idea of 'script as value' requires careful analysis of the external state-transition model concept discussed in §4.1. Informally, we contend that asserting that the value of a DoNaLD variable represents the present position of a particular point on the screen is in principle no different from asserting that the value of a Miranda variable is a description of Figure 1 as an object with a particular inherent state-transition interpretation modelled by a set of data dependencies. Finally, general-purpose operators in Miranda can be adapted for the macro generating activity that is best suited to describing scripts abstractly.

Conclusions

We have developed principles and prototypes that offer good prospects for the eventual development of visualisation systems that are simple and flexible enough to be used without deep technical programming knowledge by research mathematicians. From a theory of programming perspective, our research indicates dual roles for visualisation – which is primarily concerned with the way in which the state of a computer system is represented to the user – and declarative programming (as represented for instance in Miranda) – which is concerned with powerful models of computation making state invisible to the user. The significance of our methods lies in the synthesis of these two aspects of programming that they achieve.

Acknowledgments

Major practical contributions have been made by Yun Wai Yung, and seminal ideas developed in collaboration with Steve Russ and Alan Cartwright. The principal author is grateful for financial support from a RS/NSERC exchange award and the SERC-funded Automatic Groups Project.

References

1. M D Atkinson, W M Beynon (in preparation)
2. W M Beynon, Y W Yung Implementing a definitive notation for interactive graphics New Trends in Computer Graphics, Springer-Verlag 1988, 456-468
3. W M Beynon Evaluating definitive principles for interactive graphics New Advances in Computer Graphics, Springer-Verlag 1989, 291-303
4. W M Beynon, Y P Yung Definitive Interfaces as a Visualisation Mechanism Proc Graphics Interface '90, Canadian Information Processing Society, 1990, 285-292
5. R Bird, P Wadler Introduction to Functional Programming, Prentice-Hall, 1989
6. A Bjorner Orderings of Coxeter Groups, Combinatorics & Algebra, Boulder, AMS 1983
7. B Grunbaum Arrangements and Spreads Reg Conf Series in Math #10, AMS 1972
8. Research Software Ltd The Miranda Manual
9. B Wyvill An Interactive Graphics Language, Ph Thesis, Univ of Bradford, 1975

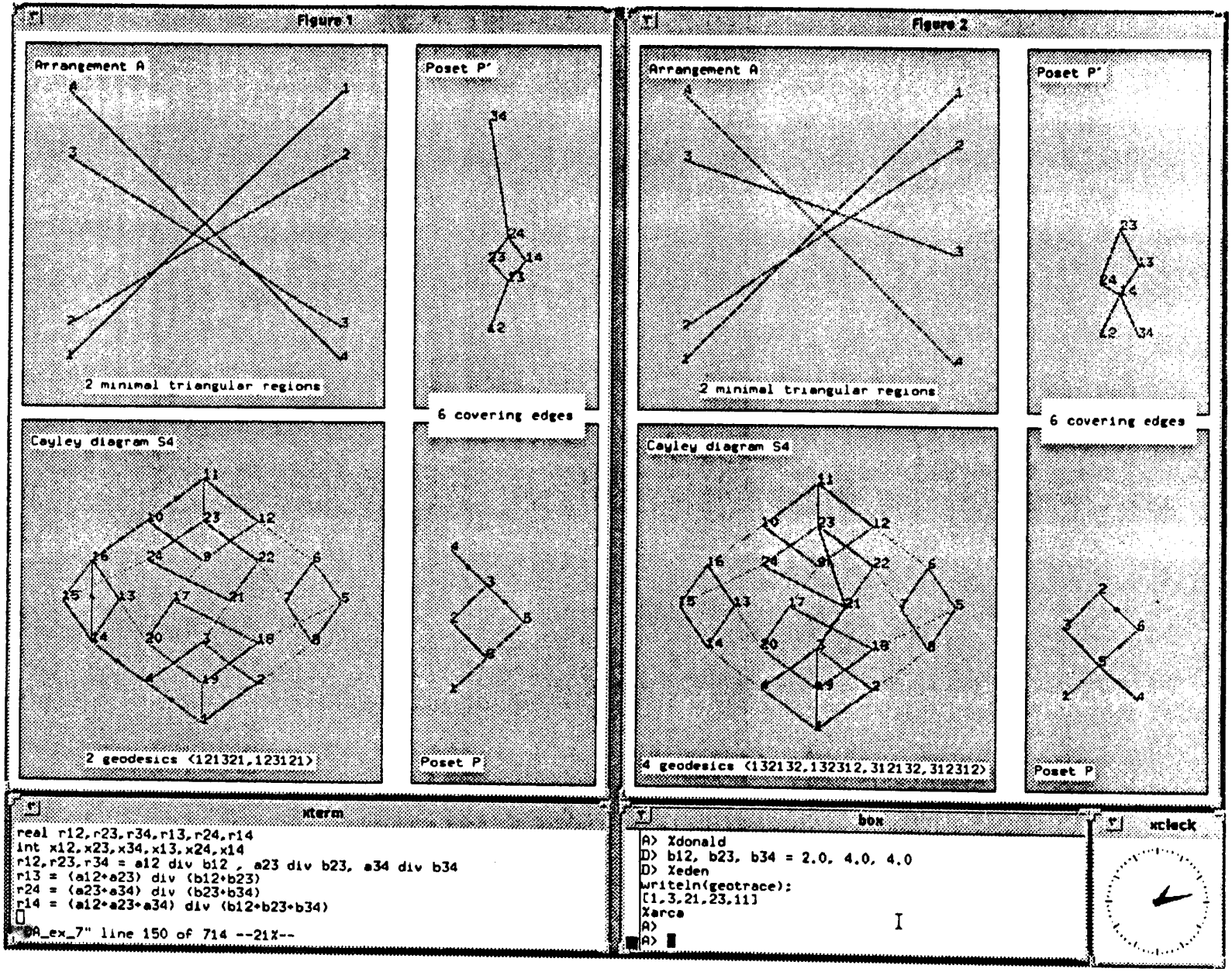
Figure 1
Arrangement A
2 minimal triangular regions
Poset P'
6 covering edges
Cayley diagram S4
2 geodesics <121321,123121>
Poset P
Figure 2
Arrangement A
2 minimal triangular regions
Poset P'
6 covering edges
Cayley diagram S4
4 geodesics <132132,132312,312132,312312>
Poset P
xterm
real r12,r23,r34,r13,r24,r14
int x12,x23,x34,x13,x24,x14
r12,r23,r34 = a12 div b12 , a23 div b23, a34 div b34
r13 = (a12*a23) div (b12*b23)
r24 = (a23*a34) div (b23*b34)
r14 = (a12*a23*a34) div (b12*b23*b34)
"@A_ex_7" line 150 of 714 --21%--
box
A> %donald
D> b12, b23, b34 = 2.0, 4.0, 4.0
D> %eden
writeln(geotrace);
[1,3,21,23,11]
%erca
A>
A>
xclock