Computation and Images in Combinatorics

CalICo Group*
This paper was completed by
M. Delest, J-M Fédou, G.Melançon and N. Rouillon

LaBRI, Université Bordeaux I,
Unité associée au C.N.R.S. n° 1304,
33405 Talence Cedex, France.

September 7, 1995

1 Introduction

Combinatorics has always been concerned by images and drawings because they give interpretations of enumeration formulae leading to simple proofs of these formulae, and sometimes they are themselves central to the problem. Even if some small example drawings do not contain all elements of the proof, they are often useful to guide the intuition.
On recent years, enumerative combinatorics has developed from this point of view, in interactions with theoretical computer science and graphics computer science. See for example [DAV89]. In the field of bijective combinatorics linked with computer science, the DSV-methodology [VIE92], [DEL95] has led to new formulae . The problem often consists in esablishing enumerating formulas by constructing bijections between sets of objects having the same cardinality. Computations are then suppressed and are replaced by effective constructions of bijective correspondences between sets. The formulae may then be viewed as reflecting combinatorial properties of the objects involved. The problem may also consist in finding geometrical transformations of objects reflecting the equations which relate them.
For several years already, enumerative combinatorics research uses symbolic computation. Most of the Computer Algebra Systems, including Macsyma [SYM84], Maple [CGG+92] and Mathematica [WOL88], allow easy algebraic and graphic manipulations but none of them gives library of functions working

*When this publication was initiated the contributors to the CalICo project were: Y. Chiricota, M. Delest, J.M. Fédou, V. Gaudin, G. Melançon and N. Rouillon - e-mail:calico@labri.u-bordeaux.fr

directly and interactively on the graphic representation of the object. Some projects have leaded to tools for performing such actions. For example, we can quote the Mathematica library done by Skiena [SKI90] which give tools for implementing combinatorial objects. This library do not allow to interact with the graphics itself. The aim of the CalICo project is the development of a software in which the manipulation of combinatorial objects can be made through all their representations: coding, formal power series and overall pictures.

1.1 A famous example.

Let us discuss one famous example: the Robinson-Schensted correspondence (see [dBR48], [SCH61], [VIE77] for details). In the theory of representations of finite groups, there is a formula relating the number of elements of a group G and the dimension of its irreducible representations. The case $G = S_n$, the group of permutations of $\{1, \ldots, n\}$, has been studied in great detail over the years. In that case, irreducible representations correspond to Ferrers diagrams filled with distinct integers to form *standard Young tableau* [1] (see figure 1.1 below). The equality then reads:

$$\sum_{\lambda} f_\lambda^2 = n!,$$

where f_λ is equal to the number of standard Young tableau with underlying Ferrers diagram corresponding to λ, for more information about Robinson-Schensted correspondance in connection with the theory of group representations, see for instance [KER91].

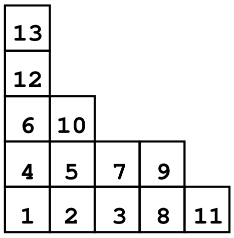


Figure 1: A standard Young Tableau.

Enumerative combinatorics then plays with algebra to give some insight on this formula. The Robinson-Schensted correspondence shows that there is a bijection between permutations and ordered pairs of standard Young tableau.

[1]Ferrers diagrams are built by piling rows of cells. Rows are left justified and decreased in size. In a standard Young tableau, labels in rows and columns increase from left to right and from bottom to top respectively.

This correspondence is in fact an algorithm taking as input a permutation and constructing a sequence of ordered pairs of tableau by reading the permutation from left to right. Many combinatorial properties of the permutation can be read on the tableau associated with it. For instance, taking the inverse of the permutation corresponds to exchanging tableau. In other terms, if (P,Q) is the pair of Young tableau associated to a permutation σ then (Q,P) is associated to σ^{-1}. This phenomenon has been explained in an elegant manner by Viennot [VIE77]. His methodology conforms to the point of view mentioned above and motivates the development of a project such as CalICo.

1.2 A piece of history.

We would like to say a few words about the history of CalICo. CalICo was initiated by M. Delest in 1989. One of the key problems in combinatorial work consists in finding an appropriate object representation, i.e. one which is useful with respect to certain transformations or manipulations of the objects in question. The design idea of CalICo was to provide a computing environment to assist this process in an essential way. For example, one should work with a grid of points instead of the standard representation of a permutation as a word. CalICo was designed to be an experimental laboratory and wishes to offer a graphicaly rich environment that facilitates working in enumerative combinatorics. Eventually, it should be a guiding tool for finding bijections between family of objects. A component of CalICo would gather informations about objects and organize them in order to assist the user. That component would be a tutor (see sections 4 and 5) for studies in combinatorics. In 1990 the project entered a concrete stage. After a primitive realization of the tutor and a first realization of a workshop [2] for manipulation and edition of polyominoes, some combinatorists (at LaBRI, Bordeaux) saw immediately how CalICo could be a tool of great help. Some of them joined the project and proposed visualization workshops for other families of objects. A visualization workshop for braids in 1991, and one for permutations in 1992 were added to the preceding workshop for polyominoes.

Although the need for a communication manager was obvious from the first day, it was completed at that time [ROU94b]. At the beginning of 1993, a second version of the tutor was developed and the visualization workshops were improved. After demonstrations at scientific meetings, the enthusiasm expressed by the scientific community and the growing demand for installing CalICo on many sites gave the project a new impulse. The project entered a new stage. The main concern was the need to unify the appearance of visualization workshops, to standardize many of their functionalities and to systematize the development of new visualization workshops. With the help of new members, the structure of a standard visualization workshop was elaborated; a skeleton including common

[2] see the chapter *Overview of CalICo* for all definitions. Just note that a workshop is an application (or software) integrated in CalICo environment.

functionalities was developed. The foundations of CalICo's architecture were set right from the beginning. Because of the many components interacting and communicating with each other, the software may be viewed as having a distributed architecture (see section 7).

In [MDMR94], we showed how CalICo could be used to illustrate bijections such as the Robinson-Schensted correspondence. We also suggested how one could go beyond a simple illustration and benefit from CalICo to actually look for new bijections between families of objects. This paper intends to give some insight into CalICo. Working with CalICo means working with pictures on which symbolic computations are performed. We discuss the architecture of CalICo made up of several workshops, revealing a dual existence of objects, both as geometrical and formal objects. We give explicit details on how objects are coded and how one may proceed to manipulate them using symbolic computations, thus describing the interactions between the different workshops. The full power of CalICo is achieved when combining its visualization workshops with a symbolic computation workshop, thus exploiting the double identity of objects. We illustrate by an extended example how one may proceed and compute on objects, working with pictures and actual data on the objects.

Now turn back to the present paper which describes CalICo, see [ROU94b] for complete description. In the next section, we show an overview of its structure. To guide the reader in this paper, we show simple examples. Then we describe the tutor (which concerns combinatorics knowledge) that is the heart of the CalICo environment. The next section is devoted to the formal coding and to links with computer algebra systems. In section 6, we describe the primary interface that allows the user to launch an application in the system and the model that we have designed to produce easily new interfaces. Some examples of this model are also given. Section 7 is devoted to the software of communication.

2 Overview of CalICo

In this section, we describe the general concept of CalICo. CalICo gathers in a single software environment a set of different applications, called *workshops*. These workshops can be run on remote computers linked by the Internet network. This software environment allows message exchange between each of those workshops. It offers a graphical and mathematical working environment.

We use the term *workshop* instead of the more usual terms *application*, *program* or *software* because it gives the intuitive notion of workshops working together in a factory. A workshop in CalICo is independent of the others and dedicated to a specific job. Independence means that all the workshops could be used without CalICo. Of course, in this case, the workshop loses some of its functionalities.

Various workshops coexist within CalICo. The *tutor* is the main workshop. The others could be divided in three groups: visualization workshops, computation workshops, generating workshops. All of them work in a concurrent way. They are gathered together in a distributed architecture. The client/server model is used.

2.1 Workshops

The main workshop: the tutor. The tutor is the heart of the CalICo environment. Its functions are to supervise the system and to assure combinatorial coherence.

This part of CalICo is actually specific to CalICo (see sections 4 and 5). It is exclusively dedicated to enumerative and algebraic combinatorics. The tutor is able to classify mathematical objects according to their combinatorial properties. It offers experimental ways in order to suggest to CalICo's users possible one-to-one correspondences between the manipulated structures.

Visualization workshops. Many researchers in combinatorics handle both mathematical formulae and graphical representations of combinatorial objects. A very enhanced example is in X. Viennot's work [VIE88] who often uses the sentence:

"Dessiner des calculs, calculer des dessins [3]*"*

CalICo includes a set of workshops which allows visualization and handles various combinatorial structures (see section 6).

Computation workshops. As we saw before, it is not enough to only manipulate graphical objects. To complete, we have to associate computational workshops with graphical workshops. This will allow the manipulation of formal coding of objects (see section 5). For the moment, CalICo integrates the package *Maple* [CGG+92]. This comes from the fact that we wrote in Maple the main workshop which is the tutor (see section 4). CalICo also integrates the package *Mathematica* [WOL88].

Generating workshops. To compute significant statistics (or properties) on objects, people often need to consider large random objects. So, CalICo includes generating workshops. In section 6 we present algorithms used in some of these generating workshops.

2.2 Collaboration between workshops

The ultimate goal is to produce a system that can be operated even by a non computer scientist and to use methods leading to easy extensions and generalizations of already existing modules.

[3] *"Drawing computations, computing drawings."*

On the one hand, we group (see section 7) in a single software environment all the workshops we need. We propose a user-friendly graphical interface to allow communication between workshops.
On the other hand, a *distributed architecture* has been chosen. Each workshop can be run on a remote computer. This distributed aspect allows parallel treatments of several processes. The interest is in running generating workshops on different computers to increase the speed of the system.
Workshops could also be dynamicaly visualized on a remote display. The interest is double. First, one can use several terminals or computers to display its work. Second, this opportunity allows us to visualize the result of a computation (by way of a drawing with colours) on the display of a person in another city or another country. This last notion is very close to the groupware field [BLK92]. In this aspect, we did not use X11 functions. Only the data giving the coding of the objects are transferred between two remote applications (see section 7).

From a technical point of view, the implementation of this work is based on the *client/server model*. GeCI (**Ge**stionnaire de **C**ommunications **I**nteractif [4]) is a unifying program. It manages dynamic connections between programs (even on remote computers) and the exchange of messages between them (see section 7 for more details). We want to enhance a particular point: the data are transmitted if they are on the same display by a copy/paste operation even if the workshops are running on different computers.

2.3 Configuration and Extensibility

CalICo can be configured at different levels. Some configurations can be done by a simple user, others need to know more about computer systems.

Dynamic configuration - simple user. With the mouse, the user can run (or kill) dynamicaly during a CalICo's session as many workshops occurrences chosen among available workshops as he wants. The user can run (and visualize) these workshops on any computer on the Internet network registered in his installation of CalICo (see next paragraph). We can say that the user creates a "work space on the net" and builds a *virtual computer* with which he will work.

Personal configuration - simple user. This configuration barely suppose to learn the configuration file syntax. When CalICo is launched, its *configuration file*, called `.CalICo.config`, is read by a scanner and a parser. This file completely describes the *piloting interface* (see the section 6 on graphical interfaces). For example, the computers and displays allowed for the current user are listed inside.

New workshop - programmers. Adding a new workshop does not need special knowledge. It consists in adding few lines to the configuration file of

[4] *ie:* Interactive Comunications Manager

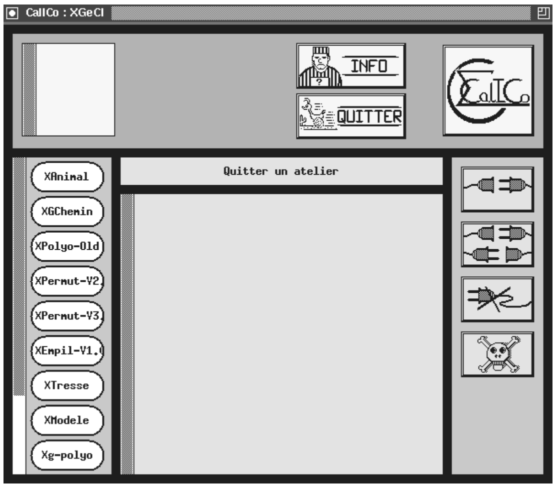


Figure 2: Piloting interface at the starting up

CalICo. On the contrary, the conception of a new workshop is a programmer job. His work will be greatly facilitated by the use of the *CalCom* library [CAL93] and the CalICo model workbench [GAU95b, GAU95a].

New combinatorial knowledge - Maple users. The user adds a combinatorial knowledge to the tutor by means of a graphical interface. The user has to know the *formal coding* of the object he wants to manipulate (see section 5) as well as CalICo's terminology. To add significant information, it is better to know *Maple* (which is the programming language of the tutor) and combinatorics.

Bellow, we discuss more precisely all those various CalICo's aspects.

3 The use of CalICo as illustrated by examples

When one issues the `calico` command at the top level shell, its *configuration file*, which name is *.CalICo.config*, is read by a parser and a scanner [5]. This file has to be found in the user's home directory in the directory *.CalICo*. If it is not found there, a file by defect is copied from the system. This file contains the description of the available workshops and therefore describes exhaustively what we call piloting interface for CalICo (see figure 2).
On the left side of the window of the piloting interface, one finds all available workshops. On the right side, the buttons allow to connect the output of a

[5] GNU Products, Flex and Bison.

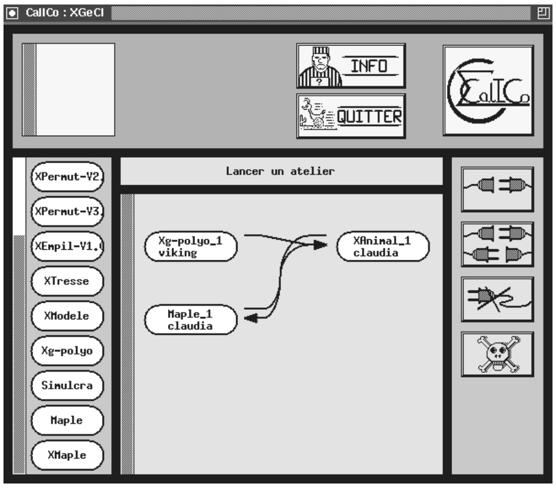


Figure 3: Piloting interface after connections

workshop to the input of another and to destroy a launched workshop. As an example, launch successively the three workshops, *Xg-polyo, XAnimal et Maple* and connect *Xg-polyo* to *XAnimal* and *XAnimal* to *Maple* (and reciprocally). We get the figure 3.

When the user launches a workshop, a window gives him the possibility to change the name of this workshop occurrence. One can choose for the execution (and/or the visualization) machine other than the local one. If the user chooses a remote machine for the execution, it is necessary that CalICo daemon process called *calicod*, runs on this machine. Running this daemon process is achieved by issuing the `calicod` command at the top level of any user on the machine. On a workstation, user must remember to add this host to the list of machines that are allowed to connect to the local X server. Use server access control program for X and execute `xhost <hostname>`.

The launching is done by clicking the *OK* button. For example *Xg-polyo* (figure 4), generator for directed animals is launched on a remote machine (here viking, the local machine being claudia as illustrates on figure 3). *Xanimal* (figure 6), workshop for animals and polyominoes is launched on the local machine claudia.

The motivation of such configuration is a work presented in [DR92]. The work has been realized using the computer algebra package *Maple*. This is why a third workshop appears in the figure 3.

Let us consider another situation when one wants to test some Maple functions on a heap of pieces [VIE86]. In this case, we need a big number of input/output

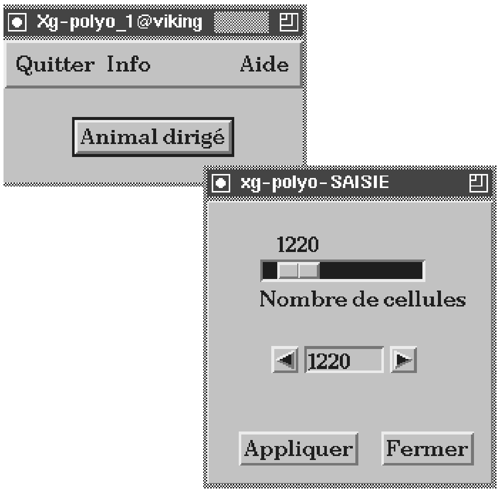


Figure 4: Random generation of animals.

moves between the manipulation of heaps in the graphical interface and Maple. Here one desire rather to use *Xmaple* to have the ease of use of the IRIS interface. Exchanges can then be made by *copy/paste* using the mouse and the CONTROL button [6].
In the workshop *XEmpilement*, by a CONTROL-LeftButton action in a heap (figure 9) one can select a copy of the formal coding underlying the heap. In *XMaple*, a MiddleButton action pastes the coding as observed on the figure 8. Reciprocally, in *XMaple*, after application of the function f on the heap, one can copy using the mouse the consequent coding. If one pastes (clicking the mailbox icon with the MiddleButton) in the workshop, the formal coding is received and then translated. A new heap is pictured, figure 10.

4 The tutor

CalICo's tutor is intended for the manipulation of any combinatorial objects. Its two main purposes are to classify objects by an automatic process and to use colours on objects in order to obtain a symbolic representation of their combinatorial properties.

[6] In the following, CONTROL-LeftButton means at the same time press CONTROL and Left Button, where Left Button (respectively Middle Button) means to press on left (respectively middle) button of the mouse.

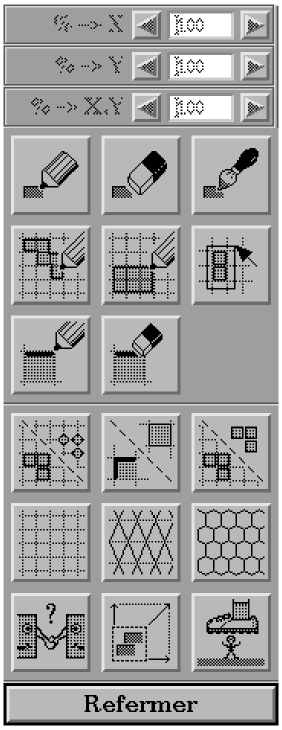


Figure 5: Toolbox of animals workshops.

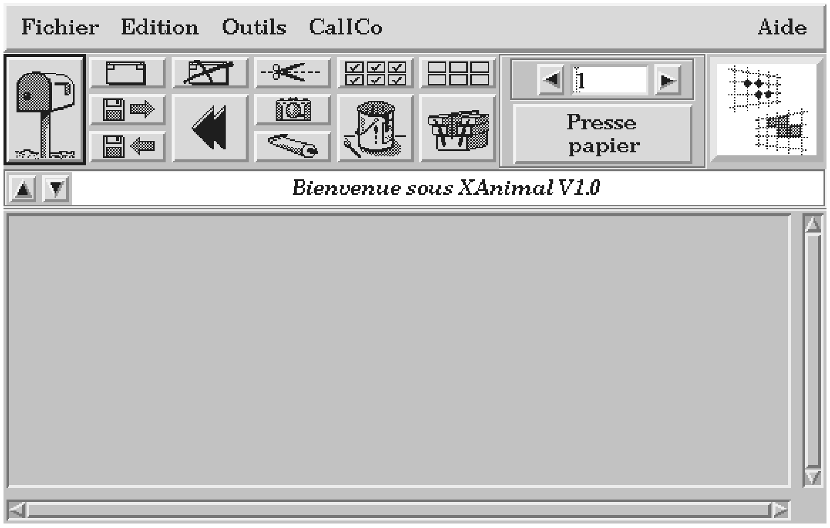


Figure 6: Animals and polyominoes visualization.

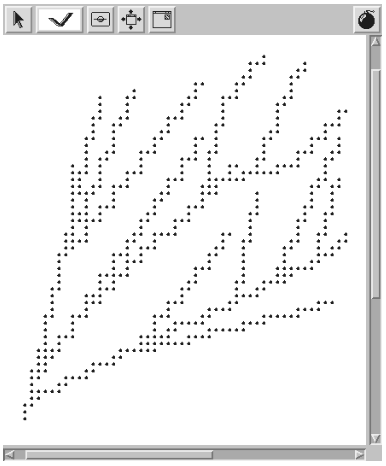


Figure 7: A random directed animal.

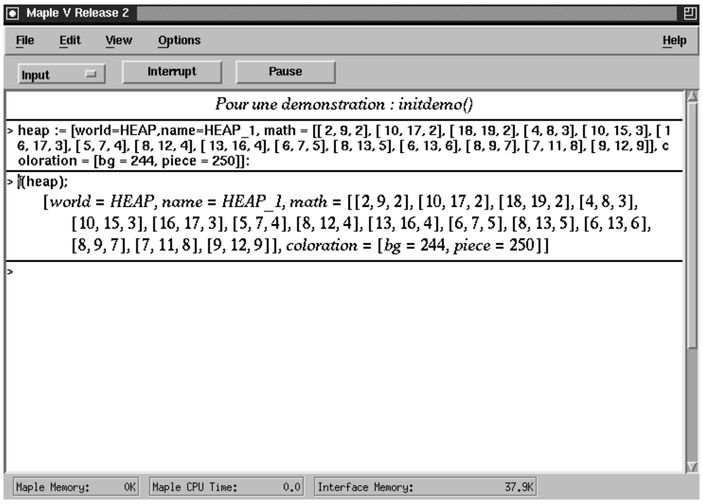


Figure 8: Xmaple session, with paste and copy

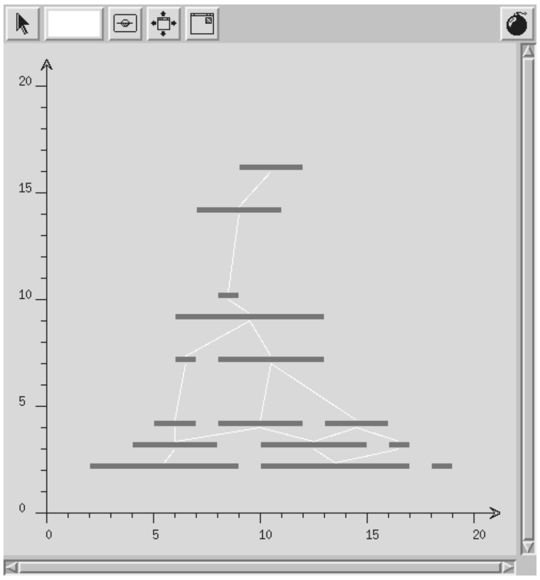


Figure 9: Initial heap, called h

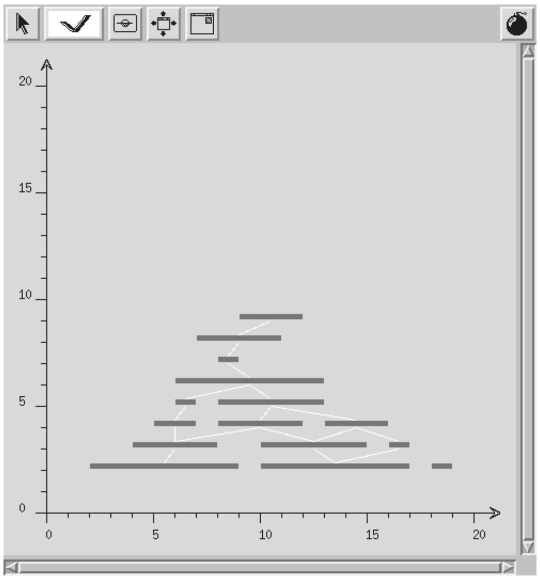


Figure 10: Pasted heap after the computation $f(h)$

First of all, we define a terminology well adapted to CalICo. We show the inheritance relationships between manipulated structures. We present the dependency graph associated with a "world" with its properties. Note that properties and definitions given in this section leads to the implementation presented in section 5.

4.1 Inheritance study

Manipulated combinatorial objects, called *elements*, are arranged in subsets, called *universe, worlds, family* and *sisterhood.*

The CalICo *universe* represents the set of all existing objects and structures in CalICo. Other universes exist, with which CalICo may exchange informations: Maple [CGG+92], Mathematica [WOL88], $\Lambda\Upsilon\Omega$[FSZ88], Gaïa [ZIM94, FZC94] and Darwin [BC88] for examples. A *world* is a sub-universe gathering together combinatorial objects that belong to the same class. CalICo's universe is composed of different worlds such as words world, polyominoes world or trees world. By analogy with statistics, we define quantitative and qualitative predicates. Let P be a predicate on an entity E. If there exists an application, denote by α from E to $\mathbb{Z}$ and an integer n such that:

$$\forall\, x \in E,\ \ P(x) \implies \alpha(x) \leq n,$$

we say that P is a *quantitative predicate*. If a predicate is not quantitative, we say that it is a *qualitative predicate*. Let E be a set of combinatorial objects of the same world and defined by the predicate P. If P is a quantitative predicate and if E is finite then E is a *sisterhood*, otherwise E is a *family*. For example, the set of trees verifying the qualitative predicate *to be a binary tree* is a family. The set of binary trees with size equal to 3 is a finite set. It is a sisterhood defined by the quantitative predicate *to have 3 vertices*. Any combinatorial object handled in CalICo is an *element*.

> *Remark.* In what follows, we will use the term *entity* when we want to refer either to a family or to a sisterhood.

We associate a *dependency graph* with each world in CalICo. Let E and E' be two entities of the world M, P and P' their respective predicates. We say that E is a *parent entity* for E' if P' implies P and for any entity E'' in M, with predicate P'', P'' implies P implies P'' implies P'. If E is a parent entity for E', we say that there is a *parent-child relation* between E and E'. Hence, we define the *dependency graph* of a world M as a directed labelled graph $G_M =< S, A, C >$, where S is the set of entities in M, A is the parent-child relations and C is the set of predicates defined in M. The transitive closure of the parent-child relation is a relation of order. We call this relation *the inheritance relation*. At each static step, for one world, we make use of a set of entities linked by the inheritance relation.

Properties of the dependency graph.
Let $G_M =< S, A, C >$ be the dependency graph of a world M, E and E' be two entities of M, with respectively predicates P and P',
i) G_M is a Directed Acyclic Graph .... *DAG prop.*
ii) if $P = P'$ then $E = E'$ .... *Clone prop.*
iii) for all path $(s_0, s_1, \ldots, s_l)$ with $l > 1$ in G_M, it does not exist an edge (s_i, s_{i+1}) in S, such that $s_0 = s_i$ and $s_l = s_{i+1}$ .... *Redundance prop.*

Defining an entity implies new inheritance relationships in the dependency graph. The following information is required: the world where the entity will be classified, the entity's name, a parent entity, an additional predicate and a set of entities that become the parents of the created entity. The three first parameters are obligatory. One of the two latter ones is sometimes sufficient. But it is not possible to ignore both at the same time (*cf. Clone prop.*).

By analogy with the synthesized and inherited attribute definitions of attribute that appear in compilers [ASU86, pages 50 and 315], we define the synthesized characteristic and the inherited characteristic. *The membership of an entity for an element* is a synthesized characteristic. This property induces that to obtain the elements of an entity E in the world M one has to collect the elements of E and the elements of all the entities, called E', such that E' is reachable from E

in G_M. *To satisfy a predicate* is an inherited characteristic. Parents hand down their predicates to their children.

4.2 Colour and combinatorics

Elements brought in CalICo's tutor have a visual representation which is an important feature in their study. The colour is used in order to visualize the combinatorial properties. Significance to the colouring can be understood by the notions of *attribute, colouring* and *interpretation* that we introduce below.

Combinatorists enumerate elements according to the value of one or more functions computed on these elements. Those functions are called *statistics* on elements. In CalICo, a *statistic* is a function from a family to $\mathbb{Z}$.

Attributes are linked to the drawing of an element. A drawing is geometrically split into indivisible structures. For example the tree drawing needs a circle for each vertex and a segment for each edge. An *attribute* is a type of indivisible structure. Occurrences of attribute are the element's parts that can be coloured. For each world, there is a finite number of attributes. For example, a tree has three attributes: vertex, edge and background. Note that for any object one can put a colour on the object's background.

Let us call weight the value of a statistic computed on an element. CalICo associates a colour with weights computed on elements. Intuitively, defining a colouring for an element consists in the following actions :

- consider the element on a geometric point of view selecting attributes
- colour the selected attributes according to the weight.

To define formally a colouring, first af all we call $\mathcal{S}$ the set of finite sequences $\mathcal{S} = \bigcup_{k \in \mathbb{N}} \mathbb{N}^k$. In a world M, we call $\mathcal{D}_M$ the set of pairs (a, s), where a is an attribute and s is in $\mathcal{S}$. Let e be in M, and let $\gamma = \{(a_i, s_i)\}_{i=1..n}$ be a set of elements in $\mathcal{D}_M$. We say that γ is *a colouring for e* iff

1. $a_i \neq a_j$ when $i \neq j$,
2. and for all $i = 1..n$, $|s_i| = n$, where n is either equal to 1 or equal to the occurrences number of a_i in e.

To colour an attribute of an element is not sufficient. Attributes are the direct illustration of the element's geometry. We need that the colouring be close to the mathematical properties of the element.
For this purpose, we introduce *interpretations*, which are the links between the geometry and the combinatorics of objects. Formally, an *interpretation* is a function from $(F \times \mathbb{N})$, where F is a family, into the set of possible colouring of an element e in F. Colours are represented symbolically by an integer. Applying

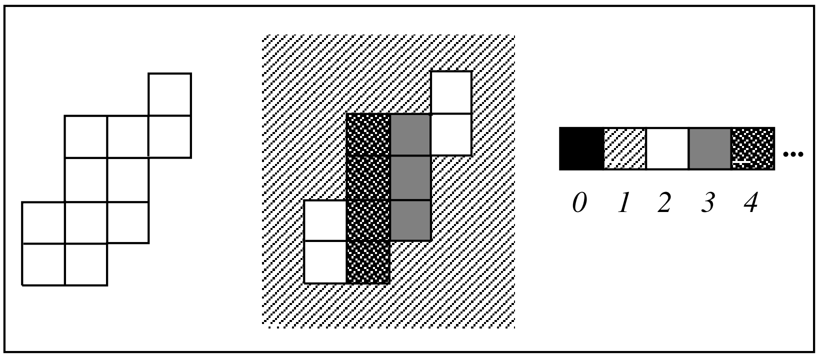


Figure 11: Example of colouring

an interpretation consists in colouring one or more attributes of e with the given "colour" (see the example what follows and figure 11).

In a world M, let $\mathcal{E}_M$ be $\mathcal{E}_M = \bigcup_{k \in I\!N}(s, \varphi)^k$, where s is either an attribute or an interpretation and φ is a statistic. Note that it is a disjoint union. Let e be an element of M, we define the application, called *colour*, from $(M \times \mathcal{E}_M)$ into the set of possible colouring for e.

Example. In figure 11, we see two occurrences of a polyominoe. The first one is without colouring. The second one has a colouring which uses two attributes: the cells and the background. The background is hachured if the polyominoe is a parallelogram polyominoe [7] and is black otherwise. The property "to be a parallelogram" for a polyominoe can be viewed as a statistic. The result, a boolean, is the weight for the polyominoe regarding this statistic. We associate two colours (hachured and black) with this weight. Until now, there is no interpretation. If we want to put the colour corresponding to the height of a column on the column, we have to use an interpretation. The statistic is the computation of the height of the column, the result of it is a weight. Setting this weight symbolically represented by a colour, on each cell of this column corresponds to the application of an interpretation on the polyominoe.

> *Remark.* Statistics and interpretations are inherited characteristics. Parents hand down their statistics and their interpretations to their children in the dependency graph.

[7] A polyominoe is a *parallelogram polyominoe* if it can be defined by two paths having elementary steps only north or east and having same first point and same last point.

5 Formal coding and symbolic computation

This section concerns the implementation of definitions and properties presented in section 4. We are dealing here with the entities handled in CalICo tutor: worlds, families, brotherhoods and elements. First we define a *unique representation* of these entities in order to handle them efficiently. This representation uses formal power series and deals with both mathematical and graphical components of the elements. We describe the fundamental operations defined on these entities: the falling operation, recognition of home domains, colouring according to statistics and interpretations. At the end of this section, we present various solutions to the problem of handling bijections between classes of objects

5.1 Unification of representations

5.1.1 Formal power series

In order to treat correctly the knowledge of different objects, we need a common description of the objects, independently of their world. Formal power series allow us to give unified definitions

Let us recall that a formal power series S is an application $S : X^* \longrightarrow K$, where X is the free monoid over the alphabet X and K a semi-ring. Counting objects using formal power series as generating functions is quite natural.

Definition 1 *The generating function associated with a set of objects E according to a parameter* size *is*

$$SG(E) = \sum_{n \geq 0} a_n \, x^n$$

where a_n is the number of elements of E having size *n.*

This definition is classicaly generalized to an arbitrary number of parameters by associating a variable with each statistic defined on the set E.

Definition 2 *Let $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ be a set of statistics defined on a set of elements E. The generating function of the set E according to the set of statistics Φ is the commutative formal power series associated with E and Φ is defined by,*

$$SF(E) = \sum_{i_1, i_2, \ldots, i_k \geq 0} a_{i_1, i_2, \ldots, i_k} \, x_1^{i_1} \, x_2^{i_2} \ldots x_k^{i_k},$$

where $a_{i_1, i_2, \ldots, i_k}$ is the number of elements e of E such that $i_j = \varphi_j(c(e))$ for j from 1 to k.

5.1.2 Uniform descriptors

We complete definition 2 by introducing another alphabet, and more precisely a coding of the objects by the mean of a formal language. We suppose that there exists a coding c sending each element onto a word. We are now dealing with two alphabets:

- The infinite commutative alphabet $X = \{x_1, x_2, x_3, \ldots\}$ is common to all the worlds of the universe and corresponds to the statitics defined in the world.
- Each world M has a proper noncommutative alphabet X_M, eventually infinite and specific to each world. It allows to code the elements of the world M. Let us denote $c(e)$ the word of ${X_M}^*$ coding of an element e.

Definition 3 *Each element e of a world M is associated with the following monomial, called descriptor, belonging to the set of the formal power series over X_M with coefficients in $\mathbb{B}.[\ x_1, x_2, \ldots, x_k\ ]$,*

$$d(e) = \left(\prod_{i=1}^{k} x_i^{\varphi_i(c(e))} \right) \ c(e).$$

The variables x_i ($1 \leq i \leq k$) are associated with the statistics φ_i defined on the world M.

Note that the descriptor of an element describes completly the element e. The noncommutative part corresponds to its coding, while the commutative part corresponds to the values of the different statistics defined on M, which is the combinatorial part of e.

Example. Let us give the example of a descriptor of a tree. Let M_a be the world of the trees where $X = \{x_1, x_2, \ldots\}$ and $X_{M_a} = \{x, \bar{x}\}$ and a an element of this world. Let us consider the statistics φ_1, φ_2, φ_3, associated respectively with the variables x_1, x_2 , x_3 such that

- $\varphi_1(a)$ is the depth of the tree a,
- $\varphi_2(a)$ is the number of leaves of a,
- $\varphi_3(a)$ is the number of internal nodes of a.

Let c be the classical coding of binary trees by Dyck words (or well parenthesed systems). For instance the descriptor the binary tree a_0 of M_a displayed figure 12 is

$$d(a_0) = x_1^3 x_2^4 x_3^3 \ xxx\bar{x}x\bar{x}\bar{x}x\bar{x}\bar{x}x\bar{x}$$

◇

The definition 3 can be easily extended to descriptors of sisterhoods, families and worlds by taking formal sums.

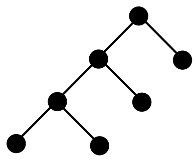


Figure 12: A binary tree a_0

Definition 4 *The* <u>*descriptor*</u> $SF(S)$ *of a set of elements* S *is defined by the formal sum*

$$SF(S) = \sum_{e \in S} d(e)$$

.

Note that such descriptors are formal power series over X_M with coefficients in $\mathbb{B}[\, x_1, x_2, \ldots, x_k \,]$. Thus the descriptors of a brotherhood C, of a family F, of a world M, are respectively defined by $SF(C) = \sum_{e \in C} d(e)$, $SF(F) = \sum_{e \in F} d(e)$, $SF(M) = \sum_{e \in M} d(e)$.

Remark. In theory, the formal power series we are dealing with can be infinite. In practice, we are only manipulating a finite number of objects so that these series are always polynomials.

5.2 Operations on the elements

We present here the most interesting operations implemented at the tutor level. We insist on the fact that one can use the tutor in order to study elements that are yet unknown at tutor level.

Remark. A first version of the tutor had been developed using Chez-Scheme [ROU91] which is an implementation of LISP. Then the tutor has been developed in Maple [CGG+92] for many reasons. Firstly it's a complete programming language and a powerfull computer algebra system, so that it is well adapted to our formal manipulations. Moreover the scientific community interested by CalICo generally knows well Maple, and a lot of such scientists have developed their own Maple packages for their research. Thus, using Maple for implementing the tutor gives us the guarantee of a true homogeneity and a very good extensibility.

5.2.1 The falling operation

The idea of this operation is to let an element go through entities using the relation parents/child. The predicates associated with each entity have the role of a filter. In order to well understand this falling principle and the other operations, we need to define the notion of *home domains* of an element.

Definition 5 *Let e be an element of a world M. An entity E (Family or sisterhood) is a* home domain *for the element e when*

- *e belongs to E and*
- *e does not belong to any descendant of E in the dependency graph of M.*

The generating family of M is the unique home domain of e when e does not belong to any other entity of the world.

Every falling element is found in its home domains and their ascendants.

Example. Consider tree 12 in the world of the trees. It is falling from the generic family of the world, successively passing through the family of the binary trees, and then through in the family of the complete binary trees and at last in the brotherhood of the trees having size 7. $\diamond$

5.2.2 Recognize

When manipulating elements, one wants to know its characteristics and properties. Some are quite obvious, others are harder to find. In particular, the knowledge of the different home domain is particularly interesting.
Recognize an object consists in applying the falling operation, that is finding all the home domains containing the element.
Recognize a set of elements consists in finding the set of the home domain which contains all the elements of the set.

5.2.3 Statistics and interpretations

When searching statistics or interpretation for an object, two cases can occur. On the one hand, one wants to know the statistics and interpretations for a known element, that is to say, recorded elements of the tutor. On the other hand, one wants to know the *statistics* and *interpretations* for new elements, that is to say non recorded elements of the tutor. In the first case, it is immediate: the information is stored in the tutor. In the other case, one has first to apply the falling principle and then compute the statistics and interpretations according to the home domains. Recall that some statistics can be defined for some families and not for other ones.

5.2.4 Colouring

The aim is to compute a colouring according to statistics or/and interpretations. First, one has to find the statistics and interpretations as explained before. Then, according to the statistic/interpretation, it is necessary to construct the coding of the object in order to send it to the corresponding visualization workshop.

5.2.5 Conception

The *conception of an element* can be done either by a visualization workshop or using a generating workshop. It consists essentialy in constructing its coding in the world and to label it by its name. Then the new elements can be recorded in the tutor.

5.2.6 Recording

Recording a set of elements consists first in recognizing the home domains of each of these elements. The recording of several elements is done sequentially. The elements are falling and are recorded successively.

5.2.7 The Universe

The main operations concerning the elements have been defined. We can also define new worlds, new families, and new sisterhoods (cf. [ROU94b]). One of the aims of the tutor is also to check and complete the validity of such new definitions.

5.3 Diffusion of knowledge

The element recorded in a world has been presented up till now from a descriptive point of view. Yet, the recording of an element has implications for the rest of the CalICo universe. Indeed, beyond the data saving, it classifies them with respect to the properties which they verify. As a consequence, the recording of elements enriches other knowledge areas in the tutor.

Here we illustrate this phenomenon by two examples (see [ROU94b] for more details).
1- Saving an element e in a family F enriches the family$F$. So, if the statistic φ applies to elements in F then $\varphi(c(e))$ will be computed and used in the following automatically in all requests. We have thus enriched the knowledge on e, on φ and on F.
2- Let F_1 and F_2 be two distinct families of a world M, such that F_1 (respectively F_2) does not belong to the ascendants of F_2 (respectively of F_1). These hypotheses imply that it does not exist in the dependency graph G_M of M a path connecting F_1 and F_2. Then, consider φ_1 (respectively φ_2) the unique statistic define on F_1 (respectively F_2) which is associated with the formal variable x_1 (respectively x_2). Let us denote by S_1 (respectively, S_2) their associated descriptors.
Suppose that one save the element e in the world M and that its home domain are F_1 and F_2. Then by definition

$$d(e) = x_1^{\varphi_1(c(e))} x_2^{\varphi_2(c(e))} c(e).$$

We can deduce that the formal power series associated with F_1 and F_2 are modified as follows

$$d(F_1) = S_1 + x_1^{\varphi_1(c(e))} x_2^{\varphi_2(c(e))} c(e)$$

$$d(F_2) = S_2 + x_1^{\varphi_1(c(e))} x_2^{\varphi_2(c(e))} c(e).$$

There is a distribution of the knowledge inside a world. Indeed, the family F_1 has enriched with the value of the statistic φ_2 perfectly valid on one of its elements while this statistic has never been declared for this family.

Remark. In the next version of CalICo's tutor, we envisage in such a case to introduce the new family satisfying the predicate P, where P is the conjunction of the predicates F_1 and F_2.

5.4 Bijections

We have said that the colour in CalICo is used for suggesting the bijections between subsets of worlds. To discover a bijection from a set E_d to a set E_a corresponds to finding a function of translation from the coding in the set E_d to the coding in the set E_a.

Property 6 *Bijective relations between worlds can be represented by deterministic finite state automata in the CalICo universe. The transitions of the automaton are the bijections and the states are sisterhoods. We call it world automaton.*

Example. The figure 13 shows a world automaton. The worlds are pictured by a very simple element. Note that one can not always pass from an element to another by transitivity because the bijections are defined for particular classes of elements in each world. It is clear nevertheless that the complete automaton for all the bijections induced a transitive relation. $\diamond$

With a view of suggesting bijections automatically, three approaches have been proposed in [ROU94b]:

- to compare sisterhood descriptions among themselves,
- to vizualise the modification after each modification of an element,
- to use objects grammar [DF94].

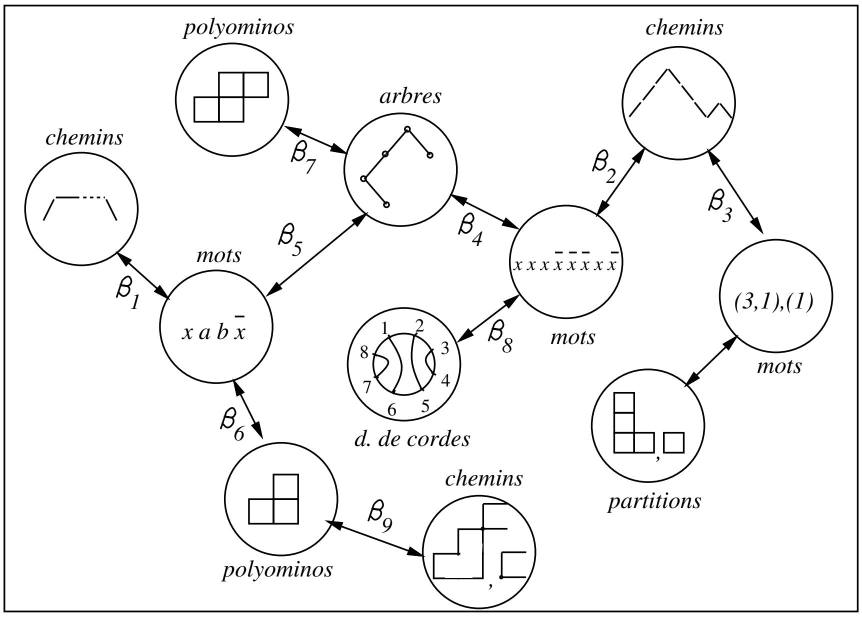


Figure 13: Example of a french world automaton

6 Graphical workshops and graphical interfaces

In this section we will describe the CalICo's graphical interfaces. First we will present the piloting interface, called *xgeci*, which is launched when the `calico` command is issued. Then we will describe the visualization workshops which are avalaible by now in the CalICo environment. We use the terminology of *graphical workshops* for those applications which have non trivial functionalities. *Graphical interfaces* will denote pure graphical applications which only interface another application. For instance xgeci is a graphical interface because it is only a nice graphical interface for the communications manager GeCI in local mode.

6.1 The Piloting Interface

In order to conduct a working session using several workshops, CalICo uses a main graphical interface, called xgeci (see figures 2 and 3 page 7). It allows to launch different workshops and to set communications between them. Thus the user can construct a *virtual machine* [GBD+93]. Each application can be run on different machines and visualized on different displays.
The interface xgeci allows to launch visualization workshops, symbolic computation workshops or any other programs which have been declared in the configuration file ".CalICo/.CalICo.config". This file is analyzed using the GNU parsers Flex and Bison. The possible applications are shown on the left side of the window of xgeci. Clicking an application produces a new dialog box (see figure 15). You can specify there on the one hand the machine on which you want a run of the application and on the other hand the display where you want to visualize it.
The different machines and displays proposed by CalICo are those which have been declared in the configuration file. Notice that adding a few lines in this file allows any application to be activated by CalICo. A part of a CalICo config file is shown in figure 14.

The main graphical interface *xgeci* allows one also to set the communication channels between all these workshops. The communication tools are shown on the right side of *xgeci* window. Once you have ran applications, you can specify how these applications will communicate by clicking on the single/double/destroy icon and then clicking on the source/target application (see figure 3 page 8). Notice that you can add or remove applications or communication channels at every moment of your CalICo session.

6.2 Visualization and manipulation workshops.

CalICo offers independent visualization workshops for different kinds of combinatorial objects. Combinatorial objects and their visualization workshops are shown in figure 6.2. Visualization workshops have been developed thanks to

```
-------------------------------------------------------------------

APPLICATIONS {
        Nom Interne : "Xpermut"
        Communication : "OUI"
        Fonction : "IGLance_appli"
        Automatique : "NON"
        Machine : "claudia"
        Display : "audrey"
        Arguments : "-pipes","__ID_PIPES__" }

GROUPE MACHINES HOTES {
        "Combinatoire" : "claudia","nastasia"
        "Calculatoire" : "hector","labri"}

GROUPE MACHINES DISPLAY {
       "Visualisation" : "mylene","sharon","audrey"}

CHEMINS {
        Machines : "labri" {
            "Xpermut" : "/home/labri/calico/Ateliers/xpermut"}

        Machines : "claudia","nastasia","hector" {
            "Xpermut" : "XPermut" } }
-------------------------------------------------------------------
```

Figure 14: Part of a french CalICo config file

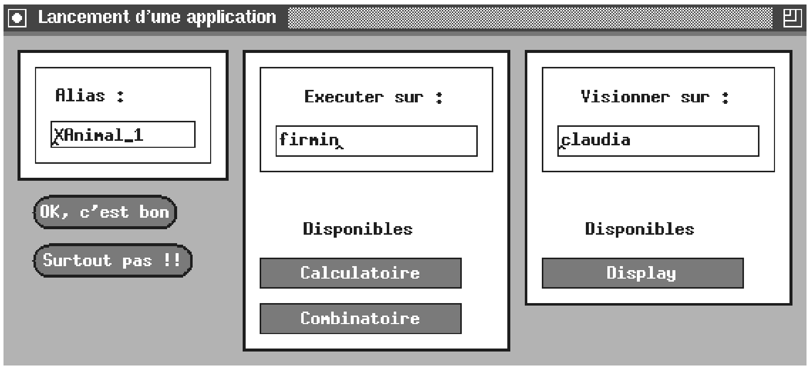


Figure 15: Launching applications from CalICo

students'projects. Since 1990, one or more groups of three or four students develop each year new visualization workshops or update old ones. All these workshops have the same philosophy and are somewhat alike even if they are dealing with different kind of objects. For instance they propose systematically a colour map toolbox and a communication box. We will describe them below.

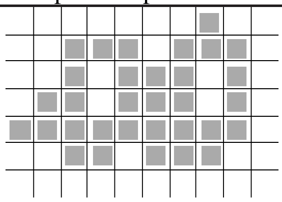
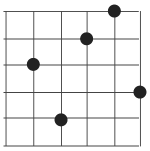
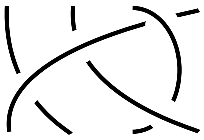
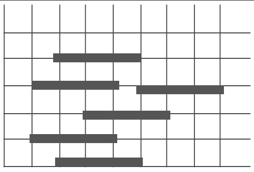
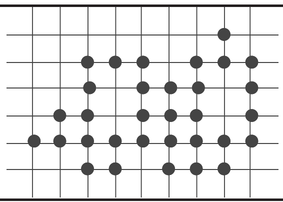
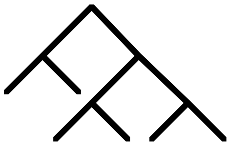
6.2.1 Generic Interface

Since 1993, we have unified the implementations of new workshops using an oriented-object approach. Thus, we developed a model for the visualization workshop, called *xmodel* [8]. We are dealing now with two kinds of workshops, early ones and xmodel-based ones. By now, there are visualization workshops for polyominoes and animals (*xanimal*), braids (*xtresses*), permutations (*xpermut*), 2D heap of pieces (*xempil*), graphs (*xgraph*), paths (*xchemin*) and trees (*xarbre*). They all are xmodel-based interfaces except *xtresses*).
Actions such as saving/recovering objects, copying/pasting/cutting an object, etc ... are independent from the nature of objects. These actions are accessible from the main window which is common to all visualization workshops (see figure 6 page 11). The common window also allows one to deal with the colourmaps of the visualization workshops. Moreover, the connection tools (send and receive) are also handled by xmodel. Concerning the menus, xmodel uses a language file which contains the name used in the different menus of the visualization workshops. For the moment these menus are in French, they can be translated using this language file. The choice of the french language was done because a large part of the communauty who is dealing with enumerative combinatorics understand french. So for us it was more convenient. A good consequence could be now to get money for french software speaking french ...

• **Main interface menus**
The different menus of the main window of the visualization interfaces (figure 6) allows basic operations on the combinatorial objects. In the menu *Fichier* one can found the functions *Nouveau* for creating a new drawing area, *Charger, Enregistrer* for loading/saving objects, *Imprimer* for printing. In the menu *Edition*, are the copy/past functions. The Painting Icon allows to open the colourmap and its tools.

• **Colormaps and colormap tools** (see figure 16). One important feature of CalICo is the use of colours to highlight properties of combinatorial objects. Every visualization workshop comes with a colourmap and colourmap tools. The colourmap is a set of 256 small coloured squares, each colour corresponds to an integer from 0 to 255. Using the colourmap tools allows one to transform the default colourmap, save or load a colourmap. The window of the colourmap tools is shown figure 16. For instance, one can swap two colours of the colourmap execute a circular permutation between several colours, generate a range of hues, create a brand new colour using RVB.

[8] xmodel has been developed using C++ and the Motif library.

Combinatorial objects	Graphical representation	Visualization workshops
Polyominoes		xpolyo (V 3.0) 1990-1992 C, Glib, Graffiti
Permutations		xpermut (V 1.0) 1992 C,AthenaWidget xpermut (V 2.0) 1994 C++, Motif, xmodel
Braids		xtressesimple (βversion) 1991 C, Glib, Grafitti xtresses (V 1.0) 1993 C++, AthenaWidget
Heaps of pieces		xempil (V 2.0) 1995 C++, Motif, xmodel
Animals		xanimal (V 1.0) 1995 C++, Motif, xmodel
Trees		xarbres (V 1.0) 1995 C++, Motif, xmodel Knvas widget
Paths		xchemins (V 1.0) 1995 C++, Motif, xmodel

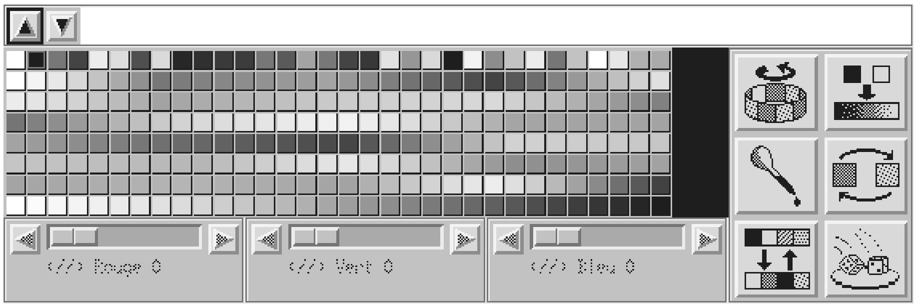


Figure 16: A colormap and its colormap tools

• **Toolbox.** Each specific visualization workshop has its own toolbox. It allows to draw objects withthe "mouse" and to modify them (symmetries and their planar transformations for instance). As an example, the animal/polyominoes toolbox is shown on figure 5.
Geometric constructions or transformations and algebraic operations are performed on the object(s) using buttons of the toolbox. Each object lives in its own window; these are then itemized in the workshop's main window for special purposes (such as selection).
• **Mailbox** A mailbox is included in each visualization workshop. When invoking the mailbox, selected objects are sent through a channel that was previously established. More precisely, the workshop sends the formal coding of an object in the communication channel defined at *xgeci* level. When a visualization workshop receives a packet, a message is printed in the message window. Clicking the mail box makes the object appear as defined in the preference file (the new object can appear in a new set of objects or in the current set of objects).

6.3 Random generation workshops.

Random generation methods have several applications particularly in providing samples of data for evaluation and tests of software and algorithms. Coupling random generation workshops with visualization tools increases considerably the interest of such tools. At the moment, CalICo environment includes two random generation workshops. The first one concerns directed animals. Another one deals with several types of plane paths, each one based on a specific algorithm (see for instance [EBS92, DEN93, WIL77]).

7 Communication manager

The aim of this section is the explanation of the construction of a real time distributed system in our context. This system allows the user, in a distributed architecture, to work jointly with independent applications and to control in interaction with the system communications between these applications. Communications between these workshops are punctual and non massive, and reply to an explicit request from the user. For convenience, we have realized a graphic interface, called piloting interface presented in section 6, allowing non-specialist in systems and communications to create, through the system, a space of work and a virtual machine on which one is going to work.

Real and cooperative distributed systems are very complex to realize, problems of order and coherence being difficult to manage. In our system, we have unloaded this management of order and coherence on the user. In fact, the information is send from a workshop to another on a user's order. In [Ba92] one underlines that an important subject of research consists in simplifying implementation and comprehension of distributed systems. In this view, we propose a solution for a particular case. The interactive communication manager, GeCI, is the module built to manage systems of communicated workshops running on remote computer and connected to the Ethernet network. GeCI offers the following properties: the capability to add easily a new workshop, the modification of connections between workshops during a session, and especially a roundup of several specialized tools in a single software environment. It seems essential to regroup specialized and independent applications in a single software environment. One finds this same approach in many recent projects, such as : Meta2-Ψlab [GG94, mG94], CAS/PI [KAJ93], ACELA project [MC95], POSSO [ROU94a]. A technical advantage of such an approach is that the tools can be developed and maintained independently.

7.1 Design

The interactive communication manager, GeCI, is a partially centralized module that manages the different workshops and creates the cohesion among them. The purpose is to combine applications to increase their potentials by distributing them on machines. Thus, a best power of calculation and a better adequacy of the machine to each of the workshops are obtained.

> *Remark.* We can see in the utilization of UNIX scripts an example of this method. The juxtaposition of small and simple programs allows one to realize sophisticated work.

Our system is before all an extensible open system, in which it is essential not to rob workshops of a part of their functionalities. Three properties are important: the architecture of the system is distributed, each workshop is able to lead to a result by being totally isolated from others workshops (it is therefore

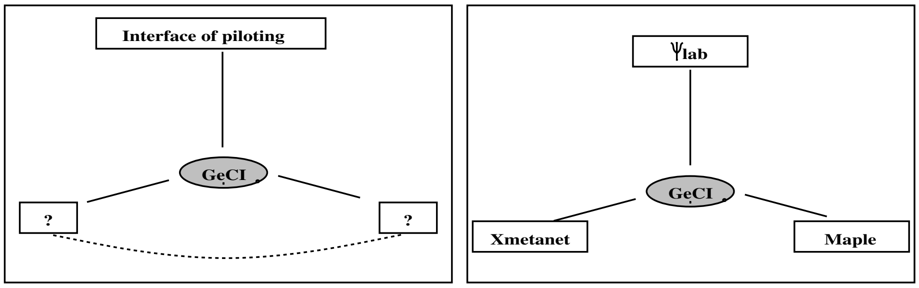


Figure 17: Diagrams of the two systems integrating GeCI

independent from others), finally workshops have the possibility to exchange information.
Connections established between workshops are not static. They are managed dynamicaly on an order of the user (or users). Users also decide the transit of information. The strategy of execution of the work depends therefore exclusively on them. GeCI establishes a two-way connection between applications [9]. It uses the usual primitives of the UNIX exploitation system and the TCP/IP protocol [RIF91]. By undertaken technical choice and the installation of a communication protocol, our communications are reliable and guarantee the arrival of messages in single copy. The communication is by default in asynchronous mode (it is the mode retained in CalICo). A request undertaken by a workshop does not imply the freezing of this workshop in waiting for the reply. The workshop can continue to work and it will be interested in the reply to its request only when the request will appear on its communication port. Communication in synchronous mode is possible, it is based on the "rendez-vous" principle. To launch an application on a distant machine, one uses a daemon [RIF91, p. 215-216], called `calicod`. This daemon is private to CalICo and dedicated to one task. The client, GeCI, formulates a specific request to the server (`calicod`). Then it executes the requested service: launching the wanted application on the remote machine designated. The utilization of usual tools TCP/IP and socket mechanisms guarantee the utilization on all sites.

7.2 Design for reusability

An important goal of the design activity is to allow for the result to evolve and to be reused in other products. Thus, this communication manager is usable in others projects, see [GGR94]. We give in figure 17 the diagrams of the two systems integrating GeCI: CalICo and Ψlab [mG94] [10].

[9] Utilization of pipes and sockets installed on the INTERNET domain.
[10] Calls are respectively: `$geci -local XGeci` and `$geci -local Scilab`

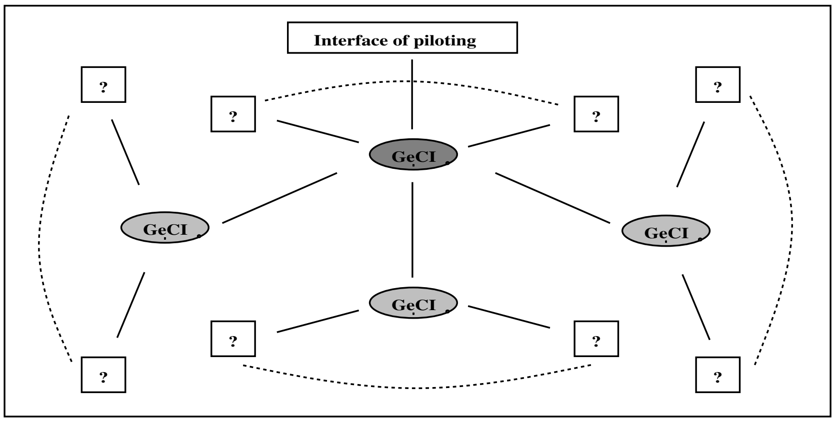


Figure 18: Set of independent and distributed workshops on connected computers network

7.3 System of Independent Workshops

To anticipate a distributed architecture particularly flexible and based on simple concepts, it is to bet on the future. Indead, a maximum of computers will be able to become possible components for the virtual machine of our system.
It is a partially centralized system, possessing a structure *in star* (see ϒfigure 18).
Each computer that is a component of the virtual machine has a central module, GeCI, that manages local communications. Each of these modules is connected to a central module, called local GeCI. Local GeCI is in direct collaboration with the user and manages the totality of the system. To launch a workshop on a remote computer, it is useless to know the port number of the computer. One uses the daemon `calicod`. The `calicod` daemon behaves as the program **`rexd`** (the SUN RPC server), with the difference that it is less powerful than **`rexd`** because specific to CalICo but thereby less "dangerous" for the security of a site.

For all X application of which one can modify the code in order to to make a workshop, one can use a high level library of functions C that we have conceived, called *CalCom*. This library facilitates the integration of an application in the system and provides the primitive of communication following our own protocol. No management on sockets has to be done by this application that now to become a workshop.

7.4 Communications and Protocol

Exchanged messages are in ASCII format. One loses a bit in time of transmission during the transit of message but one does not find problems in the framework of heterogeneous environments [BLK92]. As we said before, we have a protocol specific to our environment that follows a unique principle for all integrated applications (invisible if one uses functions of the library *CalCom*).

> *Remark.* In the framework of CalICo, combinatorial elements which are in transit in the system have a perfectly determined coding. Each application understands this coding and knows how to process it.

Each message is structured in different fields: beginning of message, identifier of sender, identifier of receiver, type of the message, data, end of message. If the workshop does not know which process it is speaking to which is most often the case) it sends its message to GeCI that knows the address of the message thanks to connections defined by the user.

The underlying protocol is TCP that provides a local order in point to point that is sufficient in our case. A limitation of our system resides in the fact that it does not management in a sophisticated manner the fault tolerance. Nevertheless, we manage to recover some failures. If one of workshops disappears in a normal manner (application leaving normally) or abnormal manner (bad functioning or stop of a machine for example) the system automatically ajusted. The virtual machine is updated and the graphic piloting interface suppresses workshop concerned in the central part of its window.

7.5 Communications by copy/paste

Another library of functions written in C that we have realized, called *libXSel*, allows the transit of data by copy/paste with the mouse between graphic applications. The utilization of functionalities of this library has been presented in section 3. This particularly convivial mechanism does not however allow validity control on data and does not allow to use screens managed by different X servers. One uses here mechanisms of CUT-BUFFERS to forward information.

7.6 Integration of an interpreter in the system

The encapsulation is obligatory for some tools if sources are not available. We have been interested especially in the problem of Maple. Nevertheless, we propose a generic interface, *GeCA*, that is usable for all types of interpreter program (Maple, Scheme, etc ...). To be able to communicate with the interpreter while it is transparent for the user, that is to say executing in background and whose standard in and standard out entries are diverted, is not sufficient. We want to work in interaction with this interpreter, as it is usual to do it, but by having, in more, the capability to make it communicate. We want to make a workshop

fully fledged with which the user will be able to work directly. The specificity of this interface is to manage *two entries* - the keyboard and the way of communications with the external world - and *two exits* - the screen and the way of communications with the external world -.
This interface of communications between the interpreter and "the external world" allows the interpreter to work without loss of its usual qualities and of its functionalities. By more, it offers specific communication functions. For a precise description of its implementation see [ROU94b].

8 Availability.

CalICo can be obtained freely via a WWW Interface :
http://www.labri.u-bordeaux.fr. See FTP de u-bordeaux.fr in the directory Local/Info/Software/CalICo
or via the anonymous ftp :
ftp.u-bordeaux.fr in the directory Pub/Local/Info/Software/CalICo An experimental version can be obtained by mail at calico@labri.u-bordeaux.fr.

8.1 Equipment

CalICo runs on workstation Sparc, under UNIX system, with an environment X-Window (X11 R4 or more). It is not indispensable, but it is strongly counseled to work on a color screen.

GeCI, the interactive communication administrator GeCI in its version 1.1 has been carried in collaboration with the members of the project META2 of INRIA-Rocquencourt on the following equipments : Sparc SUN4, HP 9000, IBM RS 6000, DEC ALPHA, DEC MIPS, PC with LINUX.

Currently the binaries that compose the CalICo environment are:

- `calicod`: this daemon process receives from the client run orders for remote applications .
- `geci`: the interactive communications administrator holds two modes,

  - local mode: `geci` is runing on the machine of the user. Joined with the piloting workshop `xgeci`, it executes orders of the user regarding launching remote workshops and interconnecting them.

  - remote mode: the set of all the machines defined by the user constitute a virtual machine. On each of these machines turns an occurrence of the process `geci`.
- `xgeci`: the graphic piloting interface of `geci` allows a user to launch and suppress interactively the workshops. It allows to create and to destroy connections between them. The interface `xgeci` transmits to `geci` all the operations to do.

Others interfaces exist [11]:

`xtuteur`: the tutor graphic interface,

`simulcra`: a communication debugger for CalICo,

`geca`: communication interface for interpretors,

`xpolyo`: for the polyominos,

`xtresse`: for braids,

`xpermut`: for permutations,

`xempil2D`: for 2D heap of pieces,

Some workshops for the random generation of combinatorial objects are available.
The installation needs 8 Mo disk space.
The tutor needs the computer algebra software `Maple` (version ++V.2 or more).

8.2 Evaluation of process

To the launching of CalICo, two process are launched: `geci` and `xgeci`.
When a graphic workshop is launched on the local machine, only the process attached to its execution is launched.
When the workshop `Maple` is launched, two process run: `Maple` and `geca`.
Finally to each computer used within a session a process `geci` runs on this machine.
The tutor in CalICo is automaticaly launched thanks to fieldAUTOMATIC in the file of configuration of `xgeci`).

8.3 Central Memory

It is not necessary to have an important central memory or a large zone of swap for an ordinary utilisation. We give, figure 19, the memory used for a session of five workshops,

Remark. One of the qualities of this environment is to minimize the usage of system resources without suppressing the suppleness of utilization.

8.4 Development

We present in this part, graphic libraries, tools and languages used in the development of the different applications of our environment.
Graphic Libraries : switchboard libraries of X-Window, the Intrinsics of the widgets, Athena Widget Set, widgets Motif.

[11] New workshops come frequently in the CalICo environment.

Memory in Ko	application
68	`geci`
576	`xgeci`
548	`xpolyo`
452	`xtresse`
616	`xpermut`
76	`geca`
344	`mapleV`
580	`xg-polyo`
3260	CalICo environment

Figure 19: Allocated memory for a CalICo session

Tools of development : Flex and Bison, lexical and syntax analysor of GNU, for the analysis of the file of configuration of `xgeci` own by each user, Maple for the tutor, WCL, Widget Creation Library, top of the public area to the Athena and Motif widgets, script shell for the installation of CalICo and the execution of each workshop.
Languages : C, C++, used compiler `gcc` of GNU, Maple used for the development of the tutor as language of programming.

9 Conclusion

As a conclusion let us do a brief abstract of what we have presented in this paper: CalICo, which means "Calcul et Image en Combinatoire" (that is *Computation and Image in Combinatorics*). We have shown that CalICo is a software system intended for combinatorial researchers. Thanks to GeCI, in a distributed environment the user handles combinatorial objects, both in a graphical and mathematical way. Having a consistent environment that integrates graphical programs, symbolic computation programs and a knowledge-based system, CalICo is a powerful tool which helps users to gain better insight into combinatorial problems. Graphical programs offer the possibility to draw objects and symbolic computation programs allow to work on their associated formal coding. A special tool, the tutor, written in Maple, classifies mathematical objects according to their combinatorial properties. In this approach, properties are automatically collected for each manipulated object. The colour is a main component in CalICo: it is the visual support that transcribes the computed statistics on objects. Of course the main remark can be that the number of workshops is small and specially oriented on the objects that we are studying frequently in Bordeaux. Thus a future works would be mainly now to implement new workshops using the Xmodel generator. This can be made by users using this last tool. Another goal will be to experiment CalICo in order to know if the tool is enough powerful for founding bijections. Another application of CalICo that we want also to explore is its use for electronic publishing.

10 Acknowledgments

CalICo project is partially supported by EC grant CHRX-CT93-0400, PRC-Maths/Info and GDR Programmation.

References

[ASU86] A. AHO, R. SETHI, and J. ULLMAN. *Compilers*. Addison-Wesley Publishing Company, 1986.

[Ba92] A. BENSOUSSAN and al. Les grands sujets de recherches en informatique et les programmes de l'inria, 1992. 25^{ieme} anniversaire de l'INRIA.

[BC88] F. BERGERON and G. CARTIER. Darwin: Computer algebra and enumerative combinatorics. *Springer lecture notes in Comp. Sc.*, 294:393–394, 1988. Proc. STACS-88.

[BLK92] M. BEAUDOIN-LAFON and A. KARSENTY. Transparency and awareness in a real-time groupware system. In *ACM Symposium on User Interface Software and Technology UIST'92*, Monterey, CA, Novembre 1992.

[CAL93] EQUIPE CALICO. Outil de développement pour les communications, manuel d'utilisation. Rapport technique, LaBRI, 1993.

[CGG+92] B. CHAR, K. GEDDES, G. GONNET, B. LEONG, M. MONOGAN, and S. WATT. *First Leaves: A Tutorial Introduction to Maple V.* Waterloo Maple Publishing, Spinger–Verlag, 1992.

[DAV89] N. JANEY D. ARQUES, G. EYROLLES and X. VIENNOT. Combinatorial analysis of ramified patterns and computer imagery of trees. In *SIGGRAPH'89*, Boston, USA, August 1989. Computer Graphics.

[dBR48] G. de B. ROBINSON. On the representation of the symetric group. *Amer. J. math*, 70, 1948.

[DEL95] M. DELEST. Algebraic languages: a bridge between combinatorics and computer science. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1995. to appear.

[DEN93] A. DENISE. Génération aléatoire et uniforme de mots. In A. Barlotti, M. Delest, and R. Pinzani, editors, *Actes du $5^{ème}$ Colloque Séries Formelles et Combinatoire Algébrique,1993*, pages 153–164. Université de Florence, 1993. à paraître dans Discrete Mathematics.

[DF94] I. DUTOUR and J-M. FÉDOU. Grammaire d'objets. Technical Report 963-94, LaBRI, Université de Bordeaux I, 1994.

[DR92] A. DENISE and N. ROUILLON. Génération de structures arborescentes. In M Lucas, editor, *Actes des 5^{ime}journées GROPLAN*, pages 83–90, Nantes, France, 1992.

[EBS92] R. PINZANI E. BARCUCCI and R. SPRUGNOLI. Génération aléatoire de chemins sous-diagonaux. In C. Reutenauer P. Leroux, editor, *Actes du $4^{ème}$ Colloque Séries Formelles et Combinatoire Algébrique, publi LaCIM 11*, Université du Québec à Montréal, 1992.

[FSZ88] P. FLAJOLET, B. SALVY, and P. ZIMMERMANN. $\Lambda_\Upsilon\Omega$: an assistant algorithms analyser. In *Proc AECC'6*, Rome, Italy, Juillet 1988.

[FZC94] P. FLAJOLET, P. ZIMMERMANN, and B. VAN CUTSEM. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

[GAU95a] V. GAUDIN. *XModele 2.0 : manuel de l'utilisateur*. LaBRI-Université Bordeaux I, Janvier 1995. N° 9995-95.

[GAU95b] V. GAUDIN. *XModele 2.0 : manuel de maintenance*. LaBRI-Université Bordeaux I, Janvier 1995. N° 9994-95.

[GBD+93] A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK, and V. SUNDERAM. Pvm 3 users's guide and reference manual. Technical report, Oak Ridge National Maboratory, Tennessee, May 1993.

[GG94] C. GOMEZ and M. GOURSAT. Metanet : a system for network analysis. *DIMACS Series in Discrete Mathematics and Theorical Computer Science (American Mathematical Society)*, Computational Support for Discrete Mathematics (15):255–268, 1994.

[GGR94] C. GOMEZ, M. GOURSAT, and N. ROUILLON. GeCI + SCILab + CalICo + Metanet + $\cdots$ ou comment faire communiquer des outils existants. In *Journées du GDR Programmation*, Lille, Septembre 1994.

[KAJ93] N. KAJLER. *Environnement graphique distribué pour le calcul formel.* PhD thesis, Université de Nice-Sophia Antipolis, 2 Mars 1993.

[KER91] A. KERBER. *Algebraic Combinatorics via Fiite Group Actions.* B.I. Wissenschaftsverlag Mannheim/Wien/Zuerich, 1991.

[MC95] L. MARTEENS and A. COHEN. *Human Interaction for Symbolic Computation*, chapter The ACELA Project: Aims and Plans. Texts and Monographs in Symbolic Computation. Springer-Verlag, 1995.

[MDMR94] J.M. FÉDOU M. DELEST, G. MELANÇON, and N. ROUILLON. Faire des mathématiques avec CalICo. *Bulletin AMQ association mathématique du québec*, pages 30–37, Mai 1994.

[mG94] Ψlab Group. *Ψlab User's Guide*. INRIA, 1994.

[RIF91] J-M. RIFFLET. *La communication sous Unix*. Collection Informatique, Mc Graw-Hill, 1991.

[ROU91] N. ROUILLON. CalICo : Une première réalisation du noyau. Mémoire de DEA Informatique, Université Bordeaux I, 1991.

[ROU94a] J-P. ROUILLER. Posso project, Novembre 1994. Journées PRC Math-Info, Luminy. Communication Orale.

[ROU94b] N. ROUILLON. *Calcul et Image en Combinatoire*. PhD thesis, Université Bordeaux I, 6 Décembre 1994.

[SCH61] C. SCHENSTED. Longest increasing and decreasing subsequences. *Canadian J. Math*, 13, 1961.

[SKI90] S. SKIENA. *Inplementing discrete Mathematics : Combinatorics and graph theory with Mathematica*. Addison-Wesley, 1990.

[SYM84] SYMBOLICS INC. *Macsyma reference manual*, 3 edition, 1984. version 10.

[VIE77] X. VIENNOT. Une forme géométrique de la correspondance de robinson-schensted. In D. Foata, editor, *Combinatoire et Représentation du groupe symétrique*, volume 579, pages 29–58. Spinger Verlag, New York, 1977. Lecture Notes in Mathematics.

[VIE86] G. VIENNOT. Heaps of pieces i: Basic definitions and combinatorial lemmas. *Lectures Notes in Maths, Combinatoire Enumérative eds G. labelle, P. leroux*, 1234:210–245, 1986.

[VIE88] X. VIENNOT. La combinatoire bijective par l'exemple, 1988. Rapport Interne, LaBRI.

[VIE92] X. VIENNOT. A survey of polyominoes enumeration. In P. Leroux and C. Reutenauer, editors, *4ème Colloque Séries Formelles et Combinatoire Algébrique*, pages 399–420, Université du Québec à Montréal, 1992. Publications du LaCIM 11.

[WIL77] H. S. WILF. A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. *Advances in Mathematics*, 24:281–291, 1977.

[WOL88] S. WOLFRAM. *Mathematica, A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Company, 1988.

[ZIM94] P. ZIMMERMANN. Gaïa: a package for the random generation of combinatorial structures. *Maple Technical Newsletter*, 1(1):38–46, 1994.