Theoretical Computer Science 79 (1991) 209-226
North-Holland

Enumeration of polyominoes using Macsyma*

M. Delest

Laboratoire Bordelais de Recherche en Informatique, Unité Associée au Centre National de la Recherche Scientifique No. 726
Département d'Informatique, U.F.R. De Mathématiques et Informatique, 351 Cours de la Libération, 33405 Talence Cedex, France

Abstract

Delest, M., Enumeration of polyominoes using Macsyma, Theoretical Computer Science 79 (1991) 209-226.

This paper shows the use of a symbolic language, Macsyma, to obtain new exact or asymptotic results in combinatorics. The examples are taken among polyominoes objects. The main purpose is to show how easy it is to bring some methods into operation in order to obtain new results in enumerative combinatorics.

Résumé

Cet article a pour but de décrire l'utilisation d'un outil de calcul formel, en l'occurrence Macsyma, pour l'obtention de résultats exacts ou asymptotiques lors d'énumérations. Les objets pris en exemple sont les polyominos. Loin d'effectuer une synthèse de ces sujets, il s'agit plutôt de donner les principales méthodes, rapides à mettre en oeuvre par un combinatoriste, sur ce type de logiciel.

1. Introduction

Macsyma is a symbolic manipulation language which allows current operations such as factorization, differentiation or solving linear or algebraic systems. It was developed using LISP from 1975 to 1983 at the Computer Science Laboratory of MIT. The diffusion is made by Symbolics Inc.

Macsyma is a good tool which works well in combinatorial problems in our current approach because we deal with the Schützenberger methodology. Let Ω be a class of combinatorial objects. Suppose that they are enumerated by the integer a_n according to the value n of a parameter p and that the corresponding generating function $f(t)=\sum_{n\geq 0} a_n t^n$ is algebraic. The methodology of Schützenberger [16]

* This work was partially supported by the "PRC de Mathématiques et Informatique".

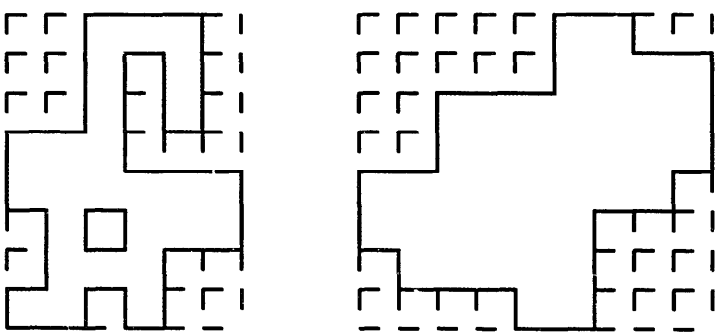
which consists of first constructing a bijection between the objects Ω and the words of an algebraic language, and then taking the "commutative image", gives an explanation for the algebricity of the generating function. The coding with words requires:

- a permanent verification of the coding during the construction of the bijections,
- the possibility of grasping big mathematical expressions.

Macsyma allows us to realize these tasks without any constraints of dimensions. Nevertheless, some concepts are not in this language. For example, it is very difficult to define new types or objects because Macsyma does not support these concepts unlike some recent languages, such as Maple [13]. Thus using Macsyma, you need to create your own standard of programmation in order to have a high compatibility between your own functions. Thus, most of our functions work on a list whose meaning is always the same, for example a functional equation having the form $G(F, x) = 0$ has the internal representation in our programs (G, F, x) in order to specify the equation, the function and the variable.

In Macsyma it is not possible to specify the domain for a variable or a function. It seems that in this field, Scratchpad [3], the software developed by IBM Research is well fitted to these problems. Moreover a lot of boolean variables must be initialized modifying the work of various functions. It is important to use them efficiently because of their big number.

We have made this paper very simple in order to allow a beginner to compute with Macsyma. Most of the examples are taken from our works on polyominoes [4–9]. The *polyominoes* are connected finite union of *cells* (unit squares) of the plane $\mathbb{Z} \times \mathbb{Z}$ (see Fig. 1).



A polyomino A convex polyomino

Fig. 1. Some polyominoes.

In Section 2 we explain our thought process when we work in Macsyma. Then, we treat successively the resolution of algebraic equations (Section 3), the research of a particular solution (Section 4), and we end with asymptotic calculus (Section 5). Appendix 1 gives an example of system which was solved using Macsyma. Appendix 2 gives a program for computing the main part of a function given by an implicit algebraic equation. We give it as an example of an elaborate program,

but we think that, today, for such a problem, the language $\Lambda_{Y}\Omega$ [12] will soon become a very good tool.

Note that since this work was completed, we have made intensive use of Macsyma and Maple and using these techniques we have found some new results in combinatorics, for example, relations between skew Ferrers diagrams and q-Bessel functions [7], relations between ternary trees and diagonal compacts animals [6].

2. Enumeration using formal calculus

In this section, we point out the main features used in Macsyma and also the main problems in using it! Generally speaking, we search for a formula which can enumerate some class of objects $\mathscr{P}$ according to some parameters.

Example 2.1. In [9], we have studied the parameter perimeter on the class of *convex* polyominoes, whose intersection with an infinite horizontal or vertical line is a connected strip of cells (see Fig. 1). Generally, we are interested in the parameters area and perimeter of polyominoes.

The Schützenberger method is based on four steps.

Step 1: Code the objects of $\mathscr{P}$ by the words of an algebraic language $\mathscr{L}$.

Step 2: Write out a non-ambiguous grammar $\mathscr{G}$ generating the language $\mathscr{L}$.

Step 3: Solve the algebraic system associated to $\mathscr{G}$ in commutative variables and obtain a generating function $\mathscr{F}$ (or a functional equation) for the language $\mathscr{L}$.

Step 4: Compute using $\mathscr{F}$ an exact formula or an asymptotic expression for the number of objects in $\mathscr{P}$ having a given value for the studied parameter.

Example 2.2. The coding language for convex polyominoes [9] is the union of three algebraic languages saying $\mathscr{L}_I$, $\mathscr{L}_{II}$ and $\mathscr{L}_{III}$ which code three types of convex polyominoes. Writing out a non-ambiguous grammar for every language is possible. The resolution of the associated algebraic systems gives the three generating functions saying $\mathscr{F}_I$, $\mathscr{F}_{II}$ and $\mathscr{F}_{III}$. Summing the expressions and expanding in Taylor series, we obtain a formula for the number of convex polyominoes having a perimeter $2n+4$. We give in Appendix 1, the grammar and the associated system for the language $\mathscr{L}_I$.

In this part of the work, we verify by means of Macsyma the intermediate and elementary results, we solve systems and sometimes, as in [7] or [8], we detect new properties and new formulas. Now we give step by step the Macsyma function we are using.

In Step 2, writing out a grammar for an algebraic language has a complexity which increases with the number of equations. It needs to show for every derivation rule a lemma of unique decomposition. These lemmas are very critical and it is a

difficult job needing a lot of tests. Often, people test it on the smaller words (according to the length) of the language. A very good mean for a systematic checking is to iterate the set of equations, in order to generate the shorter words of the language. The functions SUBST, EXPAND, PART and COEFF of Macsyma are very useful. We show in the following example the computation of Dyck words until the length $2n$ using this technique.

Example 2.3.

Lemma. *The Dyck words are the words w over $\{x, \bar{x}\}^*$ such that $w = \varepsilon$ or $w = xu\bar{x}v$ where u and v are Dyck words.*

It is easy to deduce that an equation for the Dyck language is

$$D = xD\bar{x}D + \varepsilon.$$

The computation of the first words of the Dyck language until the length $2n$ is made using the following function.

```
iterer_dyck(n):= block([s, h, d],
/* the local variable s has the value of the right part of the Dyck equation
*/
  s:1 +x.d.x̄.d,
/* H is an accumulator */
  h:s,
/* iterating loop */
  for i:1 thru n do
    h:SUBST(s,d,h),
/* suppress the non terminal words */
  h:EXPAND(SUBST(0,d,h)),
/* the result is a list formed by the formal sum of Dyck words */
/* of length ≤2n and n in order to have a self contain list*/
  [h,n]);
```

Using the output of this function, it is very easy to obtain the first coefficients of the generating function in commutative variables according to the length of words by the following function.

```
affiche(serie):= block([h,n,mot],
/* serie is supposed to be a list composed by a serie and its high order
*/
  h:SUBST(x, x̄, PART(serie,1)),
  n:PART(serie,2),
/* loop for display */
  mot:1,
  for i:0 thru n do
    (display(COEFF(h,mot,1)),
    mot:mot*x^2));
```

In this way, when an algebraic language is very difficult to handle, the formal calculus is an efficient means for the verification of the equations. It is with such a method that we obtain the equations given in Appendix 1.

In Step 3 of the methodology, Macsyma also gives some strong functions for solving for instance SOLVE, LINSOLVE, ALGSYS and RESULTANT. Nevertheless, in the case of a large system, the direct resolution using these functions is not possible for two reasons:

- the run time often becomes too expensive,
- the required memory is too large.

We will describe in Section 3 some simple techniques for handling large systems.

Now, suppose that the resolution leads to a polynomial equation in which the generating function is solution. If the degree of the polynomial is less than four or can be factorized in elementary polynomials whose degrees are less than four, Macsyma gives the solutions. One can choose a good generating function using expansion in Taylor series (function TAYLOR) and compare this with the first values obtained by the iteration process. We will develop this point in Section 4.

Note that Macsyma does not give an exact formula from a generating function $F(x)$. The function POWERSERIES is inefficient even in very simple cases.

Example 2.4. For the Dyck language, the generating function according to the length of the words is

```
d(x):(1-sqrt(1-4*x))/(2*x);
```

The evaluation of POWERSERIES(d(x),x,0) gives

$$\frac{1}{2x}-\frac{\sqrt{1-4x}}{2x}$$

and the evaluation of POWERSERIES($\sqrt{1-4x}$,x,0) is

$$\frac{2\sum_{i=0}^{\inf}\dfrac{(-4)^i x^i}{\text{beta}(\frac{3}{2}-i,\, i+1)}}{3}$$

and an exact formula easy to compute is the serie of Catalan numbers

$$d(x)=\sum_{n\geq 0}\frac{1}{n+1}\binom{2n}{n}x^n.$$

Thus, the problem of finding a simplest formula is always up to the user. In fact, making some simplifications on binomial sum is a very difficult problem in the general case; it requires a very large data basis of expressions. We do not know of any formal symbolic system which correctly deals with this problem at the present time.

It is often easy to obtain asymptotic values for the coefficients f_n of the expansion of the generating function from an implicit polynomial equation. We will discuss this problem in Section 6.

Another advantage of formal calculus takes place when the generating function is over several variables: it is easy to make a lot of variation on them and then maybe to find some new recurrences. We were often surprised by the results displayed by the program. Especially, we found:

- a distribution closed to the Narayana numbers in parallelogram study according to the perimeter [8],
- Catalan numbers in the study of directed column convex animals [5] and very recently
- q-Bessel functions in skew Ferrers diagrams, according to the number of cells [7].

Working with a symbolic language allows the combinatorists to confirm or not their conjectures. For example, Trehel has conjectured that his mutual exclusion algorithm (found with Naïmi) has a complexity equal to the nth harmonic number for a problem having an order n. Before proving it [1], we confirmed it by symbolic calculus until the order 9 (linear system with 132 equations generated by a program). The confirmation itself gave us some ideas for the proof.

In this sense, Macsyma and any other formal calculus language is like an experimental laboratory in the field of formulas and equations. But very quickly, the expansions in series from the bijection must be indexed. If not, they are forgotten if they are not useful at the time. The index could make comparisons easier, transformations by an elementary algorithm, such as Euler transforms. Something like a data basis similar to the Sloane book [17] would be of great interest.

3. Resolution of algebraic systems

In this section, we show the use of Macsyma in solving an algebraic system. We denote by $\deg_x(f)$ the degree of a polynomial f according to the variable x.

Let $\Xi = \{\xi_1, \ldots, \xi_k\}$ be a finite set of elements called *unknown*. Let $X = \{x_1, \ldots, x_p\}$ be a set of finite elements called *variables*. An algebraic system of equations over $\mathbb{K}[X]$ is a set of equations of the form

$$(E_i)\ \xi_i = p_i$$

where for i in $[1\,..\,k]$, p_i is in $K[X \cup \Xi]$. The equation E_i is linear for an unknown ξ_i if $\deg_{\xi_i}(p_i) \leqslant 1$.

The function ALGSYS of Macsyma is able to solve any system of algebraic equations. In fact, it is inefficient when it is applied to large systems. The algorithm chooses the equation in a given order but many times, the user's choice is better. Thus, it is better to proceed in the following way:

- isolate the sub-systems which are linear and solve them using the function LINSOLVE,

- use the solutions in the remaining system and eliminate the unknowns two by two using the function RESULTANT,
- in the case where only one equation remains, compute the set of solutions using the function SOLVE.

In each of these three steps, one solution is found; keep in mind that it is better to work with an unknown and only substitute its value at end.

Example 3.1. In [9], we studied convex polyominoes according to the parameter perimeter. In Appendix 1, we relate in detail the direct approach we made before handling the calculus which is written in the article. The main feature in this work is that all the generating functions depend on the Dyck enumerating function $d(x)$. The value of this function was substituted at the end of calculus.

So, very frequently, a serie σ appears all along the calculus and with successive power. Moreover, sometimes, one knows a relation of the type $\sigma^i = h_i(\sigma)$ where h_i is a rational function in which σ appears with a degree less than i. Then, the function FULLRATSIMP is very useful in order to obtain the lowest degree in the final expression. The remaining problem is that this function works in a recursive way and the run time often becomes too expensive. We prefer to use some rewriting rules using the definition function DEFRULE and the applying one APPLY1, APPLY2 and APPLYB1. In other languages it is also possible to create an array whose entries are the power of σ and the value is the function h_i and use the substitution functions.

Example 3.2. The serie $d(x)$ appears in all the calculus solving the problem of convex polyominoes. Using the definition of the Dyck equation, we have

$$d^2(x) = \frac{d(x)-1}{x^2}.$$

The following function generates the rewriting rules for the power of $d(x)$ until the order n.

```
dyck(n):= block([i],
/* The entry j of the array di will have the value of d^j */
/* rewritten in a polynomial of degree 1 in d */
  array(di,n),
  di[2]:(d-1)/x^2,
/* loop of computation for the rules */
  for i:3 thru n do
    di[i]:ratsimp(subst(di[2],d^2,ratsimp(di[i-1]*d)));
```

If a function f is an expression of some power of d, one can obtain a final expression

such that $\deg_d(f)=1$ using the following function.

```
simplifie_dyck(f):= block([i,fonc,n],
/* expansion of the function */
  fonc:expand(f),
/* compute the degree of f in the variable d */
  n:hipow(fonc,d),
/* loop for the substitution */
  for i:2 thru n do
    fonc:subst(di[i],d^i,fonc),
  ratsimp(fonc));
```

These two functions allow us to compute the generating functions $\mathcal{F}_I$, $\mathcal{F}_{II}$ and $\mathcal{F}_{III}$ in the form

$$f(x,d)=\frac{p_1(x)+d^*q_1(x)}{p_2(x)+d^*q_2(x)}.$$

When the function σ is not rational but when it is an expression with simple radicals, the function RADCAN is useful for simplification.

4. Founding a particular solution

In all this section, we will only talk about functions in one variable. The set of techniques and of problems hold in the case of several variables. We suppose that we search for an expression of a function $F(x)$ satisfying the functional polynomial equation

$$G(F,x)=0, \tag{1}$$

and also for the $n+1$ first coefficients $f_0, f_1, \ldots, f_n$ of its development in $x=0$,

$$F(x)=\sum_{i\geq 0} f_i x^i.$$

Clearly, we have the equality

$$F=G(F,x)+F.$$

From this relation, it is easy to obtain the $n+1$ first terms of the development of $F(x)$ using the following function.

```
iterer_equation(fonction,n):= block([poly,serie,variable,resultat,i,j],
/* fonction is a list whose 1st term is the equation for example G(F,x)+F
*/
  poly:part(fonction,1),
/* 2nd term is the name of the serie for example F */
  serie:part(fonction,2),
/* 3th term is the name of the variable for example x */
  variable:part(fonction,3),
```

```
/* n is the degree of the final development */
/* resultat and nouv are accumulators */
  resultat:poly,
/* iteration loop */
  for i:1 thru n do
    (resultat:ratsimp(subst(poly,serie,resultat)),
    nouv:0,
    for j:1 thru n do
      nouv:nouv+ratcoeff(resultat,x,j)*x^j,
    resultat:nouv),
  resultat);
```

Suppose that we are able to solve the equation (1) using the SOLVE function, then Macsyma gives as many solutions as the degree of G in F. The problem is what is the searched function F? We just use the function TAYLOR on every solution and compare the result to iterer_equation([G+F,F,x],n). When the solution is too complicated, the function TAYLOR does not work directly. Then one must split up the expressions into elementary ones. Let t (resp. t_1, t_2) be the polynomials given by the $n+1$ first terms of the development of the function h (resp. h_1, h_2). We apply the following properties in order to compute step by step.

- If $h = h_1 + h_2$ then $t = t_1 + t_2$.
- If $h = h_1 h_2$ then t is constituted by the $n+1$ first terms of the product $t_1 t_2$.
- If $h = n_1 \circ h_2$ then there exists $k \leq n$ such that the k first terms of $t_1 \circ t_2$ are the k first terms of the development of h.

Example 4.1. In [4], we have studied the class of column convex polyominoes according to the area and the perimeter. The degree of the functional equation was 6. Using Macsyma, we compute the six solutions and two are easy to eliminate with a direct utilisation of the TAYLOR function. The four others are solutions of an equation of degree 4. Thus in these functions appear the same subexpressions. We have written a program which computes the development of these subexpressions and then replacing in each solution, we were able to select.

A very difficult problem can occur when the solutions are functions of the nth root of a unit. In this case, Macsyma selects a canonical solution and frequently it is not possible to find the good solution. Two methods can be employed. First, add some parameters in the bijection, thus the function is more complicated and more general and may be (if you are lucky!) the nth root does not step in the final solution. The second solution is awkward and consists of introducing at some strategic points the formal expression r^n where r represents the nth root of unit. We never get something with the last one!

Example 4.2. In [5], we were interested in the generating function $P(x)$ of the number p_n of directed column-convex polyominoes having a site perimeter n (i.e.

n is the number of cells lying along the border)

$$P(x) = \sum_{n \geq 0} p_n x^n.$$

We give in the paper the functional equations which lead us to use Lagrange inversion formula. In fact, a more general equation is known of the form

$$E(z, y, x, S) = 0 \tag{2}$$

where S is a general serie enumerating the dcc-polyominoes. The function P is given by

$$E(1, x, x, P) = 0. \tag{3}$$

The evaluation of SOLVE(E(1,x,x,P),P) gives three solutions but expanding by TAYLOR does not allow to select P. All the coefficients are in $\mathbb{C}$. Let $S(x, y)$ be the function defined by (2). This equation gives also three solutions but only one fit to the correct values for S. This function is defined by

$$a(x, y) = \frac{(x-1)\sqrt{-y((4x^2-4x)y^2+(-8x^2-20x+1)y+4x^2-8x+4)}}{6\sqrt{3}},$$

$$b(x, y) = (2x^3-6x^2-6x-2)y^3+(-6x^3+18x-12)y^2 + (6x^3+9x-15)y-2x^3+6x^2-6x+2/54,$$

$$c(x, y) = (x-1)^2y^2-2(x-1)(x+2)y+(x-1)^2,$$

$$d(x, y) = \sqrt[3]{a(x, y)-b(x, y)}$$

and

$$s(x, y) = d(x, y)+\frac{c(x, y)}{9d(x, y)}-\frac{(y+2)(x-1)}{3}.$$

Finally we get $P(x) = S(x, \cdot)$ and thus the expression of the generating function $P(x)$.

The main interest in founding the explicit expression of the generating function is to obtain the formal development of $P(x)$ but also to compute easily the singularity of the function in order to get asymptotic approximation for the coefficients of the serie. We show in the next section one useful method which just needs the functional equation for computing this approximation.

5. Asymptotic calculus of the coefficients

In this section, we explain how to use Macsyma in order to compute the asymptotic value for the number f_n of objects of a given class having the value n for a parameter; this approximation is obtained in the form

$$f_n \approx K\mu^n n^{-\theta} \tag{4}$$

from the analytic form of the function $F(x)$ or from the functional equation $G(F, x) = 0$. In fact, we show in the following that it is possible to obtain with

Macsyma the main part of the function $F(x)$ in the form

$$F(x) \approx \lambda (x - x_0)^{\alpha}$$

with $\alpha \in \mathbb{Q}$, $\lambda \in \mathbb{C}$ and $x_0 \in \mathbb{R}$. In order to obtain such expressions, several methods are available. We just show the most frequently used. Details and proofs can be found in [2, 10, 11, 18].

If the function $F(x)$ has been computed in the previous step, we can try to compute the singularities. Even in this case, it is not sometimes possible to compute them. For example in [4], the function is known but it seems very difficult to compute its singularities. We need to find the smallest one according to the module. Let x_0 be this singularity. Then $F(x)$ has the decomposition

$$F(x) = H(x)S(x) + C(x)$$

where x_0 is not a singularity of $H(x)$ and $C(x)$. Around $x = x_0$, the function $H(x)$ admits the following decomposition:

$$H(x) = \sum_{i \geq 0} \eta_i (x - x_0)^i$$

thus we deduce that

$$F(x) = \eta_0 S(x) + o((x - x_0)S(x)).$$

Using the Macsyma function SOLVE, we can often compute the singularities, split $F(x)$ and then compute $\eta_0 = H(x_0)$.

Example 5.1. In [8], the study of parallelogram polyominoes according to their site perimeter leads to a calculus of the asymptotic number of such polyominoes from the generating function

$$p(x) = \frac{1 - x^2 - 2x^3 + x^4 + (x^2 - x - 1)\sqrt{1 - 2x - x^2 - 2x^3 + x^4}}{2}.$$

There is only one radical and the polynomial under the radical can be factorized by Macsyma. Then it is easy to deduce that

$$p(x) \approx \left(1 - x\frac{3 + \sqrt{5}}{2}\right)^{1/2}.$$

If it is not possible to give the singularities from the function, one can use the functional equation for computing it using Puiseux development [10]. For this, first solve the system

$$\begin{aligned} &G(F, x) = 0, \\ &\frac{\partial G(F, x)}{\partial F} = 0. \end{aligned} \tag{6}$$

In Macsyma, we make the evaluation of

```
algsys([G(F,x)=0,diff(G(F,x),F,1)=0],[F,x]);
```

among all the solutions (F_s, x_s), let (φ, ξ) be the solution such that for all s, we have $x_s \geqslant \xi$ and $F_s \geqslant \varphi$. We define the polynomial

```
Q(x,y) := G(y+φ,x+ξ).
```

Example 5.2. For the Dyck generating function, we have

$$G(F, x) = xd^2 - d + 1$$

we deduce $\varphi = 2$ and $\xi = \frac{1}{4}$.

Let y be λx^{α}. In Macsyma, it leads us to define the function

```
Q1(λ,x,α) := Q(x,λ*x^α).
```

The polynomial $Q1(\lambda, x, \alpha)$ can be written in the form $\sum_{j=1}^{n} P_j(\lambda) x^{b_j\alpha + a_j}$ with j in $[1 \,..\, n]$, $P_j(\lambda)$ a polynomial in $\mathbb{C}[\lambda]$, (a_j, b_j) in $\mathbb{R}^2$.

Example 5.3. After the substitution of variables, we obtain

$$Q1(\lambda, x, \alpha) = \lambda^2 x^{2\alpha+1} + \frac{\lambda^2}{4} x^{2\alpha} + 4\lambda x^{\alpha+1} + 4x.$$

Using the Macsyma functions which implement the list manipulation, it is easy to isolate the polynomials P_j and the pairs (a_j, b_j) in $Q1(\lambda, x, \alpha)$.

Example 5.4. We extract from $Q1(\lambda, x, \alpha)$ the following values

$$P_1 = \lambda^2, \qquad a_1 = 1, \qquad b_1 = 2,$$

$$P_2 = \lambda^2/4, \qquad a_2 = 0, \qquad b_2 = 2,$$

$$P_3 = 4\lambda, \qquad a_3 = 1, \qquad b_3 = 1,$$

$$P_4 = 4, \qquad a_1 = 1, \qquad b_4 = 0.$$

Then we determine λ and α using the Newton polygon [10]. We do not give explanations about this method but the technique leads to the selection of a set of indices. In the general case, the value of α is a rational one. From this it is easy to

deduce that

$$F(x) \approx \lambda(x - x_0)^{\alpha}.$$

Example 5.5. The two values given by the Newton polygons are $j = 1$ and $j = 4$. We deduce that $\alpha = \frac{1}{2}$ and $\lambda = 4i$ and thus

$$d(x) = 4i(x - \tfrac{1}{4})^{1/2}$$

We give in Appendix 2, a function written in Macsyma which realizes this algorithm.

Appendix 1

We give in the following a system of equations allowing the generation of the pigmented Dyck language defined in [9]. We denote by D the classical Dyck language. The name of the non-terminals are in capitals. Their name reflect their properties. So P (resp. I, O, $\underline{P}$) means an even number (resp. odd, none, non-zero even) of letters "a". The letter A in the name of a non-terminal means that it only generates letter "a". The letter R means that the words of the associated language to the non-terminal are prime Dyck words shuffled with letters "a". The non-terminal PP is associated to the pigmented Dyck language.

$$PP = R\underline{P}O.D.R.O\underline{P} + R\underline{P}O.D + D.RO\underline{P} + R\underline{P}\underline{P} + x.D.\bar{x}.D$$
$$PI = R\underline{P}O.D.R.OI + D.ROI + R\underline{P}I$$
$$IP = R\underline{I}O.D.R.O\underline{P} + R\underline{I}O.D + RI\underline{P}$$
$$II = RIO.D.ROI + RII$$
$$R\underline{P}O = x.\underline{P}OAO.\bar{x}$$
$$RO\underline{P} = x.O\underline{P}OA.\bar{x}$$
$$RIO = x.IOAO.\bar{x}$$
$$ROI = x.OIOA.\bar{x}$$
$$\underline{P}OAO = a.IOAO + x.\underline{P}OAO.\bar{x}.D$$
$$IOAO = a.\underline{P}OAO + x.IOAO.\bar{x}.D + a.x.D.\bar{x}.D$$
$$O\underline{P}OA = OIOA.a + D.x.O\underline{P}OA.\bar{x}$$
$$OIOA = O\underline{P}OA.a + D.x.OIOA.\bar{x} + D.x.D.\bar{x}.a$$
$$R\underline{P}\underline{P} = x.\underline{P}\underline{P}AA.\bar{x}$$
$$RI\underline{P} = x.I\underline{P}AA.\bar{x}$$
$$R\underline{P}I = x.\underline{P}IAA.\bar{x}$$
$$RII = x.IIAA.\bar{x}$$
$$\underline{P}\underline{P}AA = a.IIAA.a + a.I\underline{P}AO + \underline{P}IOA.a + R\underline{P}\underline{P} + R\underline{P}O.D.ROP$$
$$\underline{P}IAA = a.IPAA.a + a.IIAO + \underline{P}POA.a + R\underline{P}I + R\underline{P}O.D.ROI$$
$$I\underline{P}AA = a.PIAA.a + a.P\underline{P}AO + IIOA.a + RI\underline{P} + RIO.D.RO\underline{P}$$
$$IIAA = a.PPAA.a + a.PIAO + IPOA.a + II$$
$$\underline{P}\underline{P}AO = a.I\underline{P}AO + R\underline{P}\underline{P} + R\underline{P}O.D.RO\underline{P} + D.RO\underline{P}$$
$$I\underline{P}AO = a.P\underline{P}AO + RI\underline{P} + RIO.D.RO\underline{P}$$

$$PPOA = \underline{P}IOA.a + R\underline{P}P + R\underline{P}O.D.RO\underline{P} + R\underline{I}O.D$$
$$PIOA = \underline{P}POA.a + R\underline{P}I + R\underline{P}O.D.ROI$$
$$PPAO = a.IPAO + PP$$
$$PIAO = a.IIAO + PI$$
$$IPAO = a.PPAO + IP$$
$$IIAO = a.PIAO + II$$
$$PPOA = PIOA.a + PP$$
$$IPOA = IIOA.a + IP$$
$$PIOA = PPOA.a + PI$$
$$IIOA = IPOA.a + II$$
$$IPAA = a.PIAA.a + a.PPAO + IIOA.a + IP$$
$$PIAA = a.IPAA.a + a.IIAO + PPOA.a + PI$$
$$PPAA = a.IIAA.a + a.IPAO + PIOA.a + PP$$

Using the morphism μ from $\{a, x, \bar{x}\}^*$ into $\{x\}^*$ which transforms all the letters into the letter x, the commutative image of this system allows the computation of the generating function pp of the Dyck pigmented language. We denote by d the generating function of the Dyck language. We obtain the following system

$$pp = x^4\, d\, rp^2 + 2x^2\, d\, \underline{rp} + x^2 \underline{ppaa} + x^2\, d^2 \tag{1}$$
$$pi = x^4\, d\, rp\, ri + x^2\, d\, ri + x^2\, \underline{piaa} \tag{2}$$
$$ii = x^4\, d\, ri^2 + x^2\, iiaa \tag{3}$$
$$\underline{rp} = x\, ri + x^2\, d\, \underline{rp} \tag{4}$$
$$ri = x\, \underline{rp} + x^2\, d\, ri + x^3\, d^2 \tag{5}$$
$$\underline{ppaa} = x^2\, iiaa + 2x\, \underline{ipao} + x^2\, \underline{ppaa} + x^4\, d\, \underline{rp}^2 \tag{6}$$
$$\underline{piaa} = x^2\, piaa + x\, iiao + x\, \underline{ppao} + x^2\, \underline{piaa} + x^4\, d\, \underline{rp}\, ri \tag{7}$$
$$iiaa = x^2\, ppaa + 2x\, piao + ii \tag{8}$$
$$ppaa = x^2\, iiaa + 2x\, ipao + pp \tag{9}$$
$$piaa = x^2\, piaa + x\, ppao + x\, iiao + pi \tag{10}$$
$$\underline{ppao} = x\, \underline{ipao} + x^2\, \underline{ppaa} + x^4\, d\, \underline{rp}^2 + x^2\, \underline{rp} \tag{11}$$
$$\underline{ipao} = x\, \underline{ppao} + x^2\, \underline{piaa} + x^4\, d\, ri\, \underline{rp} \tag{12}$$
$$ppao = x\, ipao + pp \tag{13}$$
$$ipao = x\, ppao + pi \tag{14}$$
$$piao = x\, iiao + pi \tag{15}$$
$$iiao = x\, piao + ii \tag{16}$$

The computation is made in the following order:

- solving equations (15) and (16) gives piao and iiao,
- solving equations (13) and (14) gives ppao and ipao,
- solving equations (11) and (12) gives $\underline{ppao}$ and $\underline{ipao}$,
- solving equations (8), (9), (10) gives *iiaa*, *ppaa* and *piaa*,
- solving equations (6) and (7) gives $\underline{ppaa}$ and $\underline{piaa}$,
- solving equations (4) and (5) gives $\underline{rp}$ and *ri* with respect to the serie *d*,
- at last, solving equations (1), (2) and (3) gives *pp* according to the serie *d*.

Appendix 2

We only give the text of the function that realizes the computation of the main part of a function. The argument of the function is a list of the form $(G(F, x), F, x)$; the result is the main part of $F(x)$ in the form $\lambda(x-x_0)^{\alpha}$.

```
puiseux(fonction) := BLOCK([fonc,x,x0,lambd,alfa,long,liste],
  /* fonction is a list of three parameters */
  /* 1st parameter of the function G(F,x) */
  /* 2nd parameter, the name of the function F */
  /* 3rd parameter, the name of the variable x */
  /* Compute Q1 and Xi */
    x:part(fonction,3),
    fonc:puiseux_change(fonction),
    display(fonc),
    x0:part(fonc,2),
    fonc:part(fonc,1),
  /* Compute Pj and (aj, bj) */
  /* Find the pairs (0,bj) such that bj is minimum */
    long:length(fonc),
    array(a,long),
    array(b,long)
    array(p,long),
    liste:isole_puiseux([fonc,x]),
    display(liste),
  /* Compute alfa and lambda */
    fonc:final_puiseux(liste),
    display(fonc),
    kill(a,b,p),
    part(fonc,1)*(x-x0)^part(fonc,2));
puiseux_change(fonction) := BLOCK([fonc,y,x,long,solution,
                                   imin,min,i,so],
  /* The input of this function is the functional */
  /* equation and its output is a list of two */
  /* 1st element is the function Q1 */
  /* 2nd element is the main singularity */
    fonc:part(fonction,1),
    y:part(fonction,2),
    x:part(fonction,3),
  /* Solve the system (6) */
    solution:algsys([fonc = 0,diff(fonc,y,1) = 0],[x,y]),
    display(solution),
```

```
    long:length(solution),
    imin:1,
    min:part(solution,1,1,2),
    while min = 0 or imagpart(min)≠0
      or imagpart(part(solution,imin,2,2)≠0 do
    (imin:imin + 1,
    min:part(solution,imin,1,2)),
    for i:imin + 1 thru long do
      (so:part(solution,i,1,2),
      if abs(so) < abs(min)
      and so≠0
      and imagpart(so) = 0
      and imagpart(part(solution,i,2,2)) = 0
      then
        (imin:i,
        min:so,
        display(imin,min))),
    x0:part(solution,imin,1,2),
    y0:part(solution,imin,2,2),
    display(x0,y0),
  /* Change the variable by (7) */
    fonc:subst(x + x0,x,subst(y + y0,y,fonc)),
  /* Change the variable by (8) */
    fonc:ratexpand(subst(lambd*x^alfa,y,fonc)),
    [fonc,x0]);
isole_puiseux(fonc) := block([eclat,long,mono,deno,nume,j],
  /* Begin the computation of the Newton polygon */
  /* the input is a list composed with Q1 and the variable x */
    eclat:isolate(part(fonc,1),part(fonc,2)),
    long:length(eclat),
    imin:0,
    for j:1 thru long do
      (mono:part(eclat,j),
      nume:num(mono),
      deno:denom(mono),
      if atom(nume)
      then (p[j]:1/deno,
        a[j]:1,
        b[j]:0)
    else (p[j]:part(nume,1)/deno,
      if length(nume) = 3
      then p[j]:p[j]*part(nume,2),
      mono:mono/p[j],
```

```
        if atom(mono)
        then (a[j]:1,
          b[j]:0)
        else (mono:part(mono,2),
          b[j]:coeff(mono,alfa,1),
          a[j]:mono-b[j]*alfa)),
      display(a[j],b[j],p[j]),
      if a[j]=0
      then if imin=0
           then imin:j
           else if b[j]<b[imin] then imin:j),
    [long,imin]);
final_puiseux(liste):= block([long,min,indic,indic0,j,
    min0,alfa,lambd,equa,element],
  /* Final computation of alfa and lambda from */
  /* the output list of isole_puiseux */
    alfa:0,
    long:part(liste,1),
    min:part(liste,2),
    for j:1 thru long do
    (if a[j]>0 and b[j]<b[min]
    then (indic:a[j]/(b[min]-b[j]),
      display(j,indic),
      if alfa=0 or indic<alfa
      then (alfa:indic,
        equa:p[j])
      else if indic=alfa
           then equa:equa+p[j])),
    equa:equa+p[min],
    equa:solve(equa,lambd),
    display(equa),
    long:length(equa),
    lambd:0,
    min:0,
const:(-1)^alfa,
for j:1 thru long do
  (element:part(equa,j,2),
if imagpart(const*element)=0
then (min0:realpart(const*element),
  if min0>min
  then (min:min0,
    lambd:element))),
[lambd,alfa]);
```

References

[1] A. Arnold, M. Delest and S. Dulucq, Complexité moyenne de l'algorithme d'exclusion mutuelle de Naïmi-Trehel, *Journées Informatique et Mathématiques*, Marseille 1988.

[2] E. Bender, Asymptotic methods in enumeration, *SIAM Rev.* (1974) 485-515.

[3] Computer Algebra Group, Scratchpad II, *IBM Research Newsletter* 1 (1985).

[4] M. Delest, Generating function for column-convex polyominoes, *J. Combin. Theory Ser. A* **48** (1) (1988) 12-31.

[5] M. Delest and S. Dulucq, Enumeration of directed column-convex animals with given perimeter and area, Rapport LaBRI, Bordeaux, no. 87-15.

[6] M. Delest and J.M. Fedou, Exact formulas for fully compact animals, Rapport Interne LaBRI, Bordeaux, no. 89-06.

[7] M. Delest and J.M. Fedou, Enumeration of skew Ferrers diagrams, Rapport Interne LaBRI, Bordeaux, no. 89-19.

[8] M. Delest, D. Gouyou-Beauchamps and B. Vauquelin, Enumeration of parallelogram polyominoes with given bond and site perimeter, *Graphs Combin.* **3** (1987) 325-339.

[9] M. Delest and X. Viennot, Algebraic languages and polyominoes enumeration, *Theoret. Comput. Sci.* **34** (1984) 169-206.

[10] J. Dieudonne, *Calcul Infinitésimal* (Hermann, Paris, 1968).

[11] P. Flajolet, Mathematical methods in the analysis of algorithms and data structures, in: E. Börger, ed., *Trends in Theoretical Computer Science* (Computer Science Press, 1988).

[12] P. Flajolet, B. Salvy and P. Zimmermann, $\Lambda_Y\Omega$: an assistant algorithms analyser, in: *Proc. AECC'6*, Rome (1988); see also this issue, pp. 37-109.

[13] K.O. Geddes, G.H. Gonnet and B.W. Char, Maple, User's Manual, Research report CS-83-41, University of Waterloo, 1983.

[14] S. Lang, *Complex Analysis* (Addison-Wesley, Reading, MA, 1977).

[15] L.L. Lo, Asymptotic matching by the symbolic manipulator MACSYMA, *J. Comput. Phys.* **61** (1985) 38-50.

[16] M.P. Schützenberger, Certain elementary families of automata, in: *Proc. Symp. on Mathematical Theory of Automata*, Polytechnic Institute of Brooklyn (1962) 139-153.

[17] N.J. Sloane, *A Handbook of Integer Sequences* (Academic Press, New York, 1979).

[18] J.M. Steyeart, Complexité et structure des algorithmes, Thèse d'Etat, Université de Paris VII, 1984.

[19] Symbolics Inc., *Macsyma Reference Manual*, version 10, 3rd edn. (1984).