CHAPTER 9

Counting, Coding and Sampling with Words

9.0 Introduction 443
9.1 Counting: walks in sectors of the plane 445
9.1.1 Unconstrained walks and rational series 445
9.1.2 Walks on a half line and Catalan's factorization 447
9.1.3 Walks on a half plane and algebraic series 449
9.1.4 Walks on the slitplane and the cycle lemma 453
9.2 Sampling: polygons, animals and polyominoes 456
9.2.1 Generalities on sampling 457
9.2.2 Parallelogram polyominoes and the cycle lemma 458
9.2.3 Directed convex polyominoes and Catalan's factorization 460
9.2.4 Convex polyominoes and rejection sampling 461
9.2.5 Directed animals 463
9.3 Coding: trees and maps 466
9.3.1 Plane trees and generalities on coding 467
9.3.2 Conjugacy classes of trees 469
9.3.3 The closure of a plane tree 471
9.3.4 The opening of a 4-valent map 473
9.3.5 A code for planar maps 476
Problems 477
Notes 478

9.0. Introduction

This chapter illustrates the use of words to derive enumeration results and algorithms for sampling and coding.

Given a family $\mathcal{C}$ of combinatorial structures, endowed with a size such that the subset $\mathcal{C}_n$ of objects of size n is finite, we consider three problems:

- *Counting:* determine for all $n \geq 0$, the cardinal $\mathrm{Card}(\mathcal{C}_n)$ of the set $\mathcal{C}_n$ of objects with size n.
- *Sampling:* design an algorithm $\textsc{Rand}\mathcal{C}$ that, for any n, produces a random object uniformly chosen in $\mathcal{C}_n$: in other terms, the algorithm must satisfy,

for any object $O \in \mathcal{C}_n$, $\mathbf{P}(\textsc{Rand}\mathcal{C}(n) = O) = 1/\operatorname{Card}(\mathcal{C}_n)$.

– *Optimal coding:* construct a function φ that maps injectively objects of $\mathcal{C}$ on words of $\{0,1\}^*$ in such a way that an object O of size n is coded by a word $\varphi(O)$ of length roughly bounded by $\log_2 \operatorname{Card}(\mathcal{C}_n)$.

These three problems have in common an *enumerative* flavour, in the sense that they are immediately solved if a list of all objects of size n is available. However, since in general there is an exponential number of objects of size n in the families we are interested in, this solution is by no way satisfying. For a wide class of so-called *decomposable* combinatorial structures, including non ambiguous algebraic languages, algorithms with polynomial complexity can be derived from the rather systematic recursive method. Our aim is to explore classes of structures for which an even tighter link exists between counting, sampling and coding.

For a number of natural families of combinatorial structures, the counting problem has indeed a "nice" solution: by nice could be intended that there is a simple formula for $\operatorname{Card}(\mathcal{C}_n)$, that the generating series $\sum_{n\geq 0} \operatorname{Card}(\mathcal{C}_n)x^n$ is an algebraic function, etc. The rationale of this chapter is that these nice enumerative properties are the visible "traces" of deeper structural properties, and that making the latters explicit is a way to solve simultaneously and simply the three problems above.

The enumeration of walks on lattices (Section 9.1) is an inextinguishable source of nice counting formulas. These formulas can often be given simple interpretations by viewing walks as words on an alphabet of steps, and using ingredients of the combinatorics of words. In particular we shall consider some rational and algebraic languages, shuffles and the cycle lemma.

Convex or directed polyominoes (Section 9.2) illustrate the idea that nice combinatorial properties help for sampling. Since enumeration and random generation of general polyominoes appear intractable, it was proposed in statistical physics to study subclasses like convex or directed polyominoes, that display better enumerative properties. These objects can be described in terms of simple languages, often algebraic, and this leads to efficient random generators.

The family of planar maps (Section 9.3) is a further example of class with unexpectedly nice enumerative properties. Maps are the natural combinatorial abstraction for embeddings of graphs in the plane and for polygonal meshes in computational geometry, and maps were also largely studied in theoretical physics. Toy models of statistical physics, like percolation or the Ising model, are often studied on regular lattices, but also on random maps. The uniform distribution indeed appears to give, at the discrete level, the right notion of distribution of probability on possible universes as prescribed by quantum gravity. In these various contexts, results have been obtained independently on counting, sampling and coding problems. Again we rely on a combinatorial explanation of the enumerative properties of planar maps to approach these three problems.

Most of the time, we state and prove results for some particularly simple structures, while they are valid for more generic families (*e.g.* walks with more general steps, polyominoes on other lattices, maps with constraints). We made this choice to maintain the chapter relatively short, but also because on these

simple structures the "traces" are more visible, and the underlying combinatorics appears more explicitly.

All the objects that are considered in this chapter have nice geometric interpretations in the plane. We have chosen to rely on the geometric intuition of the reader to support these interpretations, and concentrate the proofs on the combinatorial aspects.

9.1. Counting: walks in sectors of the plane

A *(nearest neighbor) walk* on the square lattice $\mathbb{Z}^2$ is a finite sequence of vertices $w = (w_0, w_1, \ldots, w_n)$ in $\mathbb{Z}^2$ such that each *step* $w_i - w_{i-1}$, for $1 \leq i \leq n$, belongs to the set $\mathcal{S} = \{(0,1), (0,-1), (-1,0), (1,0)\}$. The number of steps n is the *length* of w; w_0 and w_n are respectively its startpoint and endpoint. The *reverse* walk of w is the walk $\bar{w} = (w_n, w_{n-1}, \ldots, w_1, w_0)$. A *loop* is a walk with identical startpoint and endpoint.

Elements of $\mathcal{S}$ are also denoted u, d, l, r – standing for *up, down, left* and *right*. Unless explicitly specified, we consider walks up to translation, or equivalently, we assume that they start from the origin $(0,0)$. A walk can thus be seen as a word on the alphabet $\mathcal{S} = \{u, d, l, r\}$ and we identify the set of walks with the language $\{u, d, l, r\}^*$, making no distinction between both of them.

In the rest of this section, we study families of walks with various boundary constraints: on a line, a half line, a half plane, a quarter plane, and finally, on the slitplane. This is the occasion to introduce enumerative tools that will be of use in later sections.

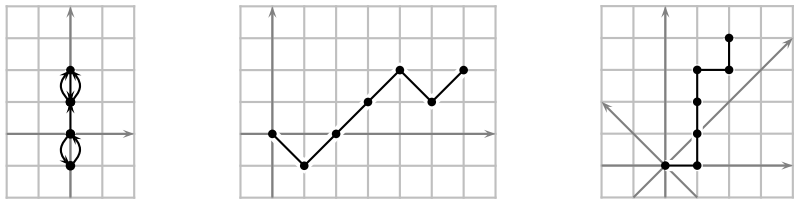
9.1.1. Unconstrained walks and rational series

Let us first consider walks that use only vertical steps (*i.e.* u or d), and hence stay on the axis ($x = 0$). These walks are sometimes called *one-dimensional simple symmetric walks*, and are often considered in their "time stretched" version: each step u or d is replaced by a $(1,1)$ or $(1,-1)$ step, in order to give an unambiguous representation in the plane, as illustrated by Figure 9.1. Up to a $\pi/4$-rotation, these walks are in one-to-one correspondence with walks with steps in $\{u, r\}$ and as such, are sometimes called *staircase walks*, or *directed two-dimensional walks*.

Counting these walks with respect to their length ℓ amounts to counting words on $\{u, d\}$ of length ℓ, and there are 2^ℓ of those. Restricting them to end at ordinate j, with $\ell = 2n + |j|$ for some nonnegative n, is hardly more difficult: for $j \geq 0$, the corresponding words are arbitrary shuffles of $n + j$ letters u and n letters d, and similarly for $j \leq 0$, they are shuffles of n letters u and $n - j$ letters d. Hence the number of walks of length $2n + |j|$ ending at ordinate j is

$$\binom{2n + |j|}{n}.$$

It will be convenient to express enumerative results in terms of languages and generating functions. In this case, the language $\mathcal{V}$ of walks on the vertical



(a) On an axis, (b) stretched, (c) and rotated.

Figure 9.1. Three representations of the one-dimensional walk *duuudu*.

axis is just $\{u, d\}^*$. Equivalently, in the algebra $\mathbb{Q}\langle\langle u, d\rangle\rangle$ of formal power series in non commuting variables, the language $\mathcal{V}$ (viewed as the formal sum of its words) is uniquely defined by the linear equation:

$$\mathcal{V} = \varepsilon + (u + d)\mathcal{V}, \tag{9.1.1}$$

which corresponds to the non ambiguous decomposition: "a walk is either the empty walk or made of a step u or d followed by a walk".

Define now $\delta(w) = |w|_u - |w|_d$ for any word w on $\mathcal{S}$, so that $\delta(w)$ is the *final ordinate* of the walk w. The *generating function* of the language $\mathcal{V}$ with respect to the length (variable t) and the final ordinate (variable y) is

$$V(t; y) = \sum_{w \in \mathcal{V}} t^{|w|} y^{\delta(w)},$$

which is an element of the algebra $\mathbb{Q}(y)[[t]]$ of formal power series in the variable t with coefficients that are rational functions in y.

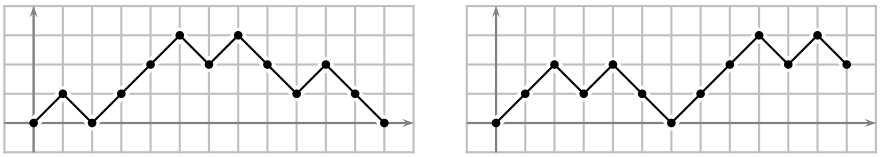
Observe that $|.|$ and δ are morphisms of monoïds $(\mathcal{S}^*, \cdot) \to (\mathbb{Z}, +)$, so that $V(t; y)$ can be viewed as the commutative image of $\mathcal{V}$ by the morphism of algebra $w \mapsto t^{|w|} y^{\delta(w)}$ from $\mathbb{Q}\langle\langle u, d\rangle\rangle$ to $\mathbb{Q}(y)[[t]]$. Taking the commutative image of Equation 9.1.1, the generating function $V(t; y)$ satisfies:

$$V(t; y) = 1 + (ty + ty^{-1})V(t; y).$$

An explicit expression of $V(t; y)$ follows, and its expansion of course agrees with the previous direct enumeration:

$$V(t; y) \;=\; \frac{1}{1 - (y + y^{-1})t} \;=\; \sum_{m=0}^{+\infty} \sum_{k=0}^{m} \binom{m}{k} t^m y^{m-2k}.$$

The commutative image mechanism produces *a priori* a formal power series of $\mathbb{Q}(y)[[t]]$, but, as in the present example, it retains properties of the initial language: the series $V(t; y)$ of the rational language $\{u, d\}^*$ is a rational function of t and y, *i.e.* belongs to $\mathbb{Q}(t, y)$. Walks with more general steps are dealt with in a similar way: for instance the language $\mathcal{W}$ associated to walks in $\mathbb{Z}^2$ is $\mathcal{S}^*$



(a) a Dyck word, (b) a Dyck prefix.

Figure 9.2. The family of Dyck words (stretched representations).

and the generating function of these walks with respect to the length and the coordinates of the endpoint is:

$$W(t;x,y) = \frac{1}{1-(x+x^{-1}+y+y^{-1})t}.$$

Another illustration is given by the family of walks that never immediately undo a step they have just done. Their language is the set of words avoiding the factors $\{ud, du, lr, rl\}$ which is well known to be rational. Accordingly their generating function with respect to the length and the coordinate of the endpoints belongs to $\mathbb{Q}(t,x,y)$. Conversely, when the generating function of a set of objects is rational, it is natural to try to encode them by words of a rational language.

9.1.2. Walks on a half line and Catalan's factorization

We shall now consider walks that stay on the upper half axis ($x = 0,\ y \geq 0$). More precisely let the *depth* of w be the absolute value of the minimal ordinate $\delta(v)$ for all prefixes v of w. Walks that stay on the upper half axis are exactly the walks with depth zero, and this condition is called the *nonnegative prefix condition*. Loops satisfying the nonnegative prefix condition are often called *Dyck words* on the alphabet $\{u,d\}$. In turn, walks satisfying the nonnegative prefix condition are sometimes referred to as *Dyck prefixes*, since any of them can be completed into a Dyck word. See Figure 9.2 for examples. Let $\mathcal{D}$ denote the language of Dyck words and $\mathcal{D}_n$ the set of Dyck words of length $2n$. The following lemma gives a central role to Dyck words.

Lemma 9.1.1 (Catalan's factorization). *The language $\{u,d\}^*$ of one-dimensional walks admits the following non ambiguous decomposition:*

$$\{u,d\}^* = (\mathcal{D}d)^*\mathcal{D}(u\mathcal{D})^*.$$

More precisely, the language of walks with depth ℓ and ending at ordinate j is

$$(\mathcal{D}d)^\ell\mathcal{D}(u\mathcal{D})^{j+\ell}$$

Proof. For any word w on the alphabet $\{u,d\}$ with depth ℓ and final ordinate j, such a factorization is obtained at first passages from ordinate $i+1$ to i for

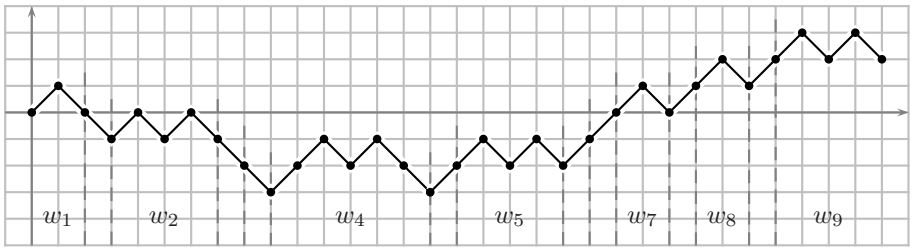


Figure 9.3. Catalan's factorization of a walk in $(\mathcal{D}d)^3\mathcal{D}(u\mathcal{D})^5$.

$i = -1, \ldots, -\ell$ and last passages from ordinate i to $i+1$ for $i = -\ell, \ldots, j-1$. The uniqueness of the decomposition follows from the fact that any strict prefix v of a word in $\mathcal{D}d$ satisfies $\delta(v) \geq 0$ by definition of $\mathcal{D}$, and hence does not belong to $\mathcal{D}d$. ∎

Catalan's factorization immediately allows us to derive the total number of walks on the half line.

PROPOSITION 9.1.2. *The number of Dyck prefixes of length m is*

$$\binom{m}{\lfloor \frac{m}{2} \rfloor}.$$

Proof. A Dyck prefix of even length is a walk with depth zero and even final ordinate $2j$ for some integer $j \geq 0$. According to Lemma 9.1.1, the language of these words is $\mathcal{D}(u\mathcal{D})^{2j}$. Upon changing the j first factors u in factors d, words of length $2n$ in this language are in bijection with words of length $2n$ in the language $(\mathcal{D}d)^j\mathcal{D}(u\mathcal{D})^j$, *i.e.* with words of the language of loops with depth j. Hence Dyck prefixes of length $2n$ are in bijection with loops of the same length, and their number is $\binom{2n}{n}$.

Similarly, a Dyck prefix of odd length ends at ordinate $2j+1$, for some $j \geq 0$. But words of equal length in the languages $\mathcal{D}(u\mathcal{D})^{2j+1}$ and $(\mathcal{D}d)^j\mathcal{D}(u\mathcal{D})^{j+1}$ are in bijection. The union of the last languages for all $j \geq 0$ is the set of words w with $\delta(w) = 1$, $\binom{2n+1}{n}$ of which have length $2n+1$. ∎

The previous proof can be summarized as follows: find a factorization into Dyck factors separated by some specific steps (typically first or last passages), and then reorganize the factorization without modifying the Dyck factors. We shall apply this principle again to give a bijective enumeration of Dyck words.

PROPOSITION 9.1.3. *The number of loops of length $2n$ that stay on the half axis ($x = 0,\ y \geq 0$) is the n-*th Catalan number*:*

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Proof (as a corollary of Proposition 9.1.2). Removing the last step of a Dyck prefix of length $2n+1$ yields a prefix of length $2n$. In this way every Dyck prefix of length $2n$ is obtained twice, except for Dyck paths that are obtained only once. Hence $\binom{2n+1}{n} = 2\binom{2n}{n} - \operatorname{Card}\mathcal{D}_n$, and the formula follows. ∎

Proof (direct bijection). We prove the relation $(n+1)\operatorname{Card}\mathcal{D}_n = \binom{2n}{n}$ by giving a bijection between the set of pairs (v, v') with $vv' \in \mathcal{D}_n$ and v empty or ending with a letter u, and the set of loops of length $2n$. To do that we first state two factorizations that follow from Lemma 9.1.1:

- the set of pairs (v, v') as above with $\delta(v) = \ell$ is $(\mathcal{D}u)^\ell \times \mathcal{D}(d\mathcal{D})^\ell$;
- the set of loops with depth ℓ is $(\mathcal{D}d)^\ell\mathcal{D}(u\mathcal{D})^\ell$.

Exchanging u and d factors in these decompositions leads to the announced bijection. ∎

The same idea allows to refine the enumeration of Dyck prefixes.

PROPOSITION 9.1.4. *The number of Dyck prefixes of length $2n+j$ and final ordinate $j \geq 0$ is*

$$\frac{j+1}{n+j+1}\binom{2n+j}{n}.$$

Proof. We prove the formula by giving a bijection between pairs (w, i) where w is a walk with $\delta(w) = j$ and $i \in \{0, \dots, j\}$, and pairs (w', k) where w' is a Dyck prefix with $\delta(w') = j$ and $k \in \{0, \dots, n+j\}$:

- to any pair (w, i) as above, associate $(w_i, \dots, w_j, w_0, \dots, w_{i-1})$ where w_0 is the loop and the other w_ℓ are the Dyck paths such that $w = w_0uw_1 \cdots uw_j$ (this is the decomposition at the last passages at level $0, \dots, j$).
- to any pair (w', k) as above, associate $(w'_0, \dots, \widehat{w}'_i, \dots, w'_j)$, where the w'_ℓ are the Dyck words such that $w' = w'_0uw'_1 \cdots uw'_j$, i is the index of the w'_i containing or following the kth letter u in the word uw', and $\widehat{w}'_i = (v, v')$ is the factorization of w'_i after this letter.

The bijection in the second proof of Proposition 9.1.3 allows to transform the pair $\widehat{w}'_i = (v, v')$ in a loop, so that both sets are associated to the same set of sequences of $j+1$ walks. ∎

9.1.3. Walks on a half plane and algebraic series

Walks in the half plane ($y \geq 0$) are hardly more complicated to enumerate than walks on the half line. Indeed, as words on the alphabet $\mathcal{S}$, these walks are completely characterized by the fact that all their prefixes v contain at least as many letters u as letters d. Hence the associated language is the set of shuffles of vertical Dyck prefixes with sequences of horizontal steps. Various formulas can be derived from this characterization: for instance, the number of loops of length $2n$ that stay in the half plane ($y \geq 0$) is

$$\sum_{k=0}^{n} \binom{2n}{2k}\binom{2k}{k}C_{n-k}.$$

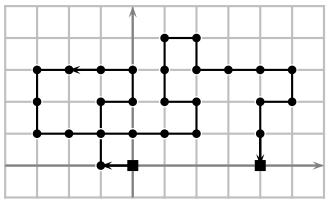


Figure 9.4. An excursion in the half plane.

Rather than going further in this direction, we shall observe that the set of these walks is an algebraic language and return to generating functions. Consider the alphabet $\mathcal{A}_k = \{u, d, x_1, \ldots, x_k\}$, and the monoïd morphism δ defined as previously by $\delta(w) = |w|_u - |w|_d$. The language $\mathcal{M}^{(k)}$ of *k-colored Motzkin words* is the set of words w on the alphabet $\mathcal{A}_k$ satisfying $\delta(w) = 0$ and the nonnegative prefix property. For $k = 0$ this is the Dyck language. For $k = 2$, upon setting $x_1 = l$, $x_2 = r$, bicolored Motzkin words are *excursions in the half plane*, *i.e.* walks in the half plane ($y \geq 0$) that finish on the axis ($y = 0$).

The language of k-colored Motzkin words admits an algebraic description:

$$\mathcal{M}^{(k)} = \varepsilon + (x_1 + \ldots + x_k)\mathcal{M}^{(k)} + u\mathcal{M}^{(k)}d\mathcal{M}^{(k)}, \tag{9.1.2}$$

which derives immediately from the non ambiguous decomposition of any non empty Motzkin word at its smallest non empty prefix v such that $\delta(v) = 0$. Taking the commutative image, the generating function $M^{(k)}(t) = \sum_{w \in \mathcal{M}^{(k)}} t^{|w|}$ of the Motzkin language with respect to the length satisfies the equation:

$$M^{(k)}(t) = 1 + ktM^{(k)}(t) + t^2 M^{(k)}(t)^2. \tag{9.1.3}$$

Observe that this equation completely determines $M^{(k)}(t)$, since it has a unique solution in the space of formal power series in the variable t (as can be checked by induction, extracting the coefficient of t^n on both sides).

Any additive parameter can be taken into consideration in the commutative image. For instance the previous algebraic decomposition yields the following proposition in the case of bicolored Motzkin words.

Proposition 9.1.5. *The generating function for walks in the half plane returning to the axis ($y = 0$), with respect to their length, abscissa of the endpoint and number of vertical steps, is:*

$$M^{(2)}(t; x, z) = \frac{1 - t(x + \frac{1}{x}) - \sqrt{[1 - t(x + \frac{1}{x} + 2z)][1 - t(x + \frac{1}{x} - 2z)]}}{2t^2 z^2}.$$

Proof. Taking the commutative image with the map $w \to t^{|w|} x^{|w|_r - |w|_l} z^{|w|_u + |w|_d}$ yields the equation

$$M^{(2)}(t; x, z) = 1 + t(x + \frac{1}{x})M^{(2)}(t; x, z) + t^2 z^2 M^{(2)}(t; x, z)^2.$$

The discriminant of this equation is

$$\Delta(t; x, z) = [t(x + \frac{1}{x}) - 1]^2 - 4t^2z^2,$$

and among the two roots of the quadratic equation, only the one of the proposition is a formal power series in t. ∎

Equation (9.1.3) shows that the series $M^{(k)}(t)$ satisfies a relation of the form $P(M^{(k)}(t), t) = 0$ with P a polynomial, which means that it is an *algebraic* formal power series. This illustrates the fact that algebraic languages that admit a non ambiguous algebraic description naturally have algebraic generating functions with respect to additive parameters. Conversely, when the generating function of a set of objects is algebraic, one would like to obtain it from an algebraic description of the objects (or more formally from an encoding of the objects by the words of an algebraic language with a non ambiguous description). In this sense, Equation (9.1.2) is more satisfying than Catalan's factorization, even though the commutative image of the latter also induces an algebraic equation.

Expanding the generating function $M^{(2)}(t, 1, 1) = (1 - 2t - \sqrt{1 - 4t})/2t^2$ in powers of t, one observe the following amusing result (*cf.* Problem 9.1.5).

COROLLARY 9.1.6. *The number of bicolored Motzkin words of length n is given by the Catalan number C_{n+1}.*

Loops in the up diagonal quadrant ($x+y \geq 0,\ y \geq x$) are simple to describe: let w be such a loop of length $2n$, and consider the projections of the walk on the two diagonals ($x = y$) and ($x = -y$). Let $\{a, b\}$ be the elementary steps on these two axes, with a corresponding to up steps and b to down steps. Steps in $\mathbb{Z}^2$ have the following projections:

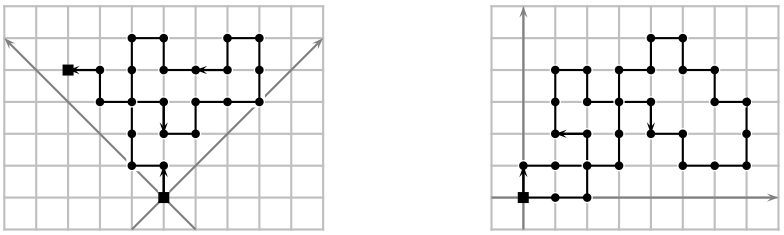
$$u \longrightarrow (a, a) \qquad d \longrightarrow (b, b) \qquad l \longrightarrow (b, a) \qquad r \longrightarrow (a, b)$$

and the projections of w on the diagonals are Dyck words of length $2n$ on $\{a, b\}$; reciprocally any pair of Dyck words of same length over this alphabet corresponds to a loop in the up diagonal quadrant. Hence:

PROPOSITION 9.1.7. *The number of loops of length $2n$ that stay in the diagonal quadrant ($x + y \geq 0,\ y \geq x$) is given by:*

$$C_n^2.$$

More generally, any walk of length $2n + |i| + j$ and endpoint (i, j) in the up diagonal quadrant is described by its projections on the two diagonal axes; these projections are decoupled Dyck prefixes of length $2n + |i| + j$ with respective ordinate of the endpoint $i + j$ and $j - i$. Hence:



(a) A walk in the diagonal up quadrant. (b) A loop in the first quadrant.

Figure 9.5. Walks in quadrants.

PROPOSITION 9.1.8. *The number of walks of length $2n+|i|+j$ and endpoint (i,j) that stay in the diagonal quadrant $(x+y \geq 0,\ y \geq x)$ is given by:*

$$\frac{(j+i+1)(j-i+1)}{(n+j+|i|+1)(n+j+1)}\binom{2n+|i|+j}{n+|i|}\binom{2n+|i|+j}{n},$$

and the total number of walks of length n that stay in the diagonal quadrant $(x+y \geq 0,\ y \geq x)$ is given by

$$\binom{n}{\lfloor \frac{n}{2} \rfloor}^2.$$

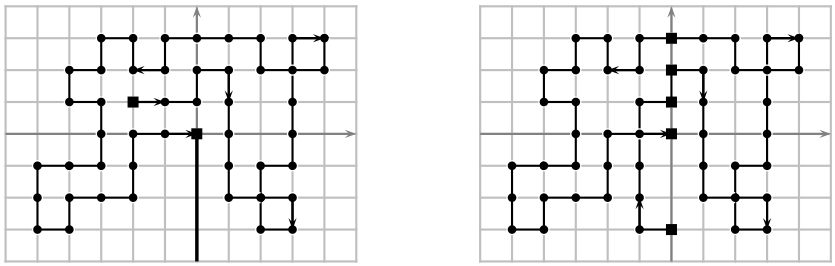
The case of loops in the first quadrant $(x \geq 0,\ y \geq 0)$ is quite similar. These loops are words w on $\mathcal{S}$ such that both restrictions of w to $\{u,d\}$ and to $\{l,r\}$ are Dyck words; hence the language of loops in the first quadrant is the shuffle of the Dyck languages on $\{u,d\}$ and $\{l,r\}$.

PROPOSITION 9.1.9. *The number of loops of length $2n$ that stay in the quadrant $(x \geq 0,\ y \geq 0)$ is given by:*

$$\sum_{k=0}^{n}\binom{2n}{2k}C_kC_{n-k} = \frac{1}{(2n+1)(2n+2)}\binom{2n+2}{n+1}^2.$$

The general case of walks with given length and endpoint or with given length is similar to the case of the diagonal quadrant and left to the reader.

A remarkable consequence of these formulas is that the languages of walks in the diagonal (or in the standard quadrant) cannot be an algebraic language: on the one hand the asymptotic number of walks of length n in the diagonal quadrant, $\binom{n}{\lfloor n/2 \rfloor}^2$, grows like $4^n/n$ when n goes to infinity; on the other hand, the possible asymptotic behaviors of the Taylor coefficients of an algebraic series are classified, and do not include the form $\rho^n n^{-i}$ for i a positive integer; therefore the generating function of walks in the diagonal quadrant is not algebraic, and the associated language cannot be algebraic either.



(a) A walk on the slitplane. (b) The factorization of a walk.

Figure 9.6. On the slitplane.

9.1.4. Walks on the slitplane and the cycle lemma

We call *slitplane* the complement of the half axis $(x = 0, y \leq 0)$ in the square lattice $\mathbb{Z}^2$. Walks on the slitplane are defined as walks that do not touch this half axis except maybe at their startpoint or endpoint, as shown in Figure 9.6(a).

The tool we shall use to enumerate walks on the slitplane is the so-called *cycle lemma*. For any alphabet $\mathcal{A}$ endowed with a morphism $\delta : (\mathcal{A}, \cdot) \to (\mathbb{Z}, +)$, a word w in $\mathcal{A}^*$ is said to have the *Łukasiewicz property* if every strict prefix v of w satisfies $\delta(v) > \delta(w)$.

LEMMA 9.1.10 (Cycle lemma). *Let $\mathcal{A}$ be an alphabet endowed with a morphism $\delta : (\mathcal{A}, \cdot) \to (\mathbb{Z}, +)$. Then a word w in $\mathcal{A}^*$ such that $\delta(w) = -1$ admits a unique factorization $w_1 w_2$ with w_1 non empty such that $w_2 w_1$ has the Łukasiewicz property.*

Proof. Let w_1 be the shortest prefix of w with $\delta(w_1)$ equal to the depth of w. Then $w_2 w_1$ has the Łukasiewicz property. Moreover, let us verify that there is no other such factorization. First assume that w'_1 is a prefix of w shorter than w_1. Then the prefix w'' of w'_2 of length $|w_1| - |w'_1|$ satisfies $\delta(w'') < 0$ and is also a strict prefix of $w'_2 w'_1$. Hence $w'_2 w'_1$ has not the Łukasiewicz property. It remains to consider the case of a prefix w'_1 of w longer than w_1. The suffix w'' of w'_1 of length $|w'_1| - |w_1|$ satisfies $\delta(w) \geq 0$ and is also a suffix of $w'_2 w'_1$. Since moreover $\delta(w'_2 w'_1) = -1$, $w'_2 w'_1$ has not the Łukasiewicz property. ∎

COROLLARY 9.1.11. *Consider the alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, endowed with a morphism δ, and let $n_1, n_2, \ldots, n_k$ be nonnegative integers such that,*

$$\sum_{i=1}^{k} n_i \delta(a_i) = -1.$$

Then the number of words with n_i letters a_i for any $1 \leq i \leq k$ that have the

Łukasiewicz property is equal to:

$$\frac{1}{n_1+\ldots+n_k}\binom{n_1+\ldots+n_k}{n_1,\ldots,n_k}.$$

Proof. For any word w as above, $\delta(w)=-1$, so that the conjugacy class of w contains $|w|$ different words. According to the cycle lemma exactly one of these $n_1+\cdots+n_k$ words has the Łukasiewicz property. The formula follows. ∎

crossref to chapter 1 ?

For $\mathcal{A}=\{u,d\}$ with $\delta(u)=1$, $\delta(d)=-1$, the set of words enumerated by the previous corollary is the Dyck-Łukasiewicz language $\mathcal{D}d$, and we recover Proposition 9.1.3.

def of code needed?

COROLLARY 9.1.12. *Let $\mathcal{C}$ be a code for a set of words on the alphabet $\mathcal{A}$. Then the generating function (with respect to the length) for Łukasiewicz words w in $\mathcal{C}^*$ such that $\delta(w)=-1$ is equal to*

$$[y^{-1}]\log\frac{1}{1-C(t;y)},$$

where $C(t;y)$ is the generating function of the code $\mathcal{C}$ with respect to the length (variable t) and to δ (variable y).

Proof. The generating function of words on the alphabet $\mathcal{A}$ with k factors in $\mathcal{C}$ is $C(t;y)^k$. Restricting the generating function to words w with $\delta(w)=-1$ is done by taking the coefficients of y^{-1} in the series. The fraction of these words that have the Łukasiewicz property is then $1/k$, so that their generating function is

$$\sum_{k\geq 1}\frac{1}{k}[y^{-1}]C(t;y)^k=[y^{-1}]\log\frac{1}{1-C(t;y)}.$$

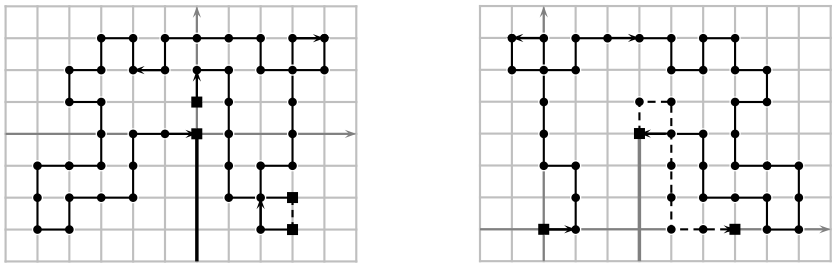
∎

To study walks on the slitplane, it is natural to decompose them at points where they touch the vertical axis ($x=0$), as shown in Figure 9.6: any walk w on the plane that finishes on the vertical axis can be uniquely factored into vertical steps on this axis and primitive excursions in the left or right half plane; in other terms, the language of these walks is

$$(u+d+l\mathcal{M}^{(l)}r+r\mathcal{M}^{(r)}l)^*$$

where $\mathcal{M}^{(l)}$ and $\mathcal{M}^{(r)}$ respectively denote the set of excursions in the left half plane ($x<0$) and in the right one ($x>0$). Hence the set $\{u,d\}\cup l\mathcal{M}^{(l)}r\cup r\mathcal{M}^{(r)}l$ forms a code $\mathcal{C}$ for walks on the plane ending on the vertical axis: these walks can thus be viewed as walks on the axis ($x=0$) with the infinite set of steps $\mathcal{C}$.

To apply the cycle lemma to walks on the slitplane, we consider again the morphism $\delta(w)=|w|_u-|w|_d$. Let us single out the class of walks on the slitplane that start at position $(0,1)$ and end on the half axis at position $(0,0)$: these walks are exactly the Łukasiewicz words w in $\mathcal{C}^*$ such that $\delta(w)=-1$.



(a) A walk w on the slitplane. (b) The half plane excursion $\varphi(w)$.

Figure 9.7. On the slitplane.

PROPOSITION 9.1.13. *The number of walks on the slitplane with startpoint* $(0,0)$*, endpoint* $(0,1)$ *and length* $2n+1$ *is:*

$$C_{2n+1} = \frac{1}{2n+2}\binom{4n+2}{2n+1}.$$

Proof. Let $C(t;y)$ be the commutative image of $\mathcal{C}$, so that $1/(1-C(t;y))$ is the generating function of words on the code $\mathcal{C}$. Observe that a $\pi/2$- (respectively $-\pi/2$-) rotation maps bijectively words of length n in $\mathcal{M}^{(l)}$ (resp. $\mathcal{M}^{(r)}$) on words of length n in the bicolored Motzkin language $\mathcal{M}^{(2)}$, hence Proposition 9.1.5 yields:

$$\begin{aligned}\log\frac{1}{1-C(t;y)} &= \frac{1}{2}\left(\log\frac{1}{1-t(y+\frac{1}{y}+2)}+\log\frac{1}{1-t(y+\frac{1}{y}-2)}\right)\\ &= \frac{1}{2}\sum_{n\geq 1}\frac{t^n}{n}\left((y+\tfrac{1}{y}+2)^n+(y+\tfrac{1}{y}-2)^n\right).\end{aligned}$$

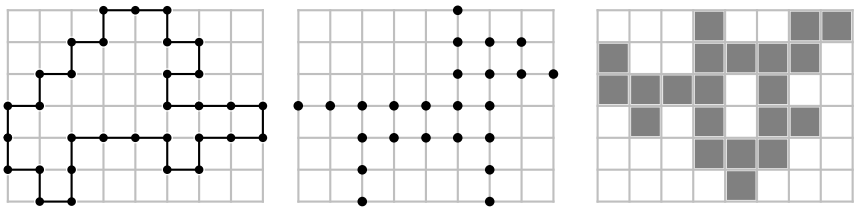
The formula follows by extracting the coefficient of y^{-1} and resumming. ∎

The above proof does not yield an interpretation of the occurrence of Catalan numbers in Proposition 9.1.13. We conclude this section with a more direct derivation.

Proof of Proposition 9.1.13 (bis). We are interested in walks w such that
- $|w|_l = |w|_r$, and $|w|_d = |w|_u + 1$,
- and for any strict prefix v of w, either $|v|_l \neq |v|_r$, or $|v|_u \geq |v|_d$.

The first condition accounts for the displacement between the startpoint and endpoint, while the second one ensures that the walks stay in the slitplane. Let us describe a one-to-one correspondence φ between these walks and excursions of even length in the half plane (bicolored Motzkin words). The result then follows from Corollary 9.1.6.

Let w be a walk as in the proposition. Since $|w|_d = |w|_u + 1$, Lemma 9.1.10 yields a unique factorization of w in $w_1 d w_2$ such that each proper prefix v of



(a) A polygon, (b) an animal, (c) and a polyomino.

Figure 9.8. Three related classes of objects.

w_2w_1d satisfies $|v|_u \geq |v|_d$: this is the factorization at the first arrival to the lowest level. Let $\bar{w_2}$ be the walk that is symmetric to w_2 with respect to the vertical axis ($x = 0$), and $\varphi(w)$ be equal to $\bar{w_2}w_1$. Then $\varphi(w)$ is a bicolored Motzkin word, corresponding to an excursion in the half plane ($y \geq 0$) of length $2n$. Moreover the factorization $\bar{w_2}w_1$ of $\varphi(w)$ is the factorization at the first passage on the lowest point on the vertical line of equidistance between the startpoint and endpoint of $\varphi(w)$.

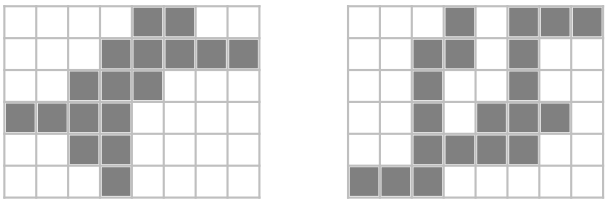
Conversely, given a bicolored Motzkin word w', let $w'_1w'_2$ be its factorization at the first passage on the lowest point on the vertical line of equidistance between its startpoint and endpoint. Let $\psi(w') = w'_2d\bar{w}'_1$. The walk $\psi(w')$ is clearly a walk in the slitplane from $(0, 1)$ to $(0, 0)$, and $\varphi(\psi(w')) = w'$. Moreover, $\psi(\varphi(w)) = w$ for any walk w as in the proposition, and this concludes the proof. ∎

As discussed in Section 9.1.3, the language of bicolored Motzkin words has a very natural algebraic decomposition. However this decomposition does not carry very well through the bijection.

9.2. Sampling: polygons, animals and polyominoes

A walk on the square lattice $\mathbb{Z}^2$ is called a *self-avoiding walk*, or a *path*, if it visits at most once each vertex of the lattice. A *self-avoiding polygon*, or simply in this text, a *polygon*, is a self-avoiding loop.

An *animal* is a set A of vertices of the lattice such that any two vertices of A are connected by a path visiting only vertices of A. Animals are considered up to translations of the lattice. Placing a unit square centered on each vertex of A, we obtain a *polyomino*. The latter are however more naturally defined as edge-connected sets of squares of the lattice. These definitions are illustrated by Figure 9.8. Each polygon is the *contour* (or the boundary) of a simply-connected polyomino, and in the plane this is a one-to-one correspondence (see Figures 9.10, 9.11 and 9.12). In particular the *length* of a polygon corresponds to the *perimeter* of the polyomino. A polygon has moreover *dimension* (p, q) if the smallest rectangle in which it can be inscribed has horizontal width p



(a) A convex polyomino. (b) A directed polyomino.

Figure 9.9. Subclasses of polyominoes.

and vertical width q. Finally the area of a polyomino is its number of cells, corresponding for animals to the number of vertices.

Little can be said from the enumerative point of view on animals, polygons or polyominoes in general. Two ideas have however been particularly successful for defining subclasses amenable to mathematical study and still of interest: restriction to convex or to directed objects. A polygon of dimension (p, q) is *convex* if its length is $2p+2q$. This definition stresses the fact that convex polygons are in some sense the most extended polygons, and do not make meanders. An equivalent, but maybe more appealing interpretation is in terms of polyominoes: a polyomino is *convex* if its intersection with any horizontal or vertical line is connected. A polyomino (respectively an animal) is *directed* if there is a cell (resp. a vertex) from which every cell (resp. vertex) can be reached by a path going up or right inside the object. These definitions are illustrated by Figure 9.9.

9.2.1. Generalities on sampling

Together with the enumerative questions, much interest has been given to the properties of random animals, polyominoes and polygons. By random is meant here the uniform distribution: objects of equal size are given equal probability to appear. We illustrate this trend by concentrating on the derivation of random generators. In order to describe these algorithms, we assume that we have at our disposal a perfect random number generator $\textsc{Rand}(m, n)$ that outputs an integer of the interval $[m, n]$ chosen with uniform probability: for all $m \leq i \leq n$,

$$\mathbf{P}(\textsc{Rand}(m, n) = i) \;=\; 1/(n - m + 1).$$

We assume unit cost for arithmetic operations and for calls to the generator $\textsc{Rand}()$. These randomness and complexity models are justified by the fact that our algorithms only sample and compute on integers that are polynomially bounded in the size of the objects generated.

We shall need a random sampler for elements of $\mathfrak{S}(w)$, the set of permutations of the letters of a fixed word w. The following algorithm does this by applying a random permutation to the letters of w.

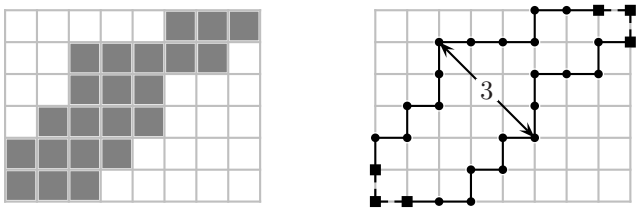


Figure 9.10. A parallelogram polyomino and its contour.

```
RandPerm(w)
1  for i ← 2 to |w| do
2        Swap(w[i], w[Rand(1, i)])
3  return w
```

Lemma 9.2.1. RandPerm(w) *returns in linear time a random element of* $\mathfrak{S}(w)$ *under the uniform distribution: for all* $w' \in \mathfrak{S}(w)$,

$$\mathbf{P}(\text{RandPerm}(w) = w') \;=\; \frac{1}{\text{Card}(\mathfrak{S}(w))}.$$

Proof. A permutation σ on the set $\{1, \ldots, n\}$ has a unique decomposition as a product $\sigma = \tau_n \ldots \tau_2$ of transpositions of the form $\tau_i = (j_i, i)$ with $1 \leq j_i \leq i$, and conversely any such decomposition provides a permutation. Therefore, the call RandPerm(w) on a word w with distinct letters generates a uniform random permutation of the letters. Upon labelling identical letters by their initial place, we conclude that uniformity is also preserved in the general case. ∎

In the rest of this part, we describe random sampling algorithms for convex polygons and directed animals.

9.2.2. Parallelogram polyominoes and the cycle lemma

A convex polyomino P is a *parallelogram polyomino* if its contour contains the bottom left and top right corners of its bounding box. Equivalently, its contour must be a *staircase polygon*, *i.e.* a polygon made of two up-right directed paths, meeting only at their extremities. These upper and lower paths, being directed, can be coded with two letters. For later purpose, it will be convenient to code them on the alphabet $\{h, v\}$, with h standing for a horizontal step and v standing for a vertical step. Starting from the bottom left corner, let vw_1h be the word coding the upper path, and hw_2v be the word coding the lower path (there is no choice for first and last letters). If P has dimension $(p+1, q+1)$ then $|w_1|_h = |w_2|_h = p$ and $|w_1|_v = |w_2|_v = q$. The *reduced code* of a staircase polygon w is the word on the alphabet $\mathcal{A} = \left\{\binom{v}{h}, \binom{v}{v}, \binom{h}{h}, \binom{h}{v}\right\}$ obtained by stacking the two words w_1 and w_2. In the example of Figure 9.10, the two paths are respectively $vw_1h = v \cdot vhvhvvhhhvhh \cdot h$ and $hw_2v = h \cdot hhvhvhvvhhvh \cdot v$.

Words on $\mathcal{A}$ that code for staircase polygons are characterized by the facts that they have an equal number of letters h in both rows, and that their prefixes contain at least as many letters $\binom{v}{h}$ as letters $\binom{h}{v}$: indeed, the morphism δ induced by $\{\delta\binom{v}{h} = 1,\ \delta\binom{h}{v} = -1,\ \delta\binom{h}{h} = \delta\binom{v}{v} = 0\}$ measures the distance between the upper and lower paths along diagonals, and the positive prefix property expresses the condition that the upper and lower paths do not meet before their endpoint. Codes of staircase polygons are thus essentially bicolored Motzkin words.

This characterization suggests to construct staircase polygons by applying the cycle lemma to words of the set $S(p, q)$ of words of length $p + q + 1$ on $\mathcal{A}$ with $p + 1$ letters h and q letters v in the first row, and p letters h and $q + 1$ letters v in the second row:

```
STAIRCASE(p, q)
 1  w'_1 ← RANDPERM(h^{p+1} v^q)            ▷ generate w' = (w'_1 / w'_2) ∈ S(p, q)
 2  w'_2 ← RANDPERM(h^p v^{q+1})
 3  (m, δ_m) ← (0, 0)                       ▷ seek the position m of the
 4  δ ← 0                                   ▷ leftmost minimum w.r.t δ
 5  for i ← 1 to p + q + 1 do
 6      if (w'_1[i], w'_2[i]) = (v, h) then
 7          δ ← δ + 1
 8      elseif (w'_1[i], w'_2[i]) = (h, v) then
 9          δ ← δ − 1
10          if δ < δ_m then
11              (m, δ_m) ← (i, δ)
12  (w_1 h, w_2 v) ← SHIFT((w'_1, w'_2), m)  ▷ get the conjugate at position m
13  return (v w_1 h, h w_2 v)
```

PROPOSITION 9.2.2. STAIRCASE(p, q) *produces the code of a random uniform staircase polygon with dimension* $(p + 1, q + 1)$ *in linear time.*

Proof. Let us first use the cycle lemma to derive the number of staircase polygons. The number of words in $S(p, q)$ is $\mathrm{Card}(S(p, q)) = \binom{p+q+1}{q}\binom{p+q+1}{p}$. Then among the $p + q + 1$ cyclic shifts of any word $w' \in S(p, q)$, exactly one is of the form $w\binom{h}{v}$ with w having the positive prefix property. Hence the number of staircase polygons with dimension $(p + 1, q + 1)$ is $\frac{1}{p+q+1}\binom{p+q+1}{q}\binom{p+q+1}{p}$.

The algorithm STAIRCASE() generates a word uniformly at random in the set $S(p, q)$, and computes its unique cyclic shift coding for a staircase polygon. The probability to get the code of a given polygon P is thus the sum of the probability to get each of its cyclic shifts. But the code of P admits $p + q + 1$ distinct cyclic shifts, and each of these word has probability $1/\mathrm{Card}(S(p, q))$ to be obtained. Thus the probability to get P is $(p + q + 1)/\mathrm{Card}(S(p, q))$, *i.e.* depends only on the dimension of P: uniformity is preserved through the cycle lemma. ∎

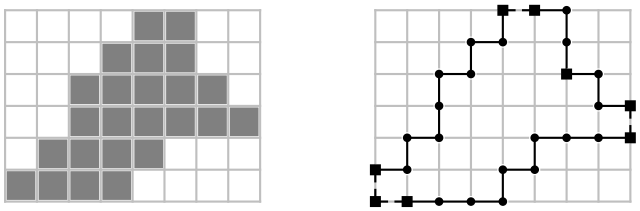


Figure 9.11. A directed convex polyomino and its contour.

9.2.3. Directed convex polyominoes and Catalan's factorization

Directed convex polyominoes are characterized among convex polyominoes by the property that their contour contains the bottom left corner of their bounding box. In other terms contours of directed convex polyominoes are *unimodal polygons*, *i.e.* shuffles of a word of the language u^*d^* and a word of the language r^*l^*. Let us consider an unimodal polygon with dimension $(p+1, q+1)$, and decompose it into an upper path and a lower path both starting from the bottom left corner and of length $p+q+2$, and respectively obtained in clockwise and counterclockwise direction. Let w'_1 and w'_2 be the codes of these two paths on the alphabet $\{h, v\}$. In the example of Figure 9.11, the two paths are respectively $w'_1 = vhvhvvhvhvhhvv$ and $w'_2 = hhhhvhvhhhvhvh$. The following properties of w'_1 are immediate consequences of the definition of unimodal polygons:

1. the word w'_1 starts with a letter v;
2. it contains at least $q+1$ letters v;
3. the first $q+1$ letters v code up steps, the other ones down steps;
4. the $(q+1)$th letter v is followed by a letter h.

The last property accounts for the right turn that the path has to make when reaching the upper boundary. Define the *reduced code* w_1 as obtained from w'_1 by deleting the two redundant letters given by Properties 1 and 4 above. Similarly the reduced code w_2 is obtained by deleting from w'_2 the first letter (that is a letter h) and the letter following the $(p+1)$th letter h (that is a letter v). Let w be the word on $\mathcal{A}$ obtained by stacking w_1 and w_2. Then again all prefixes of w contain at least as many letters $\binom{v}{h}$ as letters $\binom{h}{v}$. It turns out that this condition is sufficient for w to code an unimodal polygon: this is expressed by the following lemma, the proof of which is left to the reader.

LEMMA 9.2.3. *A word w on $\mathcal{A}$ is the stacked reduced code of an unimodal polygon with dimension $(p+1, q+1)$ if and only if all its prefixes contain at least as many letters $\binom{v}{h}$ as letters $\binom{h}{v}$, and, viewed as a pair of words on $\{h, v\}$, it contains $2p$ letters h and $2q$ letters v.*

In terms of the morphism δ of the previous section, Lemma 9.2.3 implies that a word of $\mathcal{A}^*$ is the code of an unimodal polygon if and only if it is a prefix of Motzkin word on $(\mathcal{A}, \delta)$. These prefixes are similar to prefixes of Dyck words with δ even, and the proof of Proposition 9.1.2 suggests the following algorithm.

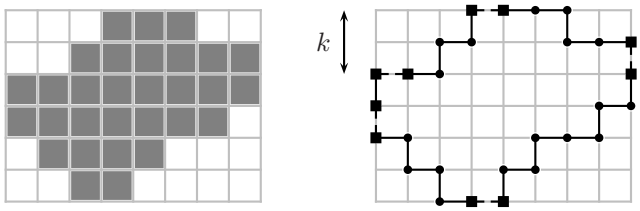


Figure 9.12. A convex polyomino and its contour.

```
UNIMODAL(p, q)
 1  w1 ← RANDPERM(h^p v^q)            ▷ generate w = (w1 over w2) with δ(w) = 0
 2  w2 ← RANDPERM(h^p v^q)
 3  δ ← 0
 4  δm ← 0
 5  for i ← 1 to p + q do
 6       if (w1[i], w2[i]) = (v, h) then
 7            δ ← δ + 1
 8       elseif (w1[i], w2[i]) = (h, v) then
 9            δ ← δ − 1
10            if δ < δm then                    ▷ leftmost minimum found
11                 (δm, w1[i], w2[i]) ← (δ, v, h)   ▷ down step to up step
12  return (w1, w2)
```

PROPOSITION 9.2.4. UNIMODAL(p, q) *produces the reduced code of a random uniform unimodal polygon with dimension* $(p+1, q+1)$ *in linear time.*

Proof. Lines 1, 2 of the algorithm construct a word $\binom{w_1}{w_2}$ satisfying $\delta\binom{w_1}{w_2} = 0$. A straightforward adaptation of the bijection used for Proposition 9.1.2 shows that these words are in one-to-one correspondence with prefixes of Motzkin words: for the current δ, steps $\binom{v}{h}$ play the role of up steps, steps $\binom{h}{v}$ that down steps, and Motzkin factors replace Dyck factors. The algorithm implements the inverse bijection, replacing leftmost down steps at negative levels by up steps.

Since the word $\binom{w_1}{w_2}$ is taken uniformly in the set of words with p letters h and q letters v in both lines, its image is uniform in the set of bicolored Motzkin prefixes with $2p$ letters h and $2q$ letters v. ∎

As a corollary of the previous proof, we also see that the number of unimodal polygons of dimension $(p+1, q+1)$ is $\binom{p+q}{p}^2$.

9.2.4. Convex polyominoes and rejection sampling

The contour of a convex polyomino with dimension $(p+1, q+1)$ can be coded as follows by a pair (w', k): start from the upper point of the contour on the left boundary, and code the path in clockwise direction by a word w' with letters h and v as previously; let moreover k be the distance of the startpoint to the top

border of the bounding box (see Figure 9.12). From the geometry, the following properties of the word w' are immediate:

1. there are $2p+2$ letters h and $2q+2$ letters v; moreover $0 \le k \le q$;
2. the first $p+1$ letters h code right steps, the other $p+1$ left steps;
3. the first k letters v code up steps, the next $q+1$ down steps, and the final $q+1-k$ up steps again;
4. the first letter is a letter h;
5. if $k > 0$ then the kth letter v is followed by a letter h;
6. the $(p+1)$th letter h is followed by a letter v;
7. the $(k+q+1)$th letter v is followed by a letter h;
8. the $(2p+2)$th letter h is followed by a letter v;
9. the letters singled out in 4, 5, 6, 7, and 8 above appear in this order.

These properties do not completely characterize the codes of convex polygons, but this is almost the case, as the reader will verify:

LEMMA 9.2.5. *A pair (w', k) satisfying the nine properties above is the code of a convex polygon if and only if the corresponding walk is a polygon, that is, if it does not visit twice the same point. This property can be checked in linear time by the following algorithm.*

```
CHECKSIMPLE(w', k)
 1  (i_1, δ_1, ε_1) ← (1, q+1−k, +1)          ▷ traversal of w' from the left
 2  (i_2, δ_2, ε_2) ← (2p+2q+3, q−k, −1)      ▷ traversal of w' from the right
 3  for ℓ ← 1 to p+1 do                       ▷ ℓ counts horizontal steps
 4      while w'[i_1] = v do                  ▷ vertical move on top
 5          (i_1, δ_1) ← (i_1+1, δ_1+ε_1)
 6      while w'[i_2] = v do                  ▷ vertical move on bottom
 7          (i_2, δ_2) ← (i_2−1, δ_2+ε_2)
 8      if δ_1 ≤ δ_2 then                     ▷ self-intersection detected
 9          return FALSE
10      if δ_1 = q+1 then                     ▷ top reached
11          ε_1 ← −1
12      if δ_2 = 0 then                       ▷ bottom reached
13          ε_2 ← +1
14      (i_1, i_2) ← (i_1+1, i_2−1)           ▷ next column
15  return TRUE
```

The reduced code (w, k) of a convex polygon is obtained by deleting the redundant letters given by Properties 4, 6, 7, 8, and if $k > 0$ by Property 5. The reduced word w has thus, if $k = 0$, $2p$ letters h and $2q$ letters v, or, if $k > 0$, $2p-1$ letter h and $2q$ letters v. Given the reduced word w and the index k there is an immediate algorithm INSERTREDUNDANTLETTERS(w, k) that reconstructs w' by inserting the missing letters from left to right.

The following generator is based on the rejection principle: words of a superset of the set of codes are generated uniformly at random until a proper code is obtained.

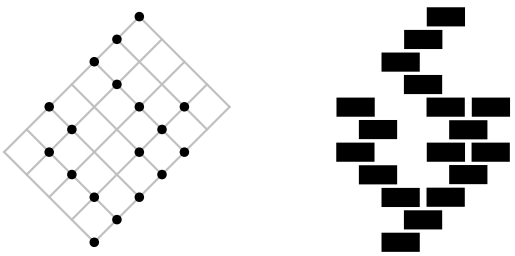


Figure 9.13. A directed animal and the equivalent strict pyramid.

```
CONVEX(p, q)
 1  do  k ← RAND(0, q)
 2      w ← RANDPERM(h^{2p} v^{2q})
 3      if k = 0 or w[2p + 2q] = h then
 4          w' ← INSERTREDUNDANTLETTERS(w, k)
 5          if CHECKSIMPLE(w', k) = TRUE then
 6              return (w', k)
 7  while TRUE
```

PROPOSITION 9.2.6. CONVEX(p, q) *produces the code of a random uniform convex polygon with dimension* $(p + 1, q + 1)$.

Proof. The fact that the output is uniform follows from the following standard rejection argument: when the algorithm stops, the probability to output a given code is proportional to the probability to get this code as an element of the superset; but elements of the superset are sampled uniformly, *i.e.* have the same probability to be generated. ∎

The expected complexity of the algorithm CONVEX() depends on the comparison between the size $(q + 1)\binom{2p+2q}{2p}$ of the superset $S_{p,q}$ in which k and w are sampled, and the size of the set $P_{p,q}$ of convex polygons with dimension $(p + 1, q + 1)$. More precisely, each loop takes linear time, the probability of success of a loop is $s_{p,q} = \mathrm{Card}(P_{p,q})/\mathrm{Card}(S_{p,q})$, and the number of loops is a geometric random variable with expectation $1/s_{p,q}$. The explicit computation of $\mathrm{Card}(P_{p,q})$ shows that this last value is bounded by a constant, but we do not include the details here (see Problem 9.2.2).

PROPOSITION 9.2.7. *The call* CONVEX(p, q) *has expected linear complexity.*

9.2.5. Directed animals

Upon rotating the lattice counterclockwise by $\pi/4$, directed animals can be given an elegant interpretation in terms of *heaps of bricks*: cells are viewed as bricks exposed to the gravity law with the bottom brick lying on the floor; the condition that animals are directed, *i.e.* that there always exists a path

downward to the bottom cell, is equivalent to the fact that every brick leans on one brick below and cannot fall.

To be more precise, let us give a definition of heaps of bricks. The alphabet of bricks is $\mathcal{B} = \{(i, i+1), i \in \mathbb{Z}\}$. Two bricks b, b' of $\mathcal{B}$ commute if and only if, as subsets of $\mathbb{Z}$, $b \cap b' = \emptyset$. Two words are equivalent, $w \equiv w'$, if one can be obtained from the other by a sequence of commutations of adjacent commuting bricks. A heap of bricks is an element of the associated partially commutative monoïd, *i.e.* an equivalence class for the relation $\equiv$. The set of minimal bricks of a heap w is the set $\min(w) = \{b \mid \exists w', w \equiv bw'\}$. A *pyramid* at abscissa i is a heap such that $\min(w) = \{(i, i+1)\}$.

The canonical geometric representation of a heap induced by the gravity law corresponds to the standard Cartier-Foata normal form of the heap: reading a heap from left to right in lines from bottom to top yields a word w of the form $w_1 \cdots w_k$ with each block w_i made of commuting letters and such that for each letter b of w_{i+1} there is a letter b' of w_i with $b \cap b' \neq \emptyset$. A heap is *strict* if moreover no two consecutive blocks of the normal form have a brick in common: in other terms in a strict heap a brick $(i, i+1)$ always lean on a brick $(i-1, i)$ or $(i+1, i+2)$, not on another brick $(i, i+1)$.

From the geometric interpretation of pyramids of bricks and the initial discussion of this paragraph, the following lemma is immediate.

LEMMA 9.2.8. *Directed animals "are" strict pyramids of bricks.*

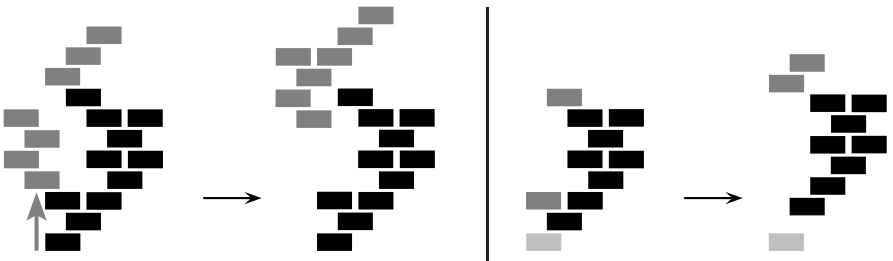
This interpretation of directed animals in terms of pyramids of bricks allows to perform decompositions that would otherwise be very difficult to explain. First define a *semi-pyramid* to be a pyramid without bricks on the left hand side of the bottom brick. Then the following two decompositions are obtained by pushing upward a brick and all the bricks that lay above it, or indirectly lean on it:

- a strict pyramid of bricks is either a strict semi-pyramid, or can be factored, by pushing upward the lowest brick with abscissa -1, into a strict pyramid at abscissa -1 stacked over a strict semi-pyramid;
- a strict semi-pyramid is reduced to a brick, or to a strict semi-pyramid at abscissa 1 over a brick, or can be factored, by pushing upward the second lowest brick with abscissa 0, into a strict semi-pyramid at abscissa 0 stacked over a strict semi-pyramid at abscissa 1 over a brick.

This joint decomposition is isomorphic to the joint decomposition of prefixes of words and of words of the Motzkin language on the alphabet $\{a, b, x_1\}$:

- a prefix of Motzkin word is either a Motzkin word or can be decomposed as uav with u a Motzkin word and v a prefix of Motzkin word.
- a Motzkin word is reduced to the empty word ε, or is of the form $x_1 u$ with u a Motzkin word, or can be decomposed as $aubv$ with u and v two Motzkin words.

These isomorphic decompositions induce a bijection between strict pyramids of n bricks and prefixes of Motzkin words of length $n-1$.



(a) Factorization of a pyramid into a pyramid and a semi-pyramid.

(b) Factorization of a semi-pyramid into a brick and two semi-pyramids.

Figure 9.14. Decomposition of pyramids.

COROLLARY 9.2.9. *Prefixes of Motzkin words can be bijectively transformed into strict pyramids of bricks in linear time.*

The Motzkin language being algebraic, uniform random generation could be done using a recursive approach. We describe instead another application of the rejection principle which is both more elegant and more efficient for this specific problem. Let us consider again the alphabet $\mathcal{A}_k = \{u, d, x_1, \ldots, x_k\}$ and the associated k-colored Motzkin words of Section 9.1.3. A naive algorithm to generate uniform random prefixes of k-colored Motzkin words of length n consists in generating uniform random words of $(\mathcal{A}_k)^n$ and rejecting. However a simple calculation shows that the probability of success is of order $O(n^{-1/2})$ thus giving an algorithm with expected complexity $O(n^{3/2})$. A slight refinement on this idea is to observe that rejection can be decided on the fly. This turns out to be surprisingly efficient.

```
FLORENTINEREJECTION(n, k)
 1  do  w ← ε
 2      for i ← 1 to n do                ▷ generate from left to right
 3          w[i] ← RAND(1, k + 2)
 4          if w[i] = k + 1 then
 5              δ ← δ + 1
 6              w[i] ← u
 7          elseif w[i] = k + 2 then
 8              δ ← δ − 1
 9              w[i] ← d
10              if δ < 0 then            ▷ if a negative prefix is detected
11                  break                ▷ restart from scratch
12  while i ≠ n + 1                      ▷ until w is a valid n letters word
13  return w
```

This algorithm obviously produces a prefix of k-colored Motzkin word.

LEMMA 9.2.10. *The function* FLORENTINEREJECTION(n,k) *generates a random uniform prefix of k-Motzkin word of length n in expected linear time.*

Proof. For simplicity the analysis is presented in the case $k=0$ but the same strategy of analysis applies to the general case (using generating functions instead of elementary counting). It will be convenient to consider that when the construction fails at the ith step of the inner loop, we finish the loop and generate $n-i$ more letters at no cost. This modification of the algorithm do not affect the final result or the cost, but allow us to think at each iteration as producing a uniform random word of $(\mathcal{A}_k)^n$. From this point of view, the Florentine rejection behaves like standard rejection and therefore it is uniform on prefixes.

The probability of success of the inner loop is $p_n = \binom{2n}{n}2^{-2n} = p_n$, and the number of aborted loops is a geometric random variable with expected value $1/p_n = O(n^{1/2})$. Let us now compute the expected cost of a failure: a failure with cost $2i+1$ is obtained for a word w of the form ubv with u a Dyck word of length $2i$ and v in $\{a,b\}^{2n-2i-1}$. Hence the cumulated cost for all these $2^{2n}-\binom{2n}{n}$ words is $\sum_{i=0}^{n-1}(2i+1)C_i 2^{2n-2i-1} = 2^{2n-1}\sum_{i=0}^{n-1}\binom{2i+1}{i}2^{-2i} = O(2^{2n}n^{1/2})$. With $O(n^{1/2})$ aborted loops with cost $O(n^{1/2})$ each, and one successful loop with cost n, the total expected cost is linear as announced. ∎

Florentine rejection thus uses on average a linear number of random bits. As opposed to this a call to RANDPERM(w) for a word w of length n uses about $n\log n$ bits, and this is in general suboptimal from a theoretical point of view. For instance for $w = a^n b^n$, $\log\binom{2n}{n} \sim 2n$ bits should suffice. In this case an optimal solution (on average) is obtained using FLORENTINEREJECTION$(n,0)$ to get a prefix of Dyck words and Catalan's factorization (Proposition 9.1.2) to transform it into a word of $\mathfrak{S}(a^n b^n)$. As opposed to this, it is an open problem in general to sample in linear time from $\mathfrak{S}(w)$ using $O(\log \mathrm{Card}(\mathfrak{S}(w)))$ random bits.

9.3. Coding: trees and maps

A *planar map*[1] is a proper embedding of a connected graph in the plane. Multiple edges and loops are allowed, and proper means that edges are smooth simple arcs which meet only at their endpoints. The faces of a planar map are the connected components of the complement of the graph in the plane: apart from one *infinite face*, all faces are bounded and homeomorphic to disks. All the planar maps we consider are *rooted*: they have an oriented edge, called the *root*, which is incident to the infinite face on its right-hand side. Examples of rooted maps are presented in Figure 9.15.

From now on we shall consider that two planar maps are the same if one can be mapped onto the other (including roots) by an homeomorphism of the plane. However there are still many more planar maps than planar graphs, as illustrated by Figure 9.15. Indeed homeomorphisms of the plane respect the

[1] The word *map* is intended here in its geographic sense, like in road-map.

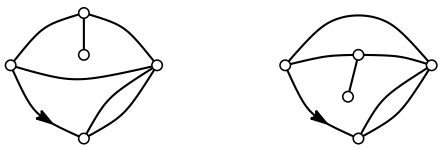


Figure 9.15. Two rooted planar maps with the same underlying graph.

neighborhood of each vertex, so that the circular order of edges around vertices is fixed.

From a combinatorial point of view, a planar map can in fact entirely be specified as follows: label half-edges (or *darts*) and for each half-edge give the names of the opposite half-edge, and of the next half-edge around its origin in counterclockwise direction. As a consequence the number of planar maps with n edges is finite. Moreover these labeled maps capture exactly the level at which algorithms on maps are implemented in computational geometry, using darts as elementary data structures. Carrying on with labeled maps, one could also reach a purely combinatorial setting and eliminate the geometry (at least at the formal level of proofs). However for the sake of conciseness it appears more efficient to keep higher level geometric arguments.

Examples of specific families of planar maps are numerous. A *triangulation of a k-gon* is a planar map without multiple edges such that all bounded faces have degree 3 and the infinite face has degree k (the degree of a face is the number of sides of edges to which it is incident). A *k-valent map* is a planar map such that all vertices have degree k (the degree of a vertex is the number of half-edges to which it is incident).

9.3.1. Plane trees and generalities on coding

A *rooted plane tree*, or hereafter simply a *plane tree* is a planar map with one face. A *planted plane tree* is a plane tree such that the root vertex has degree 1. A *binary tree* is a planted plane tree with vertices of degree 3 and 1 only, respectively called *nodes* and *leaves*. These definitions agree with classical recursive definitions of plane trees: for instance a plane tree can be decomposed as an ordered sequence of subtrees attached to the root.

The *contour traversal* of a planar map is the walk on the vertices and edges of the map that starts from (the right-hand side of) the root edge, and turns around the map in counterclockwise direction so as to visit the boundary of the infinite face. (The reader is encouraged to imagine an ant walking around the map.) The contour traversal of a plane tree visits in particular twice every edge: the first time away from the root vertex, and the second time toward the root vertex. The preorder on the vertices of a planted plane tree is defined by ordering vertices according to the first passage of the contour traversal.

The *Dyck code* of a planted plane tree with $n+1$ edges is the word of length $2n$ on the alphabet $\{u, d\}$ obtained during a contour traversal of the tree by

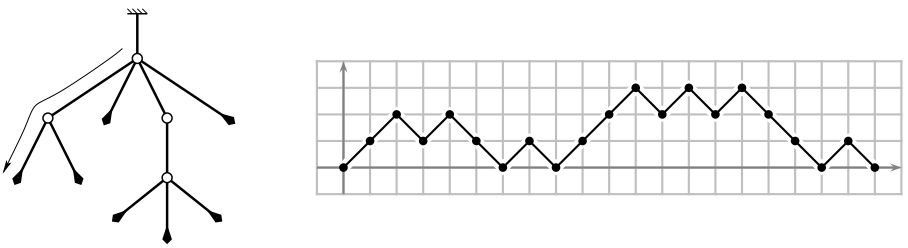


Figure 9.16. A planted plane tree and its Dyck code.

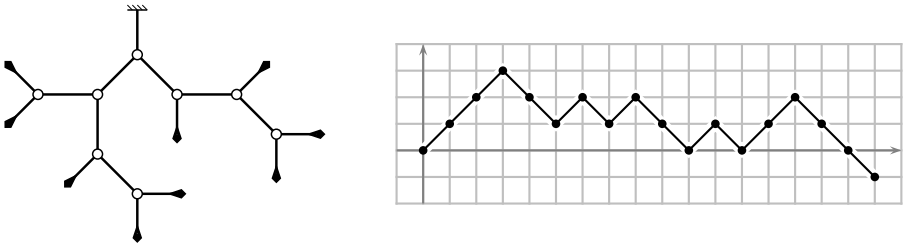


Figure 9.17. A planted binary tree and its prefix code.

writing a letter u each time a non-root edge is visited for the first time (away from the root vertex), and a letter d each time a non-root edge is visited for the second time (toward the root vertex). The reader should convince himself that the Dyck code of a tree characterizes it.

LEMMA 9.3.1. *Dyck encoding is a bijection between planted plane trees with $n+1$ edges and Dyck words of length $2n$. In particular the number of planted plane trees with $n+1$ edges is the nth Catalan number.*

The *prefix* or *Łukasiewicz code* of a planted plane tree with n edges is the word of length n on the alphabet $\{x_i, i \geq 0\}$ obtained during a contour traversal of the tree by writing a letter x_i each time a non-root vertex with degree $i+1$ is visited for the first time. Let us define the morphism δ by $\delta(x_i) = i-1$. Then the prefix code w of a planted plane tree has the Łukasiewicz property (*i.e.* for each strict prefix v of w, $\delta(v) > \delta(w)$). In particular, upon setting $x_2 = u$ and $x_0 = d$, we obtain the following lemma for the case of binary trees:

LEMMA 9.3.2. *Prefix encoding is a bijection between binary trees with n nodes, (and thus $n+2$ leaves and $2n+1$ edges) and words of length $2n+1$ of the Dyck-Łukasiewicz language $\mathcal{D}d$. In particular the number of binary trees with n nodes is the nth Catalan number.*

Recall that the optimal coding problem for a family $\mathcal{C}$ of combinatorial structures consists in finding a function φ that maps injectively objects of $\mathcal{C}$ on words

of $\{0,1\}^*$ in such a way that an object O of size n is coded by a word $\varphi(O)$ of length roughly bounded by $\log_2 \mathrm{Card}(\mathcal{C}_n)$, with $\mathcal{C}_n$ the set of objects of size n. Since the nth Catalan number satisfies $\log C_n \sim 2n$ as n goes to infinity, Dyck codes and prefix codes respectively solve the optimal coding problem for plane trees and for binary trees. On the other hand, the Dyck code of a binary tree with n nodes has length $4n+2$, so that Dyck codes are far from optimality with respect to the family of binary trees: the optimality of a code is relative to the entropy $\log \mathcal{C}_n$ of the set $\mathcal{C}_n$ under consideration.

More generally, consider the set of planted plane trees with d_i nodes of degree i (and thus $\ell = 1+\sum(i-2)d_i$ non-root leaves). Prefix encoding defines a bijection between this set of trees and the subset of words of $S(x_0^\ell x_1^{d_1} \dots x_k^{d_k})$ that have the Łukasiewicz property. But according to the cycle lemma, the fraction of such words of length n among words of same length in $S(x_0^\ell x_1^{d_1} \dots x_k^{d_k})$ is $1/n$. Now words on a finite alphabet with fixed proportion of letters can be encoded optimally by the so-called entropy coder. Hence prefix encoding combined with entropy encoding yields optimal coding for plane trees with a fixed proportion of nodes of each degree.

9.3.2. Conjugacy classes of trees

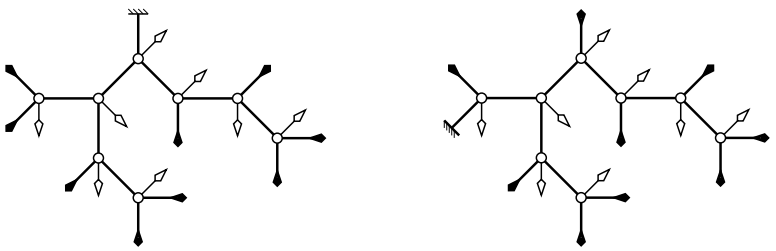
From now on, we consider planted plane trees with two types of vertices of degree 1, respectively called *buds* and *leaves*. Vertices of higher degree are called *nodes*. In particular, a *blossoming tree* is a planted plane tree such that each node has degree 4 and is adjacent to exactly one bud; a blossoming tree with n nodes has thus $n+2$ leaves and n buds. Examples of blossoming trees are given in Figure 9.18.

LEMMA 9.3.3. *The number of blossoming trees that are planted on a leaf and have n nodes is $\frac{3^n}{n+1}\binom{2n}{n}$. The number of blossoming trees that are planted on a bud and have n nodes is $\frac{3^n}{n+2}\binom{2n}{n-1}$.*

Proof. Let $\mathcal{B}'_n$ and $\mathcal{B}''_n$ denote these two sets of blossoming trees. A blossoming tree of the first type can be uniquely obtained from a binary tree with n nodes by attaching a bud to each node in one of the three possible ways. Together with Lemma 9.3.2, this proves the first formula.

Now let us consider the set of doubly planted blossoming trees, one root being a leaf and the second one a bud. Such a tree with n nodes can be considered either as a blossoming tree in $\mathcal{B}'_n$ with a marked bud, or as a blossoming tree in $\mathcal{B}''_n$ with a marked leaf. Hence doubly planted blossoming trees with n nodes are either counted by $n\,\mathrm{Card}(\mathcal{B}'_n)$ or by $(n+2)\,\mathrm{Card}(\mathcal{B}''_n)$. As a consequence, $\mathrm{Card}(\mathcal{B}''_n) = \frac{n}{n+2}\cdot\frac{3^n}{n+1}\binom{2n}{n}$, which proves the second formula. ∎

Let T be a planted plane tree with n nodes. During a contour traversal of T, its buds and leaves are visited in a sequence (by convention the root vertex is visited at the end of the traversal). Accordingly the *border word* is the word with letters $\{b,\ell\}$ obtained along the contour traversal by writing a letter b each time a bud is visited and a letter ℓ each time a leaf is visited. For example, the border



(a) A blossoming tree, (b) and a balanced one.

Figure 9.18. Two conjugate blossoming trees.

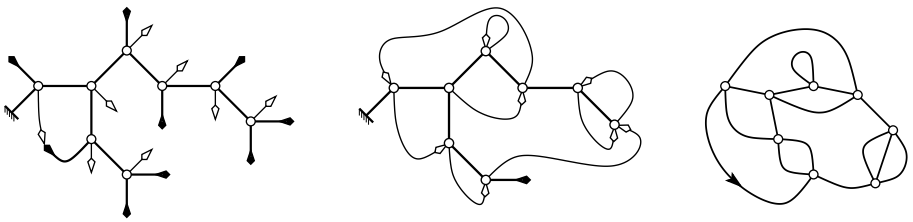
words of the blossoming trees of Figure 9.18 are respectively $\ell\ell b\ell b\ell\ell bb\ell b\ell\ell b\ell bb\ell$ and $b\ell b\ell\ell bb\ell b\ell\ell b\ell bb\ell\ell\ell$.

Two planted plane trees T and T' are *conjugate* if one is obtained from the other by re-rooting. In other terms, two planted plane trees are in the same *conjugacy class of trees* if they share the same underlying unrooted plane tree. This terminology is motivated by the remark that conjugate planted plane trees have conjugate border words. Taking $\delta(b) = +1$ and $\delta(\ell) = -1$, the cycle lemma suggests the following definition: a planted plane tree is *balanced* if its border word has the Łukasiewicz property. With this definition, and remembering that blossoming trees have two more leaves than buds, the cycle lemma for those trees reads: a blossoming tree has exactly two canonical leaves such that the conjugate trees rooted at these leaves are balanced.

LEMMA 9.3.4. *There are* $\frac{2}{n+2}\frac{3^n}{n+1}\binom{2n}{n}$ *balanced blossoming trees with* n *nodes.*

Proof. The first proof is again based on a double counting argument. Let $\mathcal{B}_n^*$ be the set of balanced blossoming trees with n nodes. The number of balanced blossoming trees with a secondary root leaf is $(n+2)\,\mathrm{Card}(\mathcal{B}_n^*)$. Upon exchanging the role of the two roots, these trees are also blossoming trees with a secondary root leaf taken among the two canonical leaves: their number is thus $2\cdot\frac{3^n}{n+1}\binom{2n}{n}$. The result follows. ∎

Proof (bis). An alternative proof is based on the following remark: the number of balanced re-rootings of any blossoming tree is equal to the difference between its numbers of leaves and buds, so that, in each conjugacy class of trees, the number of balanced trees is exactly the difference between the number of trees rooted on a leaf and the number of trees rooted on a bud. Hence the number of balanced blossoming trees with n nodes is the difference $\frac{3^n}{n+1}\binom{2n}{n} - \frac{3^n}{n+2}\binom{2n}{n-1}$. ∎



(a) A fusion, (b) the partial closure (c) and the complete closure.

Figure 9.19. The closure of the balanced blossoming tree of Figure 9.18(b).

9.3.3. The closure of a plane tree

The *closure* of a planted plane tree with two more leaves than buds is obtained by repeating the following construction until only two leaves remain: perform a contour traversal, and each time a leaf follows a bud in the sequence of vertices of degree 1 met by the walk, match them, *i.e.* fuse the two corresponding dangling edges in the unique way that creates a bounded face with no vertex of degree 1 inside (see Figure 9.19(a)).

LEMMA 9.3.5. *The closure of a plane tree with n nodes and two more leaves than buds terminates and produces a planar map with the same n nodes and two leaves, which are both incident to the infinite face. In particular the closure of a blossoming tree has n vertices of degree four, plus two of degree one in the infinite face.*

If moreover the tree is balanced, then its root vertex is one of the two remaining leaves.

Proof. At each iteration all factors $b\ell$ of the border word are detected, and deleted since the corresponding pairs of bud and leaf are matched. In particular at least one pair is matched at each iteration, so that the construction terminates. Vertices of degree at least two remain unchanged while all buds and leaves are eliminated but the two canonical roots. ∎

As described above the closure could require a quadratic number of operations. The following algorithm takes a planted plane tree with two more leaves than buds and computes its closure in linear time. It uses the following items:

- a local stack with functions PUTINSTACK(), POPFROMSTACK() and ISSTACKEMPTY(),
- a function NEXTFREEVERTEX(*vertex*) that starts a contour traversal after the vertex of degree 1 *vertex* and returns the first vertex of degree 1 found,
- a function TYPE(*vertex*) that tells whether *vertex* is a bud or a leaf,
- a function FUSEINTOEDGE(*bud*, *leaf*) that realizes the fusion of a bud *bud* and a leaf *leaf* into an edge.

```
Closure(T)
 1  n ← NumberOfLeaves(T)
 2  vertex ← RootOf(T)
 3  (ℓ1, ℓ2) ← (vertex, vertex)
 4  while n > 2 do
 5        vertex ← NextFreeVertex(vertex)
 6        if Type(vertex) = bud then
 7              PutInStack(vertex)
 8        elseif IsStackEmpty() then
 9              (ℓ1, ℓ2) ← (ℓ2, vertex)
10        else bud ← PopFromStack()
11              FuseIntoEdge(bud, vertex)
12              n ← n − 1
13  if ℓ1 = ℓ2 then
14        ℓ2 ← NextFreeVertex(vertex)
15  return (T, ℓ1, ℓ2)
```

Remark 9.3.6. Lines 13 and 14 only treat the special case of a balanced blossoming tree in which the second free leaf is the last one of the border word.

The *complete closure* of a balanced blossoming tree is obtained from its closure by fusing the two remaining vertices of degree 1 and the incident dangling edges into a root edge. Lemma 9.3.5 implies that the complete closure of a blossoming tree with n nodes is a 4-valent map with n vertices. The following more precise theorem will be proved in the next section.

Theorem 9.3.7. *The complete closure is one-to-one between balanced blossoming trees with n nodes and 4-valent maps with n vertices. In particular the number of these maps is* $\frac{2}{n+2}\frac{3^n}{n+1}\binom{2n}{n}$.

As a corollary we already have the complete description of a random sampling algorithm for 4-valent maps with n vertices. Apart from the function Closure(), it uses the random generator FlorentineRejection() defined in Section 9.2 and the following items:

- a function PrefixDecode(w) that constructs the binary tree encoded by a Dyck-Łukasiewicz word w,
- a function AddBud(n, i) that adds a bud to a node n in one of the three possible manners,
- a function AddRoot(M, ℓ_1, ℓ_2) that roots the map M by fusing its two leaves ℓ_1 and ℓ_2 into an oriented edge.

```
RandMap(n)
 1  w ← FlorentineRejection(n, 0)
 2  T ← PrefixDecode(wd)
 3  for node ∈ T do
 4        AddBud(node, Rand(1, 3))
 5  (M, ℓ1, ℓ2) ← Closure(T)
 6  if Rand(1, 2) = 1 then
 7        AddRoot(M, ℓ1, ℓ2)
 8  else AddRoot(M, ℓ2, ℓ1)
 9  return M
```

COROLLARY 9.3.8. RandMap(n) *outputs a uniform random 4-valent map with n vertices in linear time.*

9.3.4. The opening of a 4-valent map

The *dual* of a planar map M is the planar map M^* defined as follows: in each face of M put a vertex, and join these new vertices by edges dual to the edges of M. By construction the vertices, edges and faces of M^* are respectively in bijection with faces, edges and vertices of M. This construction is illustrated by Figure 9.20(a). The proof of the following property of duality in planar maps is left to the reader.

LEMMA 9.3.9. *Let (E_1, E_2) be a partition of the set of edges of a planar map M. Then E_1 is a spanning tree of M if and only if E_2^* is a spanning tree of M^*. When this case we call (E_1, E_2) a* spanning tree decomposition *of M.*

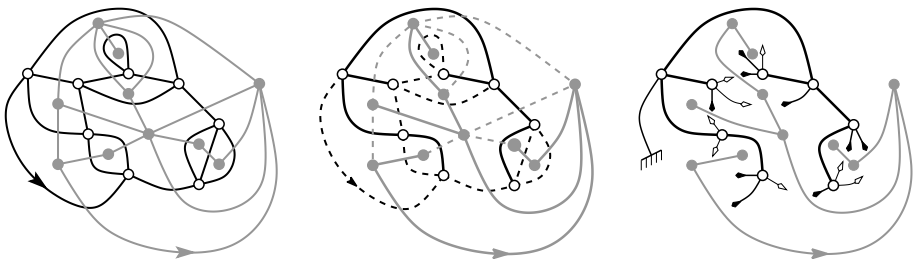
From now on, let M be a planar map, and (E_1, E_2) be a spanning tree decomposition of M. For e an edge of E_2, *opening e with respect to* (E_1, E_2) will mean: orienting e so that the cycle it induces with the tree E_1 is counterclockwise, and then replacing e by two dangling edges, the one attached to the origin of e holding a bud $b(e)$, the other one holding a leaf $\ell(e)$. We shall always assume moreover that the root r of M belongs to E_2. Then, the *opening of M with respect to* (E_1, E_2) is the tree T defined as follows: (see Figure 9.20(c))

- open each edge $e \in E_2$ with respect to (E_1, E_2),
- replace the bud $b(r)$ by a leaf and plant the tree on it.

The tree T thus consists of the edges of the spanning tree E_1 together with pairs of dangling edges associated to edges of E_2. More precisely, these edges contribute to one bud and one leaf except for the root which contributes to two leaves. By construction, the opening T of a 4-valent planar map M with n vertices has n nodes of degree 4, n buds and $n + 2$ leaves.

LEMMA 9.3.10. *The complete closure of the opening of a planar map M with respect to any spanning tree decomposition is the planar map M itself.*

Proof. The opening of an edge merges the two faces incident to it. Since E_2^* forms a spanning tree of M^*, the openings can be performed sequentially so that



(a) A map and its dual, (b) a spanning tree decomposition, (c) and the corresponding opening.

Figure 9.20. An opening of the map of Figure 9.19(c).

one of the two merged faces is always the infinite face. It is then immediate at each step that the pair of bud and leaf created by the opening of an edge corresponds to a matched pair in the closure. ∎

There are in general many spanning tree decompositions of M, and the right one must be chosen to invert the closure. To explain how this is done we need to introduce the distance in the dual map M^*: two faces of M are adjacent if they share a common edge, and the distance between two faces f and f' is the length k of the shortest path $(f_0, \dots, f_k)$ where $f_0 = f$, $f_k = f'$ and for all i, the two faces f_i and f_{i-1} are adjacent. Observe that the dual M^* of a 4-valent map has only faces with even degrees (in fact degree 4), so that it does not contain any cycle of odd length, and the distances of a face f to two adjacent faces f' and f'' always differ by 1.

To each face f of M, associate the face $r(f)$ incident to the root edge r and closest to f for the distance in M^*. The set $\mathcal{P}(f)$ of paths of minimal length from $r(f)$ to f forms a bundle of paths bounded by two paths $P_0(f)$ and $P_1(f)$, with $P_0(f)$ having the bundle on its right hand side. We shall call $P_0(f)$ the *leftmost minimal path from the root to f*. The union of r^* and of the edges of the paths $P_0(f)$ for all faces f of M forms a spanning tree of M^*: the existence of a cycle would prevent one of the paths from being leftmost. This tree is called the *leftmost breadth first search tree of M^* starting from r^**, because it is also given by a breadth first search traversal with the left hand rule. As stated in the following proposition, it is the spanning tree we are looking for.

Proposition 9.3.11. *Let M be a 4-valent map with root edge r and (E_1, E_2) be a spanning tree decomposition such that $r \in E_2$. Then the opening of M with respect to (E_1, E_2) is a blossoming tree if and only if E_2^* is the leftmost breadth first search tree of M^* starting from r^*.*

The proof of this proposition is based on two lemmas. The first one is a characterization of blossoming trees.

LEMMA 9.3.12. *A tree T with n buds, $n+2$ leaves and n nodes of degree 4 is a blossoming tree if and only if, for every inner edge e, the two components of $T \setminus e$ both contain one more leave than buds.*

Proof. The characterization is trivial for $n = 1$, and remains true when a further node with two leaves and a bud is attached in place of a leaf. The lemma thus follows by induction since every tree can be obtained by adding new nodes incrementally. ∎

For the second lemma it is useful to view the spanning tree E_2^* as rooted on r^*, with the convention that the infinite face of M is the origin of the root.

LEMMA 9.3.13. *Let e be an edge of E_1 separating two faces f, f', with f before f' in the leftmost depth first order on the tree E_2^*. Consider the paths P and P' from f and f' to their common ancestor in E_2^*, which define with e^* a cycle separating a bounded region B of the plane from an unbounded one U. Then,*

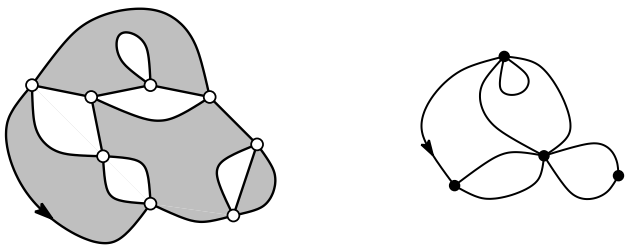
- *the opening of an edge of P with respect to (E_1, E_2) creates a leaf in B and a bud in U,*
- *and the opening of an edge of P' with respect to (E_1, E_2) creates a bud in B and a leaf in U.*

Proof. The result is immediate upon comparing the orientation used in the definition of the opening of an edge and the orientation of the cycle going from e^* up the path P and down the path P'. ∎

Proof of Proposition 9.3.11. First assume that E_2^* is the leftmost breadth first search tree of M^* starting from r^*, and let T be the opening of M with respect to E_2^*. According to Lemma 9.3.12, it suffices to check that for any edge e of E_1, both components of $T \setminus e$ contain one more leaves than buds. Let us consider the paths P and P' of Lemma 9.3.13. The breadth first search condition on E_2^* implies that the length of these two paths differ at most by 1, hence exactly by 1, in view of the discussion of distances in M^*. The leftmost condition on E_2^* moreover implies that the shortest path of the two must be P'. Finally observe that two components of $T \setminus e$ are separated by the dual cycle of Lemma 9.3.13, so that this lemma can be used to count buds and leaves in the two regions. This can be done easily upon distinguishing whether r^* is on P or not.

Let now E_2^* be a spanning tree of M^* different from the leftmost breadth first search tree $E_2'^*$. Then there are leftmost minimal paths that do not appear in E_2^*. Among the shortest of them let $P_0(f)$ be the leftmost one, connecting the root to a face f. Since $P_0(f)$ is minimal, all its edges but the last one e belong to E_2^*. Moreover, by definition of $P_0(f)$, this path is to the left and no longer than the path $P(f)$ connecting the root to f in E_2^*. Applying Lemma 9.3.13 to e, $P \subset P_0(f)$ and $P' \subset P(f)$ and comparing the length of these two paths shows that P' is longer than P, so that the two components of $T \setminus e$ have not the expected number of buds and leaves. ∎

The opening of M with respect to (E_1, E_2) with E_2^* the leftmost breadth first search tree of M^* at r^* will be called simply the *opening of* M. In view



(a) Bicoloration of faces. (b) Corresponding map.

Figure 9.21. Inverse of the edge-map construction.

of Lemma 9.3.10, Proposition 9.3.11 completes the proof of Theorem 9.3.7: the opening is the inverse of the closure. Moreover it induces a linear time algorithm OPENING(M) that recovers the unique balanced blossoming tree T such that CLOSURE(T)$= M$:

OPENING(M)
1 Perform a leftmost bfs traversal of the dual map M^* starting from r^*.
2 Open the edges of the resulting tree to create buds and leaves.
3 Return the resulting balanced blossom tree.

9.3.5. A code for planar maps

Theorem 9.3.7 deals with a specific family of planar maps, namely 4-valent ones. It turns out however that 4-valent maps play for planar maps the role that edge-graph play for graphs. More precisely, define the *edge-map* of a planar map M as the 4-valent map M^{φ} having as vertex set the set of edges of M and having an edge c^{φ} for each corner c of the map M.

PROPOSITION 9.3.14. *The edge-map construction is a bijection between planar maps with n edges and 4-valent maps with n vertices. In particular the number of planar maps with n edges is* $\frac{2}{n+2}\frac{3^n}{n+1}\binom{2n}{n}$.

Proof. The inverse construction follows from the remark that the faces of a 4-valent map F can be colored in two colors, black and white, so that adjacent faces have different colors. The planar map M is obtained by putting a vertex into each black face of F and joining these vertices by an edge across each vertex of F. ∎

The edge-map construction thus allows us to deduce from Theorem 9.3.7 a code for the family of planar map.

```
EncodeMap(M)
 1  F ← EdgeMap(M)
 2  T ← Opening(F)
 3  for node ∈ T do
 4       w'[node] ← PositionOfBud(node)
 5       T ← RemoveBud(T)
 6  w ← BinaryCode(T)
 7  return (w, w')
```

Theorem 9.3.15. *The algorithm* EncodeMap() *encodes a planar map with n edges by a pair of words respectively in $\{a,b\}^{2n}$ and $\{0,1,2\}^n$. In view of the number of planar maps, this code is optimal.*

Problems

Section 9.1

9.1.1 Show that the generating function of a rational language with respect to the length is rational.

9.1.2 Compute the generating function with respect to the length of walks that never immediately undo a step they have just done.

9.1.3 Define the *area under a Dyck word* as the number of integer points between the horizontal axis and the associated walk. Use Catalan's factorization to show that the sum of the area under all Dyck words of length $2n$ is 4^n.

(Chottin and Cori 1982)

9.1.4 Show that an algebraic language that can be generated by a non ambiguous context free grammar has an algebraic generating function with respect to the length.

9.1.5 Give a bijective proof of the fact that the number of bicolored Motzkin words of length n is equal to the number of Dyck words of length $2n+2$.

9.1.6 Give a bijective proof of the right hand side formula in Proposition 9.1.9 for the number of loops of length $2n$ that stay in the quadrant ($x \geq 0, y \geq 0$).

(Guy et al. 1992)

Section 9.2

9.2.1 What is the number of staircase and unimodal polygons with semi-perimeter n?

*9.2.2 Show bijectively that the number of convex polyominoes with bounding box (p,q) is

$$\binom{2p+2q}{2p} + q\binom{2p+2q-1}{2p-1} - 2(p+q)\binom{p+q-1}{q}\binom{p+q-1}{p}.$$

What is the number of convex polyominoes with semi-perimeter n?
(Bousquet-Mélou and Guttmann 1997,Gessel 2000)

9.2.3 An animal on the square lattice has compact source if there exists k such that every vertex of the animal can be reached from one of the vertices $(i, k-i)$ with $0 \leq i \leq k$ by a path going north or east inside the animal. In particular directed animals are exactly the animals with compact source for $k = 0$.
Prove that there are 3^{n-1} animals of size n with compact source.
(Gouyou-Beauchamps and Viennot 1988)

*9.2.4 Give a bijection between bilateral Dyck paths of length n and (non necessarily strict) pyramids of n bricks such that the number of pairs of steps connecting levels i and $i+1$ is mapped onto the number of bricks in position $(i, i+1)$.
(Viennot 1986)

**9.2.5 Give a uniform random sampling algorithm of expected linear complexity for the set of words of length n on an arbitrary fixed finite alphabet that have the Łukasiewicz property.

Section 9.3

9.3.1 Give a direct bijection between plane trees with n edges and binary trees with n nodes.

*9.3.2 What is the number of rooted planar maps with d_i vertices of degree $2i$ for all $i \geq 0$ and no odd degree vertex?
(Schaeffer 1997)

**9.3.3 Compute the generating function of rooted planar maps according to the distribution of degrees.
(Bouttier et al. 2002)

**9.3.4 Show that planted plane trees with two leaves per inner vertices are in one-to-one correspondence with rooted triangulations with a marked face.
(Poulalhon and Schaeffer 2003)

Notes

Although this chapter can be read independently, it is intended as a companion to Chapter 11, Words and trees, in Lothaire 1997. Systematic approaches to enumeration, in particular using generating functions, are described in the books Goulden and Jackson 1983, Bergeron et al. 1998 and in the more recent Stanley 1999, Flajolet and Sedgewick 2002. In particular the relevance of rational, algebraic and D-finite series to enumeration is emphasized in the last two ones.

The enumeration of walks in the plane, in the half plane and in the quarter plane has become part of the combinatorial folklore, as well as Dyck walks and Catalan's factorization. The cycle lemma is attributed in the combinatorial

literature to Dvoretzky and Motzkin 1947, where it is used to derive Proposition 9.1.4. As first shown by Raney 1960 (see also Chapter 11 of Lothaire 1997), the cycle lemma is a combinatorial version of the Lagrange inversion formula, which has numerous applications in enumerative combinatorics. More detailed historical accounts can be found in Pitman 1998 and Stanley 1999.

The classification of the possible asymptotic behaviors of the Taylor coefficients of an algebraic series can be found in Flajolet 1987. The generating function of walks on the slitplane according to the length and the coordinates of the extremities was first shown algebraic and computed in Bousquet-Mélou and Schaeffer 2002. This is one in a series of results obtained recently by writing and solving linear equations with catalytic variables, see Banderier and Flajolet 2002, Bousquet-Mélou 2002 (these references are also good entry points to the literature on counting walks on lattices). The first proof we present illustrates a very general approach developed in Bousquet-Mélou 2001. The second proof is taken from Barcucci et al. 2001.

The foundation of combinatorial random generation was laid in Nijenhuis and Wilf 1978 with the recursive method. As shown in Flajolet et al. 1994, this approach leads systematically to polynomial algorithms for decomposable combinatorial structures. The (much more specialized) application of the cycle lemma to random generation is discussed in Dershowitz and Zaks 1990 and Alonso et al. 1997. The Florentine rejection algorithm is taken from Barcucci et al. 1995. A systematic utilisation of mixed probabilistic/combinatorial arguments for sampling was recently proposed in Duchon et al. 2002.

General references on polyominoes are Klarner 1997, van Rensburg 2000. Exact enumerative results are surveyed in Bousquet-Mélou 1996. The algorithms to sample convex and directed convex polyominoes are adapted from Hochstättler et al. 1996 and Del Lungo et al. 2001. From the enumeration point of view, these results are encompassed by Bousquet-Mélou and Guttmann 1997, which deals with convex polygons in any dimension. Our treatment of directed animals and heaps of bricks is adapted from Bétréma and Penaud 1993. These results built on the combinatorial intepretation of the commutation monoïd of Cartier and Foata 1969 in terms of heaps of pieces due to Viennot 1986.

Starting from the seminal work of Tutte 1962, the literature on combinatorial maps has grown almost independently in combinatorics and in physics. Some surveys are Cori and Machì 1992 (combinatorial point of view), Ambjørn et al. 1997 (physical point of view) and Di Francesco 2001 (mixed points of view).

A more detailed description of codes for plane trees appear in Chapter 11 of Lothaire 1997. The idea to use algebraic languages to encode maps already appeared in Cori 1975, and plane trees are explicitly used in Cori and Vauquelin 1981. Conjugacy classes of trees were introduced in Schaeffer 1997, as well as the bijection between balanced trees and planar maps. Applications to coding and sampling are discussed in Poulalhon and Schaeffer 2003.