Università degli studi di Ferrara

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Matematica

Modelli replicativi basati su sistemi di Lindenmayer e trasformazioni di Möbius

Relatore:
Chiar.mo Prof.
Josef Eschgfäller

Laureanda:
Sandra Pareschi

Anno Accademico
2006 - 2007

Indice

Introduzione

Questa tesi contiene alcuni programmi in Python per la rappresentazione grafica di sistemi di Lindenmayer e, nell'ultimo capitolo, alcuni esperimenti con un nuovo tipo di rappresentazione che potrebbe essere interessante nelle applicazioni alla modellizzazione della crescita di colture cellulari e tessuti o della propagazione di epidemie.

Nel primo capitolo vengono descritte alcune proprietà del monoide libero A^* generato da un alfabeto A. Dopo aver richiamato alcuni classici risultati sull'immergibilità di un semigruppo in un gruppo, si dimostra che il monoide libero A^* soddisfa le condizioni di Maltsev ed è quindi sottomonoide di un gruppo.

Nel secondo capitolo un sistema di Lindenmayer è definito come tripla (A, φ, Ω), in cui φ è un endomorfismo del monoide A^* ed Ω una parola del monoide detta *elemento iniziatore*. Un endomorfismo di A^* può essere considerato come un meccanismo di riscrittura, che parte dall'elemento iniziatore, al quale si sostituisce, ripetutamente, l'immagine dell'ultima parola ottenuta tramite l'endomorfismo φ. Una prima realizzazione dei sistemi di Lindenmayer in Python è molto semplice e richiede soltanto due funzioni che definiscono un dizionario con le regole del sistema e una lista di stringhe i cui elementi vengono riscritti secondo le regole definite. Come primi esempi di sistemi di Lindenmayer vengono presentati la successione di Morse e la successione di Cantor.

Nel terzo capitolo discutiamo prima alcuni aspetti teorici riguardanti parole bilanciate, sui quali si basa l'uso di parentesi nella rappresentazione di strutture ramificate tramite sistemi di Lindenmayer. Lavorando direttamente con liste annidate di funzioni, possiamo però realizzare l'elaborazione ramificata in modo molto più semplice ed efficiente.

Con questa tecnica nel quarto capitolo possiamo realizzare la rappresentazione grafica di alcune figure classiche mediante il metodo della tartaruga.

Nel quinto capitolo vengono discusse brevemente le trasformazioni di Möbius, in particolare gli automorfismi olomorfi del disco unitario e le trasformazioni di Möbius reali che trasformano l'intervallo unitario in sè stesso.

Il sesto capitolo contiene alcune nuove idee per la rappresentazione di culture cellulari o della propagazione di epidemie tramite sistemi di Lindenmayer, in cui si utilizzano trasformazioni di Möbius del disco unitario e trasformazioni affini e quadratiche del piano.

1. Il monoide A^*

Situazione 1.1. Sia A un insieme. A è considerato come l'alfabeto che usiamo nelle parole che vogliamo generare. Gli elementi dell'insieme A si chiamano le *lettere* di questo alfabeto.

Definizione 1.2. Con A^+ denotiamo l'insieme delle sequenze finite (dette in questo contesto *parole*) di lunghezza ≥ 1 formate con lettere appartenenti ad A. Introducendo la parola vuota ε (di lunghezza 0), poniamo $A^* := A^+ \cup \{\varepsilon\}$. Formalmente $A^* = \bigcup_{n=0}^{\infty} A^n$ ed $A^+ = \bigcup_{n=1}^{\infty} A^n$.

Con $A^0 = \emptyset = \varepsilon$ ovvero $A^* = A^+ \cup A^0$. Se $x \in A^n$, diciamo che x possiede lunghezza n e scriviamo $|x| := n$.

In Python gli elementi di A^* verranno rappresentati come stringhe.

Nota 1.3. A^* diventa un monoide (un semigruppo dotato di elemento neutro che in questo caso coincide con ε) se come prodotto di due parole utilizziamo la loro naturale concatenazione.

Nella teoria di A^* una sequenza $(a_1, \ldots, a_n)$ viene spesso scritta nella forma $a_1 \ldots a_n$, tralasciando le parentesi.

Il prodotto è quindi definito da

$$(a_1 \ldots a_n)(b_1 \ldots b_m) := a_1 \ldots a_n b_1 \ldots b_m$$

È anche chiaro a questo punto che $a_1 \ldots a_n$ è proprio il prodotto delle lettere $a_1, \ldots, a_n$, considerate come elementi di A^*.

Per $X, Y \subset A^*$ poniamo $XY := \{xy \mid x \in X,\ y \in Y\}$. È chiaro che $A^n = A \cdots A$.

Il monoide A^* si chiama il *monoide libero* sull'alfabeto A ed è altamente non commutativo; il semigruppo A^+ si chiama il *semigruppo libero sull'alfabeto*. Nell'informatica teorica i sottoinsiemi di A^* sono detti linguaggi formali.

Definizione 1.4. Sia S un semigruppo. Per un sottoinsieme $X \subset S$ denotiamo con $\mathrm{semigruppo}(X)$ il semigruppo generato da X e, nel caso S sia un monoide, con $\mathrm{monoide}(X)$ il monoide generato da X. Quindi

$$\mathrm{semigruppo}(X) = \{x_1 \ldots x_n \mid n \geq 1 \text{ e } x_1, \ldots, x_n \in X\},$$

e $\mathrm{monoide}(X) = \mathrm{semigruppo}(X) \cup \{1_S\}$.

Definizione 1.5. (1) S e T siano semigruppi e $\varphi : S \to T$ un'applicazione. φ si chiama un *omomorfismo di semigruppi* se $\varphi(xy) = \varphi(x)\varphi(y)$ per ogni $x, y \in S$.

(2) Siano S e T monoidi con elementi neutri 1_S e 1_T. Un'applicazione $\varphi : S \to T$ si chiama un omomorfismo di monoidi se φ è un omomorfismo di semigruppi e se $\varphi(1_S) = 1_T$. Si parla di un endomorfismo se $S = T$.

Osservazione 1.6. S sia un monoide, T un semigruppo e $\varphi : S \to T$ un omomorfismo di semigruppi.

(1) Se φ suriettivo, allora T è un monoide con elemento neutro $\varphi(1_S)$. φ è quindi anche un omomorfismo di monoidi.

(2) Se invece φ è iniettivo e T un monoide, φ non è necessariamente un omomorfismo di monoidi; può cioè accadere che $\varphi(1_S) \neq 1_T$.

Dimostrazione. (1) φ è suriettivo. Sia $t \in T$. Siccome φ è suriettivo esiste un $s \in S$ tale che $t = \varphi(S)$ quindi $\varphi(1_S)t = \varphi(1_S)\varphi(S) = \varphi(1_S.S) = \varphi(S) = t$.

(2) Siano $S := (0, .)$ e $T := (0, 1, .)$. L'iniezione $\varphi : S \to T$ è iniettiva, ma $\varphi(1_S) = \varphi(0) = 0 \neq 1 = 1_T$.

Teorema 1.7. *T sia un monoide e $\varphi_0 : A \to T$ un'applicazione qualsiasi. Allora esiste un unico omomorfismo di monoidi $\varphi : A^* \to T$ che su A coincide con φ_0.*

Dimostrazione. (1) È chiaro che φ deve avere la proprietà che

$\varphi(a_1 \dots a_n) = \varphi(a_1) \dots \varphi(a_n) = \varphi_0(a_1) \dots \varphi_0(a_n)$ per $a_1 \dots a_n \in A$.

Questo è quindi l'unico modo possibile per definire φ.

(2) Bisogna però dimostrare che funziona, cioè che φ in questo modo risulta ben definito. Ma ciò segue dal fatto che ogni parola $x \in A^+$ ha un'unica rappresentazione della forma $x = a_1 \dots a_n$ con $a_1 \dots a_n \in A$.

(3) Dimostriamo che φ è veramente un omomorfismo di monoidi. Per definizione vale $\varphi(\varepsilon) = 1_T$. Siano $x = a_1 \dots a_n, y = b_1 \dots b_m$ con $n, m \geq 1$ e $a_i, b_j \in A$. Allora

$\varphi(xy) = \varphi(a_1 \dots a_n b_1 \dots b_m) = \varphi_0(a_1) \dots \varphi_0(a_n)\varphi_0(b_1) \dots \varphi_0(b_m) = \varphi(x)\varphi(y)$.

Teorema 1.8. *T sia un semigruppo ed $A \subset T$ con* $\text{semigruppo}(A) = T$. *Allora esiste un omomorfismo di semigruppi suriettivo $\varphi : A^* \to T$. Se T è un monoide, φ può essere scelto come omomorfismo di monoidi.*

Dimostrazione. Clifford/Preston, 117-118.

Corollario 1.9. *Ogni semigruppo è immagine omomorfa di un semigruppo libero e ogni monoide è immagine omomorfa di un monoide libero.*

Dimostrazione. Nel Teorema 1.8 possiamo scegliere A = T.

Definizione 1.10. Un semigruppo S si dice *cancellativo* se, per $x, a, b \in S$ si hanno sempre le implicazioni:

$$xa = xb \quad \Rightarrow a = b$$

$$ax = bx \quad \Rightarrow a = b$$

Osservazione 1.11. A^* è cancellativo.

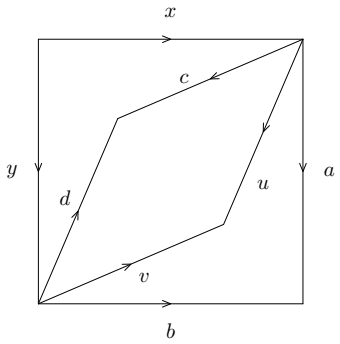
Osservazione 1.12. Un sottosemigruppo di un gruppo è sempre cancellativo.

Proposizione 1.13. *Un semigruppo commutativo è cancellativo se e solo se è sottosemigruppo di un gruppo.*

Dimostrazione. Corso di Algebra. Per trovare il gruppo si usa la stessa tecnica come quando si passa da $\mathbb{N}$ a $\mathbb{Z}$ (per l'addizione) o da $\mathbb{Z}$ a $\mathbb{Q}$ (più precisamente da $\mathbb{Z} \setminus 0$ a $\mathbb{Q} \setminus 0$, per la moltiplicazione).

Definizione 1.14. Un semigruppo S soddisfa le *condizioni di Maltsev* se, per $x, y, u, v, a, b, c, d \in S$ si hanno sempre le implicazioni

$$xa = xb,\ xc = yd,\ ua = vb \ \Rightarrow uc = vd$$



La condizione richiede che nella figura anche il quadrangolo interno sia commutativo.

Teorema 1.15 (teorema di Maltsev). *Un semigruppo è sottosemigruppo di un gruppo se e solo se soddisfa le condizioni di Maltsev.*

Dimostrazione. La dimostrazione, molto difficile, si trova in Clifford/Preston, 297, una recente esposizione moderna nella tesi di Alan Cain del 2005.

Osservazione 1.16. Maltsev ha dato esempi di semigruppi cancellativi che non soddisfano le condizioni della def. 1.14 e quindi non sono sottosemigruppi di un gruppo.

Nota 1.17. S e T siano monoidi. Diciamo che S è un *sottomonoide* di T se S è un sottosemigruppo di T e se in più $1_S = 1_T$. La seconda condizione non segue dalla prima, come mostra l'esempio $S = \{0\}$ e $T = \{0, 1\}$ già visto nella dimostrazione dell'osservazione 1.6. Se un monoide S è invece sottosemigruppo di un gruppo G, allora S è anche sottomonoide di G. Infatti 1_S è un elemento idempotente di G, ma un gruppo contiene un solo idempotente e ciò implica $1_S = 1_G$.

Teorema 1.18. *Il monoide libero A* è sottomonoide di un gruppo.*

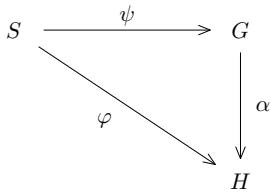
Dimostrazione. Verifichiamo le condizioni di Maltsev.

Siano $x, y, u, v, a, b, c, d \in A^*$ tali che $xa = yb$, $xc = yd$, $ua = vb$. Dobbiamo dimostrare che $uc = vd$. Per simmetria possiamo osservare che $|x| \leq |y|$. L'equazione $xa = yb$ implica allora che esiste $s \in A^*$ tale che $y = xs$. Abbiamo così $xa = xsb$ e $xc = xsd$ e quindi $a = sb, c = sd$. Però $ua = vb$ e quindi $usb = vb$, per cui $us = v$. Ciò implica $uc = usd = vd$.

Nota 1.19. Si può dimostrare (Clifford/Preston, 290-292) che per ogni sottogruppo S esistono un gruppo G (detto *gruppo universale* di S) e un omomorfismo di sottogruppi $\psi : S \to G$ con le seguenti proprietà:

(1) $\text{semigruppo}(\psi(S)) = G$

(2) Per ogni gruppo H ed ogni omomorfismo di sottogruppi $\varphi : S \to H$ per cui $\text{semigruppo}(\varphi(S)) = H$ esiste un (necessariamente unico) omomorfismo di gruppi $\alpha : G \to H$ che rende commutativo il diagramma



L'unicità di α segue dalla condizione (1).

Proposizione 1.20. *S sia un semigruppo e $\psi : S \to G$ come nella nota 1.19. Se S è sottosemigruppo di un gruppo, allora ψ è iniettivo.*
S può quindi in questo caso essere considerato come sottosemigruppo del proprio gruppo universale.

Dimostrazione. Clifford/Preston, 292

Definizione 1.21. Un semigruppo S si dice *finitamente generato* se esiste un sottoinsieme finito $X \subset S$ tale che $\text{semigruppo}(X) = S$.

Proposizione 1.22. *Un semigruppo è sottosemigruppo di un gruppo se e solo se ogni suo sottosemigruppo finitamente generato è sottosemigruppo di un gruppo.*

Dimostrazione. Clifford/Preston, 293.

2. Sistemi di Lindenmayer

Situazione 2.1. A sia un insieme.

Definizione 2.2. Un sistema di Lindenmayer definito su A è una tripla ordinata (A, φ, Ω), in cui $\varphi : A^* \to A^*$ è un endomorfismo di monoidi ed $\Omega \in A^*$ è una parola detta *elemento iniziatore*. Il sistema si dice finito se A è un insieme finito.

Proposizione 2.3. *X sia un sottoinsieme di A^* tale che* $\mathrm{monoide}(X) = A^*$. *Allora $A \subset X$. A è quindi il più piccolo generatore di A^*.*

Dimostrazione. Assumiano che $A \not\subset X$. Allora esiste $a \in A \setminus X$. Ciò implica $a \neq \varepsilon$ (perchè ε non è mai elemento dell'alfabeto A), quindi per ipotesi esistono $x_1 \dots x_m \in X$ tali che $a = x_1 \dots x_m$; inoltre necessariamente almeno uno degli x_j dev'essere $\neq \varepsilon$, ad esempio $x_1 \neq \varepsilon$. Perciò $|x_1| \geq 1$, mentre $1 = |a| = |x_1| + \ldots + |x_m|$, cosicché $|x_2| = \ldots = |x_m| = 0$. Ciò implica $a = x_1 \in X$, una contraddizione.

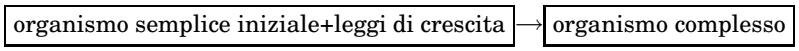
Corollario 2.4. *B sia un altro insieme tale che $A^* = B^*$. Allora $A = B$.*

Dimostrazione. È sufficiente dimostrare che $A^* = \mathrm{monoide}(A), B^* = \mathrm{monoide}(B)$. Dalla prop. 2.3 abbiamo $B \subset A$ ed $A \subset B$.

Osservazione 2.5. (A, φ, Ω) sia un sistema di Lindenmayer. Per il corollario 2.4 A è univocamente determinata da A^* e quindi anche l'applicazione $\varphi_{|A} : A \to A^*$. Se viceversa è data una qualsiasi applicazione $\varphi_0 : A \to A^*$, per il teorema 1.7, è definito un unico omomorfismo di monoidi $\varphi : A^* \to A^*$ tale che $\varphi_0 = \varphi_{|A}$.

Per questa ragione per descrivere un sistema di Lindenmayer su A è sufficiente indicare la restrizione $\varphi_{|A} : A \to A^*$ e l'elemento iniziatore Ω, come faremo quasi sempre nel seguito.

Nota 2.6. Un endomorfismo di A* può essere considerato come un *meccanismo di riscrittura*; l'idea è di imitare un principio generale della natura:

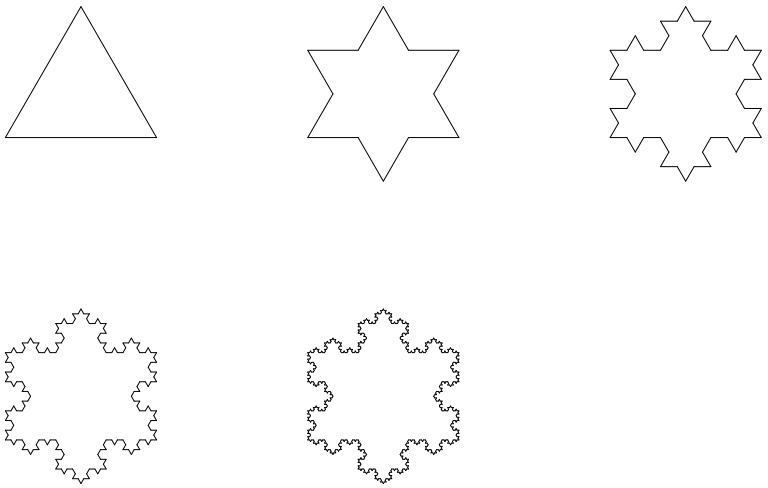


Aristid Lindenmayer (1925-1989) era un botanico olandese che utilizzò questi sistemi per descrivere (soprattutto in modo grafico) ed analizzare l'accrescimento di piante. Un bellissimo resoconto dei risultati di queste ricerche si trova nel libro *The algorithmic beauty of plants* scritto poco prima di morire con P. Prusinkiewicz.

Un semplicissimo, quanto antico, esempio di riscrittura è il fiocco di neve di Koch (1905). Si parte da un elemento *iniziatore*, il triangolo equilatero, poi si sostituisce ogni lato del triangolo con una linea spezzata orientata, l'elemento *generatore* formata da quattro tratti di lunghezza uguale. Si ripete poi questa regola di riscrittura per ogni tratto dell'oggetto che viene sostituito con la linea spezzata; ogni volta che si riapplica la regola il numero di segmenti che compongono il

nuovo oggetto quadruplica (e se si vuole che lo spazio occupato dalla figura rimanga lo stesso la linea spezzata deve essere adeguatamente rimpicciolita, in modo tale da avere gli estremi coincidenti con quelli del segmento da sostituire).

Iterando questo procedimento si perviene ad un'immagine che assomiglia ad un fiocco di neve.



Si vede che dopo pochi passaggi le figure cambiano in modo sempre meno percepibile; si può dimostrare che esse convergono ad una figura limite nella *metrica di Hausdorff* definita sull'insieme dei sottoinsiemi compatti non vuoti di $\mathbb{R}^2$ nel modo seguente:

$$d(A,B) := \max(\max\{\, d(a,B) \mid a \in A\}, \max\{\, d(A,b) \mid b \in B\})$$

È importante sottolineare che nei sistemi di Lindenmayer la riscrittura avviene in parallelo, cioè le regole vengono applicate simultaneamente ad ogni carattere di una data parola, a differenza da quanto accade nei linguaggi di Chomsky (usati spesso per descrivere i linguaggi di programmazione).

Nota 2.7. Elenchiamo alcune delle più popolari applicazioni dei sistemi di Lindenmayer:

1. Sistemi di Lindenmayer possono essere impiegati per simulare l'accrescimento di un organismo o di un intero sistema ecologico e per analizzarne i meccanismi di crescita. Si possono così individuare i parametri che determinano l'evoluzione di un organismo o di un ecosistema.

2. Due campi dove più intensamente si impiegano piante virtuali sono il cinema e i giochi al calcolatore, dove vengono usate in scene esterne, in effetti speciali, nelle simulazioni di paesaggi che possono essere esplorati interattivamente.

3. Sistemi di Lindenmayer possono essere usati per la memorizzazione economica di immagini. Infatti, invece di dover conservare tutto il contenuto di una parte intera dello schermo (ad esempio 600x600 pixel = 360000 bit = 45000 byte per un'immagine in

bianconero) è sufficiente conservare la stringa che rappresenta l'iniziatore (ad esempio 50 byte) e le stringhe che contengono le leggi di crescita (ad esempio 20x30 byte = 600 byte), quindi in tutto 650 invece di 45000 byte.

Osservazione 2.8. Sistemi di Lindenmayer forniscono un metodo matematico molto elegante per produrre alcune figure frattali classiche come il fiocco di neve di Koch, le curve che riempiono lo spazio di Hilbert e Peano, e altre. Queste figure possiedono la proprietà dell'autosimilitudine: ingrandendo un qualsiasi tratto della curva si scopre un insieme di particolari altrettanto ricco e complesso del precedente; questo procedimento può proseguire all'infinito.

Nota 2.9. *La successione di Morse.*

Questa successione è forse il più noto esempio di un sistema di Lindenmayer. Essa compare sotto molte vesti nella *dinamica simbolica* (lo studio delle periodicità e quasiperiodicità di parole infinite, cioè di elementi di $A^{\mathbb{N}}$ o $A^{\mathbb{Z}}$). Infatti la successione di Morse è la più semplice successione quasiperiodica, ma non periodica. Essa è definita nel modo seguente:

$$A = \{0, 1\}$$

$$iniziatore : 0$$

$$\varphi_{|A} : 0 \longmapsto 01,\ 1 \longmapsto 10$$

Quindi la successione si sviluppa in questo modo:

$$0$$

$$01$$

$$0110$$

$$01101001$$

$$0110100110010110$$

$$\dots$$

Si vede che la successione può essere generata anche in altri modi, ad esempio aggiungendo alla successione ottenuta al passo precedente la successione che si ottiene da essa scambiando 1 con 0. Vediamo in particolare che la successione si allunga sempre senza mai cambiare nelle parti costruite negli stadi precedenti.

Definizione 2.10. Un sistema di Lindenmayer (A, φ, Ω) si dice *propagante*, se $\varphi(a) \neq \varepsilon \ \forall \ a \in A$. È chiaro che ciò accade se e solo se $|\varphi(x)| \geq |x|$ per ogni $x \in A^*$.

Osservazione 2.11. Nei programmi useremo come lettere dell'alfabeto stringhe in Python che non contengono spazi bianchi; per questo mediante la funzione *Lista* della Nota 2.12 una stringa che contiene spazi bianchi viene trasformata in una lista di stringhe:

```
'a  b3  axy  0'----> ['a','b3','axy','0']
```

Nota 2.12. La realizzazione di sistemi di Lindenmayer in Python è molto semplice e richiede soltanto le seguenti due funzioni, di cui indichiamo prima i prototipi:

*Linden(r,a):*Il primo argomento è un dizionario che contiene le regole del sistema. Il secondo argomento è una lista di stringhe oppure una stringa che viene trasformata, tramite il metodo *split*, in una lista di stringhe. Ogni elemento di questa lista viene poi riscritto secondo le regole rapportate dal dizionario r; il risultato restituito dalla funzione è la lista di stringhe che in questo modo si ottiene tramite concatenazione proprio come nella dimostrazione del teorema 1.7.

Stringa(v,larg): Questa funzione viene usata nell'output.Il primo argomento è una lista di stringhe, il secondo la larghezza che si vuole impostare. Gli elementi della lista v vengono concatenati interponendo degli spazi o dei caratteri di nuova riga in modo che non venga superata la larghezza indicata. Questa funzione può essere utilizzata per la visualizzazione diretta di un'iterazione e soprattutto per rendere più leggibili i file PostScript che useremo nella grafica, come vederemo nel prossimo capitolo.

I dettagli delle due funzioni:

```
# r = dizionario delle regole.
def Linden (r,a):
  if not isinstance(s,list): a=a.split()
  b=[]
  for x in a:
     y=r.get(x,None)
     if y!=None: b.extend(Lista(y))
  return b
```

Con *isinstance* si può verificare se un oggetto appartiene ad una classe. Con *d.get(x)* si ottiene $d[x]$; questo metodo può essere usato con un secondo argomento con cui si può impostare il valore che viene restituito quando la chiave indicata nel primo non esiste.

```
def Stringa(v,larg=40):
  w=[]; riga=''
  for x in v:
    if x=='':continue
    if len(riga)>larg:
      w.append(riga); riga=x
      else:
    if riga: sep=' '
    else: sep=''
      riga+='%s%s' %(sep,x)
    w.append(riga)
    return '\n'.join(w)
```

Esempio 2.13. Per la successione di Morse possiamo usare le seguenti istruzioni

```
#02-13.py
execfile('linden')
morse={'0':'0 1','1':'1 0'}
x='0'
for k in xrange(8):
    print Stringa(x,30); print
    x=Linden(morse,x)
```

Esempio 2.14. Nel prossimo capitolo vedremo che il sistema di Lindenmayer

$\boxed{n \to nbn; b \to bbb; \Omega = n}$

corrisponde all'insieme di Cantor.

```
#02-14.py
cantor={'n':'n b n','b':'b b b'}
x='n'
for k in range(6):
    print Stringa(x,36); print
    x=Linden(cantor,x)
```

Esempio 2.15. Anche in un sistema non propagante le stringhe successivamente generate possono aumentare di lunghezza:

```
#02-15.py
 regole={'av':'av 0 d','0':'','d':'0 av d av'}
 x='av'
 for k in range(6):
     print Stringa(x,32),print
     x=Linden(regole,x)
```

3. Parentesi ed elaborazione ramificata

Situazione 3.1. A sia un insieme di cui i simboli [e] non ne fanno parte. Poniamo $\hat{A} := A \cup \{[,]\}$.

Definizione 3.2. Sia $x = x_1 \dots x_m \in \hat{A}^*$. Per $1 \leq h \leq m$ siano

$\alpha_h :=$ numero delle volte che [appare in $x_1 \dots x_h$;

$\beta_h :=$ numero delle volte che] appare in $x_1 \dots x_h$.

La parola x si dice *bilanciata* (rispetto alla parentesi quadra) se soddisfa le seguenti condizioni:

1. $m \geq 2$.
2. $\alpha_m = \beta_m$.
3. $\alpha_h > \beta_h$ per ogni $h \in \{1, \dots, m-1\}$.

Siccome per la prop. 3.3 queste condizioni implicano $x_1 =$ [e $x_m =$], esse corrispondono alla richiesta che, in una interpretazione naturale, x rappresenti una lista annidata (eventualmente vuota) di elementi di A.

Proposizione 3.3. *La parola $x = x_1 \dots x_m \in \hat{A}^*$ sia bilanciata. Allora $x_1 =$[e $x_m =$].*

Dimostrazione.

1. Sia $x_1 \neq$ [. Allora $\alpha_1 = 0 \leq \beta_1$ in contraddizione all'ipotesi.
2. Sia $x_m \neq$]. Allora $\beta_m = \beta_{m1} < \alpha_{m1} \leq \alpha_m$, in contraddizione all'ipotesi (2) della def. 3.2.
3. Dalla def. 3.2. perché $1 < m$.

Nota 3.4. La seguente funzione prende come argomento una stringa a e la completa, se necessario, a sinistra e a destra con [rispettivamente] al fine di ottenere una stringa bilanciata.

```
def bilan(a):
  stack=[]; i=0; alfa=beta=-1; n=len(a)
  for j,x in enumerate(a):
    if x== '[': stack.append(j)
    elif x==']'
      if len(stack): alfa=stack.pop()
      else: i+=1; alfa=i
      beta=j
    k=len(stack)
    if k>0: alfa= stack[0]; beta=n+k-1
    u='%s%s%s' %('['*i,a,']'*k)
    if alfa>-i or beta<n+k-1: u='[%s]' %(u)
    return u
```

L'argomento di questa funzione è una stringa. Creo una pila vuota e analizzo il primo carattere della stringa. Se il carattere non è una parentesi passo al carattere successivo, se è una parentesi aperta lo inserisco nella pila; altrimenti è una parentesi chiusa. Quando finiscono i caratteri il numero delle parentesi aperte deve essere uguale a quello delle chiuse, altrimenti vengono aggiunte le parentesi necessarie ad ottenere una stringa bilanciata.

Esempio 3.5. Applichiamo la funzione della nota 3.4 in alcuni casi tipici:

```
for a in ('a]bc]d[ef[gh]]i]jk[lm][n','[ab]c[de]','[]',']]e[u]a',
    'abc','[[',']]','abc[de','[a][b]','[[a][b]][cd]',
    ']a]]b[cd[a[[[[',']ab]'): print bilan(a)

# [[[[a]bc]d[ef[gh]]i]jk[lm][n]]
# [[ab]c[de]]
# []
# [[[]]e[u]a]
# [abc]
# [[]]
# [[]]
# [abc[de]]
# [[a][b]]
# [[[a][b]][cd]]
# [[[[]a]]b[cd[a[[[[]]]]]]]
# [ab]
```

Nota 3.6. La seguente funzione prende come argomento una stringa *che deve essere bilanciata* (ottenuta ad es. usando `bilan`) e la trasforma nella lista annidata di stringhe corrispondente.

```
def annida (a):
  u=a.split(); stack=[]; v=[]
  for x in u:
    if x=='[': stack.append([])
    elif x==']':
      if len(stack)>1: stack[-2].append(stack.pop())
      else: return stack[0]
    else: stack[-1].append(x)
```

Esempio 3.7. Applichiamo la funzione `annida` in alcuni casi tipici.

```
a='[ 1 2 [ 3 5 [ 6 7 [ 8 9 ] 10 11 [ 12 13 ] 14 ] 15 ] 16 ]'
print annida(a)
# ['1', '2', ['3', '5', ['6', '7', ['8', '9'], '10', '11', ['12', '13'],
'14'], '15'], '16']

a=' '.join(list('[[[a]bc]d[ef[gh]]i]jk[lm][n]'))
print annida(a)
# [[['a'], 'b', 'c'], 'd', ['e', 'f', ['g', 'h']], 'i']

a='[ a b ] c [ d e ]'
print annida(a)
# ['a', 'b']
```

```
a='[ a ] [ b ]'
print annida(a)
# ['a']

a='[ [ a ] [ b ] ] [ c d ]'
print annida(a)
# [['a'], ['b']]
```

Nota 3.8. In Python comunque non abbiamo bisogno di algoritmi per le stringhe, perché possiamo lavorare direttamente con liste annidate di funzioni. La funzione di base per l'elaborazione ramificata è perció la seguente:

```
# laf e' una lista annidata di funzioni.
def elab (x,laf):
  if laf==[]: return
  f=laf[0]
  if not isinstance(f,(list,tuple)):
    if callable(f): f(x)
    elab(x,laf[1:])
  else: y=copia(x); elab(x,f); elab(y,laf[1:])
```

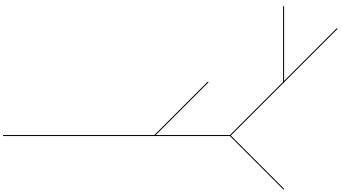
Nota 3.9. Definiamo la seguente classe:

```
class stato:
    def__init__(A,p,dp); A.p=punto(p); A.dp=punto(dp)
    def mov(A): A.p+=A.dp
    def gira(A,t): A.dp=A.dp.rot(t)
```

Adesso con

```
def a(x): copia(x); x.mov(); -retta(u.p,x.p)

def s(x): x.gira(45)

def d(x): x.gira(-45)

File('0309')
x=stato([0,0],[1,0])
laf=[a,a,[s,a],a,[s,a,[s,a],a],d,a]
elab(x,laf)
```

otteniamo la figura



Nota 3.10. Anche nella generazione di sistemi di Lindenmayer lavoreremo direttamente con liste annidate, usando la seguente funzione `linden` al posto di `Linden` (cfr.nota 2.12):

```
#linden.py
# la e' una lista che puo' essere annidata
# diz il dizionario delle regole

def linden(la,diz):
    lb=[]
    for x in la:
        if not isinstance(x,(list,tuple)): lb.extend(diz.get(x,[x]))
        else: lb.append(linden(x,diz))
    return lb
```

Esempio 3.11. :

```
morse={0:[0,1], 1:[0,1]}

la=[0]
for k in xrange(4): la=linden(la,morse); print la

################################################

diz={0:[0,1], 1:[1,[0,[0,1]]]}

la=[0]
for k in xrange(3): la=linden(la,diz); print la
```

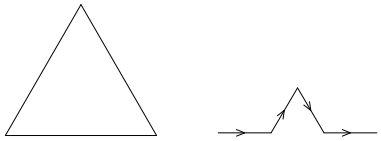
Nota 3.12. Sostituire anche `Stringa` con `stringa` (cfr. nota 2.12). Quando viene indicato il parametro larg, su ogni riga vengono stampati non più di larg caratteri.

```
def stringa (a,larg=None):
  if not isinstance(a,(list,tuple)):return str(a)
  ls=[]
  for x in a:ls.append(stringa(x))
  b='[%s]'%(''.join(ls))
  if larg==None: return b
  w=[]; riga=''
  for x in b.split():
    if x=='':continue
    if len(riga)>larg:
      w.append(riga); riga=x
    else:
      if riga: sep=' '
      else: sep=''
      riga+='%s%s' %(sep,x)
    w.append(riga)
    return '\n'.join(w)
```

4. Rappresentazioni grafiche

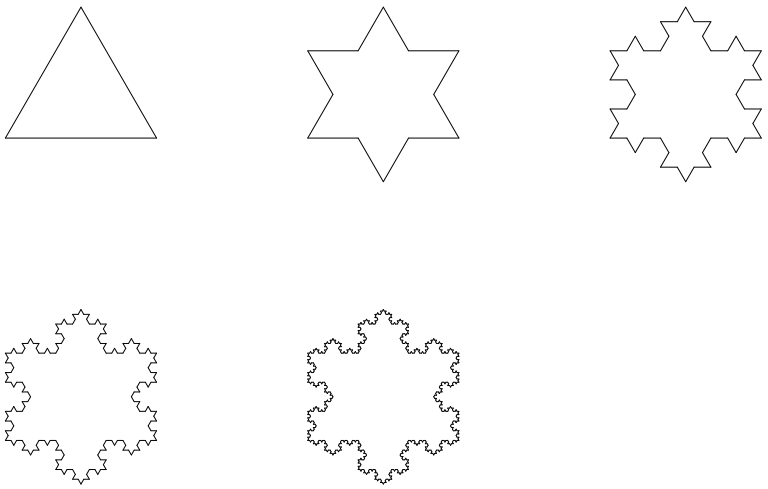
Nota 4.1. Il fiocco di neve di Koch corrisponde alla regola

$\boxed{a \to adassada}$ con iniziatore $\boxed{assassa}$



dove a rappresenta il movimento in avanti, s una rotazione di 60 gradi a sinistra, d una rotazione di 60 gradi a destra. Si parte da un elemento *iniziatore*, il triangolo equilatero, poi si sostituisce ogni lato del triangolo con una linea spezzata in 4 parti, l'elemento *generatore* la cui definizione può essere vista dal disegno:

Nota 4.2.

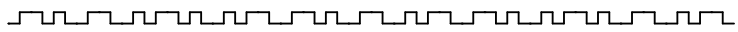


Questa volta dobbiamo aumentare il limite di ricorsione

```
execfile('../programmi/alfa.py')
sys.setrecursionlimit(8000); matita(2)

File('0401a')
def a(x):u=copia(x); x.mov(); -retta(u.p,x.p)

def s(x):x.gira(60)

def d(x):x.gira(-60)

File('0401')
koch={a:[a,d,a,s,s,a,d,a]}
p=[a,s,s,a,s,s,a];dx=6.0
for k in xrange(5):
  dx/=3
  if k<3: x=4*k; y=0
  else: x=4*(k-3); y=-4
```

```
  z=stato([x,y],[dx,0]): elab(z,p)
  if k<5: p=linden(p,koch)

Fine()
```

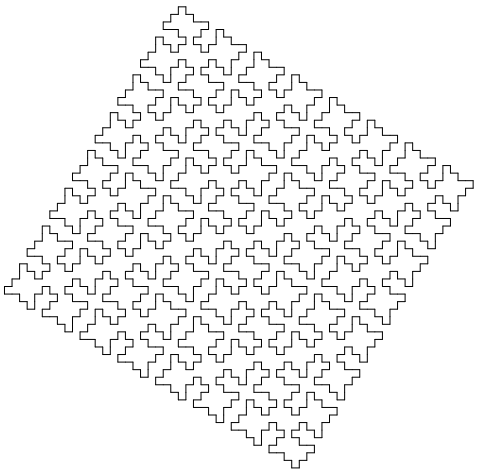
Esempio 4.3. Il metodo usato nella nota 4.2 si chiama *il metodo della tartaruga* ed utilizza triplette (x, y, α), dove le coordinate cartesiane (x, y) rappresentano la posizione della tartaruga e l'angolo α è interpretato come il senso in cui la tartaruga si sta dirigendo. Specificamente, questo metodo può essere applicato per interpretare le stringhe che sono generate dai sistemi di Lindenmayer.



```
File('0403a',scala=0.15);matita(4)
def a(z): w=copia(z);z.p[1]=0; conn(w,z); w=copia(z);z.mov();conn(w,z)
def b(x): w=copia(z);z.p[1]=1; conn(w,z); w=copia(z);z.mov();conn(w,z)
morse={a:[a,b],b:[b,a]}
p=[a]
for k in xrange(6): p=linden(p,morse)
z=stato([0,0],[1,0]); elab(z,p)
```

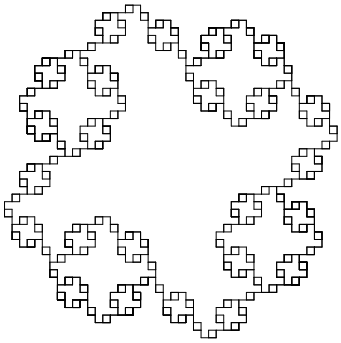
Esempio 4.4. Iniziatore : $as0asas0a$.

Regola: $0 \to 0adasad0asasa0adasad0$.



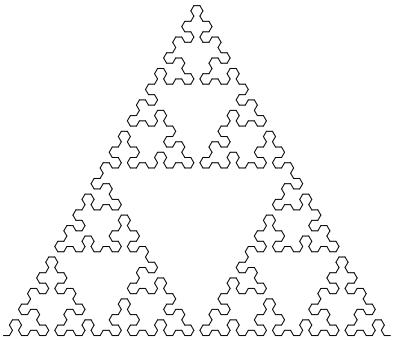
Esempio 4.5. Iniziatore : $asasasa$.

Regola: $a \to aasasasasasada$.



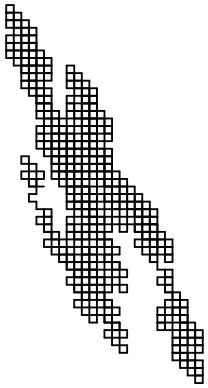
Esempio 4.6. Iniziatore : $0a$.

Regola: $0 \to 1ad0ad1; 1 \to 0as1as0$.



Esempio 4.7. Iniziatore : $a0$.

Regola: $0 \to 0s1das; 1 \to da0d1$.

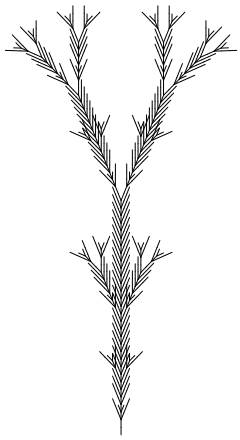


Esempio 4.8. Iniziatore : a.

Regola: $a \to a[sa]a[da]a$.

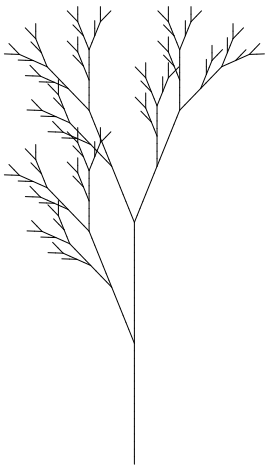
Esempio 4.9. Iniziatore : 1.

Regola: $0 \rightarrow 0[daaa][saaa]a0; 1 \rightarrow 1a0[s1][d1]$.



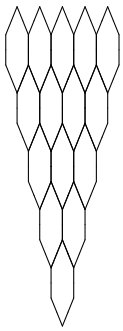
Esempio 4.10. Iniziatore : 0.

Regola: $a \rightarrow aa; 0 \rightarrow a[s0]a[d0]s0$.



Esempio 4.11. Iniziatore : 0.

Regola: $0 \to [daasaa][1]saa; 1 \to [daasaa[1]saa][saadaadaa]$.



5. Trasformazioni di Möbius

Definizione 5.1. Introduciamo le seguenti notazioni, comunemente usate in analisi complessa:

$\bar{\mathbb{C}}$:= sfera di Riemann $\mathbb{C} \cup \{\infty\}$ con la sua naturale topologia e struttura olomorfa.

$\mathbb{D} := \{z \in \mathbb{C} | \ |z| < 1\}$ (disco unitario).

$\mathbb{H} := \{z \in \mathbb{C} | \ \operatorname{Im} z > 0\}$ (semipiano superiore).

Definizione 5.2. Una *trasformazione di Möebius* è un'applicazione $f : \bar{\mathbb{C}} \to \bar{\mathbb{C}}$ della forma $f(z) = \frac{az+b}{cz+d}$ con $a, b, c, d \in \mathbb{C}$ ed $ad - bc \neq 0$. Se per $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in GL(2, \mathbb{C})$ definiamo $Az := \frac{az+b}{cz+d}$, abbiamo perciò $f(z) = A(z)$. Denotiamo quindi in questo caso f con f_A.

La teoria generale delle trasformazioni di Möbius si trova nella tesi di A.Veratelli.

Una trasformazione di Möbius si dice *reale* se è della forma $f(z) = Az$ con $A \in GL(2, \mathbb{R})$.

Osservazione 5.3. Siano $A, B \in GL(2, \mathbb{C})$. Si dimostra facilmente che $f_A = f_B$ se e solo se $B = \lambda A$ con $\lambda \in \mathbb{C} \setminus 0$. Per i determinanti abbiamo allora $detB = \lambda^2 detA$. Se A e B sono reali, anche λ deve essere reale e vediamo che $detB > 0 \Leftrightarrow detA > 0$. In tal caso diciamo che $f := f_A = f_B$ è una trasformazione di Möbius a determinante positivo.

Proposizione 5.4. *Siano $A, B \in GL(2, \mathbb{C})$. Allora $f_A \circ f_B = f_{AB}$.*

Vediamo in pratica che la composizione di due trasformazioni di Möbius è ancora una trasformazione di Möbius e che la composizione di due trasformazioni di Möbius reali è ancora una trasformazione di Möbius.

Dimostrazione. Siano

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \text{e} \qquad B = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

Allora

$$AB = \begin{pmatrix} a\alpha + b\gamma & a\beta + b\delta \\ c\alpha + d\gamma & c\beta + d\delta \end{pmatrix}$$

mentre per $z \in \mathbb{C}$ abbiamo

$$f_A \circ f_B(z) = f_A(\frac{\alpha z + \beta}{\gamma z + \delta}) = \frac{a\frac{\alpha z + \beta}{\gamma z + \delta} + b}{c\frac{\alpha z + \beta}{\gamma z + \delta} + d}$$
$$= \frac{a(\alpha z + \beta) + b(\gamma z + \delta)}{c(\alpha z + \beta) + d(\gamma z + \delta)} = \frac{(a\alpha + b\gamma)z + a\beta + b\delta}{(c\alpha + d\gamma)z + c\beta + d\delta} = f_{AB}(z)$$

Definizione 5.5. D sia un dominio di $\bar{\mathbb{C}}$. Un *automorfismo olomorfo* di D è un'applicazione biolomorfa $D \to D$.

Teorema 5.6. *(1) Gli automorfismo olomorfi di $\mathbb{C}$ sono esattamente le applicazioni della forma $\bigcirc_z az + b$ con $a \in \mathbb{C} \setminus 0$.*

(2) Gli automorfismi olomorfi di $\bar{\mathbb{C}}$ sono esattamente le trasformazioni di Möbius.

(3) Gli automorfismi olomorfi di $\mathbb{D}$ sono esattamente le trasformazioni di Möbius della forma

$f(z) = e^{i\alpha} \frac{z+w}{\bar{w}z+1}$

con $w \in \mathbb{D}$ ed $\alpha \in \mathbb{R}$. Si noti che $f(0) = w$. Per $w = 0$ ed $\alpha = 0$ si ottiene l'identità.

(4) Gli automorfismi olomorfi di $\mathbb{H}$ sono esattamente le trasformazioni di Möbius reali a determinanti positivi. Esse sono quindi le applicazioni della forma

$f(z) = \frac{az+b}{cz+d}$ *con* $a, b, c, d \in \mathbb{R}$

e, senza perdita in generalità, $ad - bc = 1$. Questa è una delle ragioni per la grande importanza del gruppo $SL(2, \mathbb{R})$ delle matrici 2 x 2 reali a determinante uguale a 1 nell'analisi complessa.

Dimostrazione. Veratelli, pagg. 151-153,160-162,186.

Proposizione 5.7. *Le trasformazioni di Möbius reali $f \neq \mathrm{id}$ definite su $[0, 1]$ e tali che $f(0) = 0$ ed $f(1) = 1$ sono esattamente le applicazioni $f = \bigcirc_x \frac{ax}{x+a-1}$ con $a \in \mathbb{R} \setminus [0, 1]$.*

Dimostrazione. Seguiamo Gazzetta, pag.90.

(1) f è della forma $f(x) = \frac{ax+b}{cx+d}$ con $a, b, c, d, \in \mathbb{R}$ ed $ad - bc \neq 0$.

Dalla condizione $f(0) = 0$ segue necessariamente che $b = 0$.

Perciò $f(x) = \frac{ax}{cx+d}$.

(2) Sia $c = 0$. Allora $f(x) = \frac{ax}{d}$. L'ipotesi $f(1) = 1$ implica $a = d$

e quindi $f = id$, in contraddizione all'ipotesi.

(3) Abbiamo quindi $c \neq 0$. Perciò $f(x) = \frac{ax}{cx+d} = \frac{(a/c)x}{x+d/c}$,

cosicché possiamo assumere che $f(x) = \frac{ax}{x+d}$.

Adesso $1 = f(1) = \frac{a}{1+d}$, per cui $d = a - 1$.

(4) f è definita su $[0,1]$ se e solo se $-d \notin [0,1]$, quindi se e solo se

$1-a \notin [0,1]$, e ciò accade se e solo se $a \notin [0,1]$.

Proposizione 5.8. *Sia* $f := \bigcirc_x \frac{ax}{x+a-1}$ *con* $a \in \mathbb{R} \setminus [0,1]$. *Allora:*

1. f *è definita su* $[0,1]$.
2. f *è strettamente crescente.*
3. $f([0,1]) = [0,1]$.
4. *Se* $a < 0$*, allora* $f(x) < x$ *per ogni* $x \in (0,1)$.

   Se $a > 1$*, allora* $f(x) > x$ *per ogni* $x \in (0,1)$.

Dimostrazione. Gazzetta, pagg. 90-91.

Osservazione 5.9. Scambiando x con $1-x$ vediamo che le trasformazioni di Möbius reali $f \neq \mathrm{N}$ definite su $[0,1]$ con $f(0) = 1$ e $f(0) = 0$ sono esattamente le applicazioni dela forma $f := \bigcirc_x \frac{a(1-x)}{1-x+a-1} = \frac{a(1-x)}{a-x}$. Anche in questo caso si ha $f([0,1]) = [0,1]$.

6. Modelli replicativi nel piano

Nota 6.1. Presentiamo in questo capitolo una generalizzazione dei sistemi di Lindenmayer basata sulla seguente idea:

(1) I sia un insieme finito $\neq \emptyset$ ed S un gruppoide (cioè un insieme $\neq \emptyset$ dotato di un'applicazione $S \times S \longrightarrow S$) non necessariamente finito. Allora per il teorema 1.7 ogni applicazione $I \times S \xrightarrow{\varphi_0} (I \times S)^*$ può essere estesa ad un unico endomorfismo φ di $(I \times S)^*$.

(2) Consideriamo adesso applicazioni φ_0 della forma

$$(i \times s) \mapsto (e_{i_1}, t_{i_1}, s) \dots (e_{i_m}, t_{i_m}, s)$$

Iterando l'endomorfismo φ così ottenuti, da ogni coppia (i, s) di partenza otteniamo parole della forma

$$(i_1, s_1) \dots (i_k, s_k).\text{(*)}$$

(3) Assumiamo adesso che X sia un insieme e che S sia un sottoinsieme di X^X, cioè un insiem di applicazioni $X \to X$. Da una successione della forma (*) possiamo ottenere la successione di funzioni $(s_1, \dots s_k)$ e da essa, per ogni $x \in X$, la successione $(s_1(x), \dots s_k(x))$.

Nota 6.2. Nel seguito applichiamo l'idea della nota 6.1 nel seguente contesto: D sia un sottoinsieme di $\mathbb{R}^2$, K un insieme finito (i cui elementi verranno usati nei disegni come colori, ma che possono esprimere attributi di vario tipo nelle applicazioni concrete). Posto X:= $D \times K$, assumiamo infine che S sia un sottomonoide (rispetto alla composizione di applicazioni) di X^X. È chiaro che D potrebbe essere un insieme qualsiasi; solo per le rappresentazioni grafiche consideriamo $D \subset \mathbb{R}^2$, ad es. $D = \mathbb{R}^2, D = [0, 1]^2, D = \mathbb{D}, D = \mathbb{H}$.

Nota 6.3. Definiamo quindi le funzioni *comp* per la composizione di funzioni e *glinden* per i sistemi di Lindenmayer generalizzati nel modo seguente:

```
# Composizione di funzioni.
def comp (f,g): return lambda x: f(g(x))

# Funzione per i sistemi di Lindenmayer generalizzati.
# la e' una lista di coppie della forma (i,s).
def glinden (la,diz):
  lb=[]
  for (i,s)in la:
    u=diz[i]
    for (j,t) in u: lb.append([j.comp(t,s)])
return lb
```

Definiamo inoltre una classe i cui oggetti rappresentano punti colorati:

```
# Punto colorato.
class ptc:
  def __init__ (A.p,colore): A.p=p; A.colore=colore
  def disegna(A): C=cerchio(0.01,A.p); + C ** colore(a.colore); - C
```

Nota 6.4. Negli esempi in questo capitolo applicheremo la costruzione generale della nota 6.1 agli automorfismi olomorfi del disco unitario descritti nel teorema 5.6. Definiamo in Python la seguente funzione che genera queste trasformazioni:

```
# Trasformazioni del cerchio con f(0)=w
def moebius(w,alfa,colfun=None):
  w=complex(x); v=w.conjugate()
  u=math.cos(alfa)+math.sin(alfa)*1j
  def f(A):
    z=complex(*A.p);z=u*(z+w)/(v*z+1)
    if not colfun: colore=A.colore
    else: colore=colfun(A.colore)
    return ptc([z.real,z.imag].colore)
return f
```

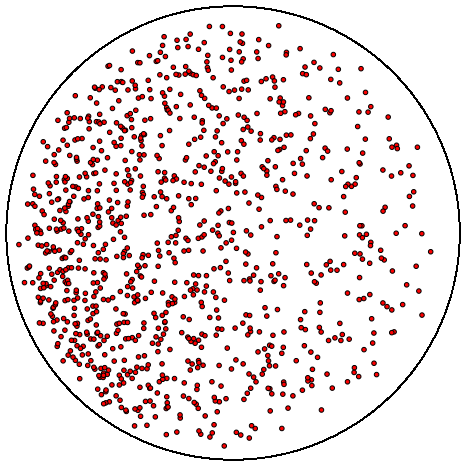
Esempio 6.5. Nel primo esempio usiamo la regola

$$0 \longrightarrow (0,f)(1,g)$$
$$1 \longrightarrow (1,h)(0,k)$$

in cui, partendo con il punto colorato $(0,0,'red')$ con le istruzioni

```
f=moebius(0.2,4)
g=moebius(0.4j,0)
h=moebius(-0.2,0.5)
k=moebius(0.1+0.2j,2)

########################

File('0605',scala=3)
diz={0:[[0,f],[1,g]],1:[[1,h],[0,k]]}
la=[[0,id]]; A=ptc([0,0],'red')
for j in xrange(10): la=glinden(la,diz)
for(i,f) in la: f(A).disegna() -cerchio(1,[0,0])
```

otteniamo la figura



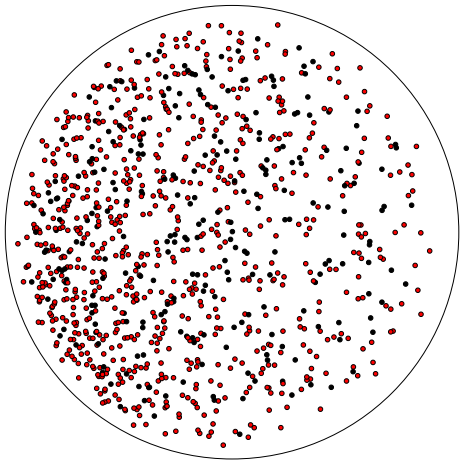
Esempio 6.6. Se nell'esempio 6.5 sostituiamo la definizione di f e g con

```
f=moebius(0.2,4,nero)
g=moebius(0.4j,0,rosso)
```

dove abbiamo definito

```
def nero (c): return 'black'

def rosso (c): return 'red'
```

otteniamo

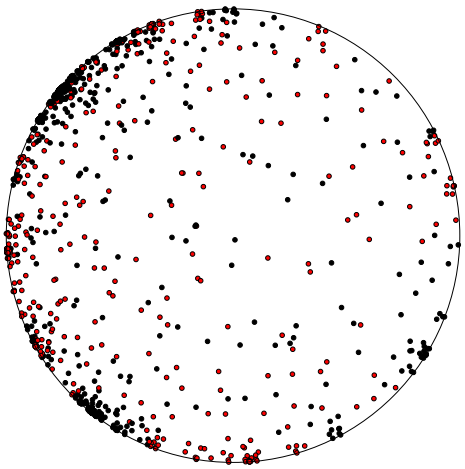


Questa figura si distingue da quella dell'esempio 6.5 per i colori dei punti.

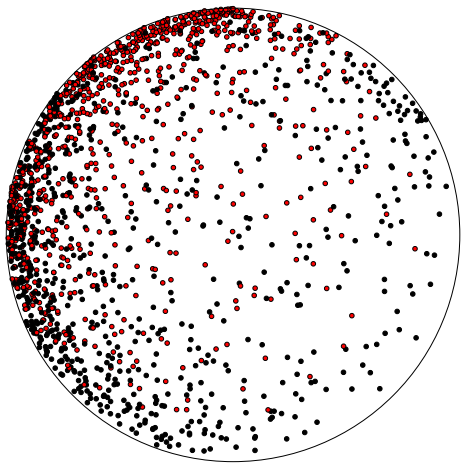
Esempio 6.7. Predefinendo la funzione di trasformazione nel modo seguente

```
f=moebius(0.2,4,nero)
g=moebius(-0.1+0.4j,0,rosso)
h=moebius(-0.2+0.7j,0.5,nero)
k=moebius(0.1+0.2j,2)
```

otteniamo



Esempio 6.8. Otteniamo la figura



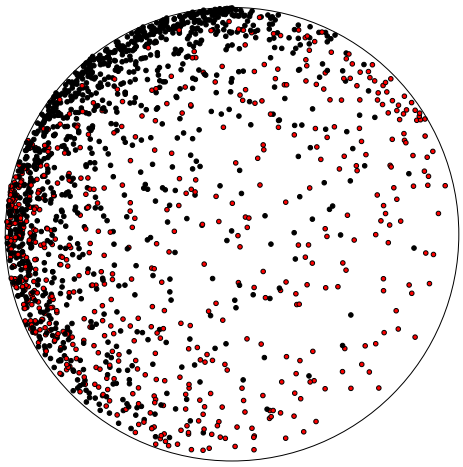
con le istruzioni

```
f=moebius(0.2,4,nero)
g=moebius(0.4j,0,rosso)
h=moebius(-0.2,0.5,nero)
k=moebius(0.4+0.2j,2)
######################
File('0608',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,h],[0,k],[1,g]]}
la=[[0,id]]; A=ptc([0,0],'red')
for j in xrange(8): la=glinden(la,diz)
for (i,f) in la: f(A).disegna()
- cerchio(1,[0,0])

Fine()
```

Esempio 6.9. Possiamo anche far dipendere il colore del disegno dal parametro i come nelle istruzioni

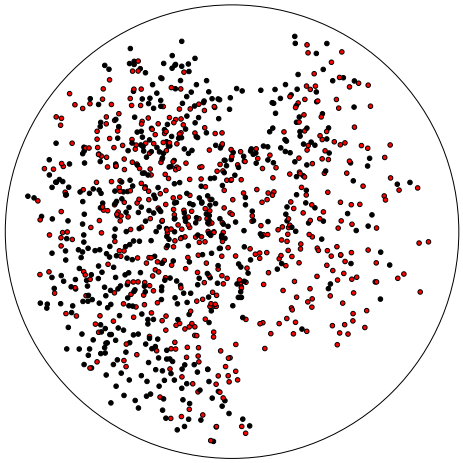
```
f=moebius(0.2,4)
g=moebius(0.4j,0)
h=moebius(-0.2,0.5)
k=moebius(0.4+0.2j,2)
####################
File('0609',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,h],[0,k],[1,g]]}
la=[[0,id]]; A=ptc([0,0],'red')
for j in xrange(8): la=glinden(la,diz)
for (i,f) in la:
  B=f(A)
  if i==0: B.colore='red'
  else: B.colore='black'
  B.disegna()
- cerchio(1,[0,0])

Fine()
```

con le quali otteniamo



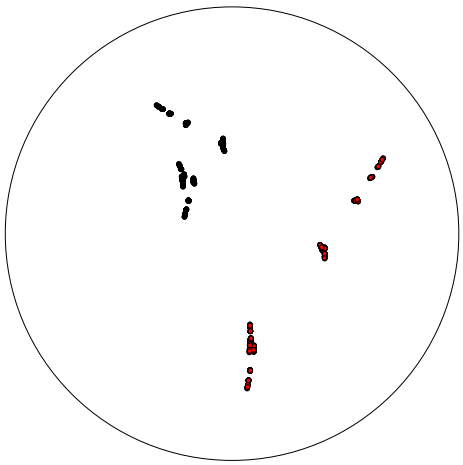
Esempio 6.10. La figura seguente è simile a quella dell'esempio 6.6, presenta però alcune regioni in cui non appaiono punti. La si ottiene con le istruzioni

```
f=moebius(0.2,4)
g=moebius(0.2j,0,nero)
h=moebius(-0.2,0.5)
k=moebius(0.4,2,rosso)
####################
File('0610',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,h],[0,k]]}
la=[[0,id]]; A=ptc([0,0],'red')
for j in xrange(10): la=glinden(la,diz)
for (i,f) in la: f(A).disegna()
- cerchio(1,[0,0])

Fine()
```



Esempio 6.11. Figure piuttosto diverse si ottengono tramite trasformazioni affini:

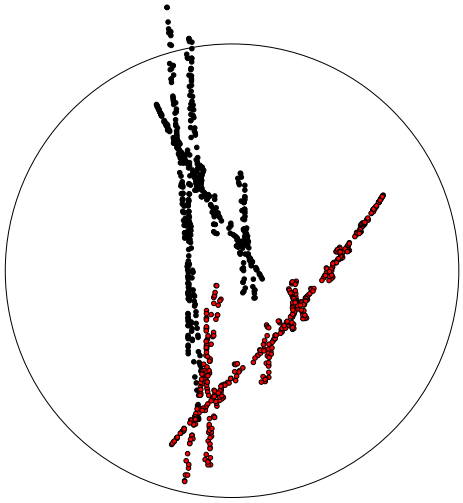
```
def aff (a,b,c,d,u,v,colfun=None):
  def f (A):
    (x,y)=A.p
    if not colfun: colore=A.colore
    else: colore=colfun(A.colore)
    return ptc([a*x+b*y+u,c*x+d*y+v],colore)
  return f

f=aff(0.6,-0.1,0.4,0.2,0.3,0)
g=aff(-0.5,0,0,0.2,0,0.5,nero)
h=aff(0.2,0.1,0.3,-0.4,-0.2,0.4)
k=aff(0.1,0,0.3,-0.5,0.1,-0.3,rosso)
######################
File('0611',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,h],[0,k]]}
la=[[0,id]]; A=ptc([0.4,0],'red')
for j in xrange(10): la=glinden(la,diz)
for (i,f) in la: f(A).disegna()
- cerchio(1,[0,0])

Fine()
```



Esempio 6.12. Con le istruzioni

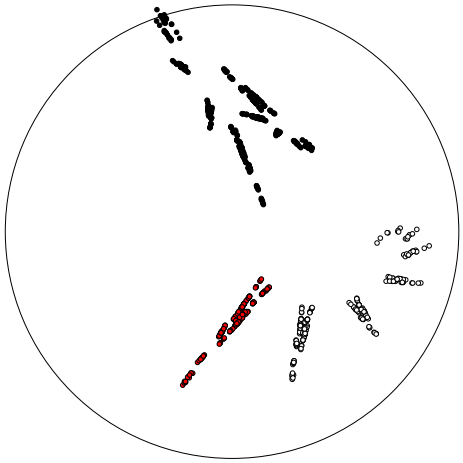
```
f=aff(0.6,-0.1,0.4,0.2,0.3,0)
g=aff(-0.5,0,0,0.7,0,0.5,nero)
h=aff(0.2,0.1,1.3,-1,-0.2,0.8)
k=aff(1.1,0,0.3,-0.5,0.1,-0.3,rosso)
####################
File('0612',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,h],[0,k]]}
la=[[0,id]]; A=ptc([0.4,0],'red')
for j in xrange(10): la=glinden(la,diz)
for (i,f) in la: f(A).disegna()
- cerchio(1,[0,0])

Fine()
```

otteniamo la figura



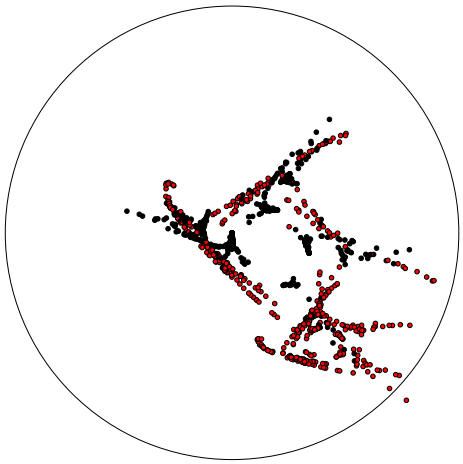
Esempio 6.13. Un esempio affine a tre colori:

```
f=aff(0.8,-0.5,0.4,0.4,0.1,-0.3,bianco)
g=aff(-0.5,0.1,0.7,0.2,0.1,0.4,nero)
h=aff(0.4,-0.2,0.3,-0.4,0.1,-0.2,rosso)
######################
File('0613',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,g],[0,h]]}
la=[[0,id]]; A=ptc([0,0],'red')
for j in xrange(10): la=glinden(la,diz)
for (i,f) in la: f(A).disegna()
- cerchio(1,[0,0])

Fine()
```



Esempio 6.14. Un esperimento con funzioni quadratiche:

```
def fun (phi,colfun=None):
  def f (A):
    (x,y)=A.p
    if not colfun: colore=A.colore
    else: colore=colfun(A.colore)
    return ptc(phi(x,y),colore)
  return f
f=fun(lambda x,y: [x*y,0.3*x+0.2*y],nero)
g=fun(lambda x,y: [y+x*x+0.2,y*y-x+0.1])
h=fun(lambda x,y: [0.5*x-0.2*y+0.3,0.2*x+0.4*y-0.5],rosso)
######################
File('0614',scala=3)
diz={0:[[0,f],[1,g]], 1:[[1,g],[0,h]]}
la=[[0,id]]; A=ptc([0,0],'red')
for j in xrange(10): la=glinden(la,diz)
for (i,f) in la: f(A).disegna()
- cerchio(1,[0,0])

Fine()
```



Bibliografia

A. Cain: Presentations for subsemigroups of groups. PhD thesis Univ. Edinburgh 2005.

A. Clifford/G. Preston: The algebraic theory of semigroups II. AMS 1967.

O. Deussen: Computergenerierte Pflanzen. Springer 2003.

D. Gambi: Sistemi di Lindenmayer e automi cellulari. Tesi, Ferrara 1991.

M. Gazzetta: Legami stocastici, t-norme e logica fuzzy. Tesi LT Ferrara 2007.

P. Prusinkiewicz/A. Lindenmayer: The algorithmic beauty of plants. Springer 1990.

A. Veratelli: Le trasformazioni di Mbius. Tesi Ferrara 1988.