a cura di Francesco Romani

Conversione automatica di grafici da *Mathematica* a Java

Viene presentato un esempio di traduzione automatica di grafici descritti da un programma *Mathematica* in grafici equivalenti descritti nel linguaggio Java. Lo scopo per cui presentiamo il nostro traduttore è soprattutto "didattico", per mostrare un esempio di generazione automatica di codice in *Mathematica*. Un possibile utilizzo pratico potrebbe essere la generazione in *Mathematica* di grafici "intelligenti" come carte geografiche, mappe tematiche grafi ecc. e la loro visualizzazione in programmi Java.

Introduzione

È probabilmente superfluo sottolineare l'importanza che riveste il linguaggio **Java** nel panorama dell'informatica attuale: ben lungi dall'essere solo "il linguaggio per mettere le animazioni nelle pagine Web" **Java** ha tutte le carte in regola per divenire la lingua comune per i prossimi anni. **Java** è un linguaggio multipiattaforma, come *Mathematica*, essendo a basso livello e le esecuzioni sono molto più efficienti ma la programmazione è meno immediata.

Il programma di traduzione che presentiamo è molto semplificato e non copre tutta la varietà dei grafici Mathematica ma, solo parzialmente, alcuni grafici bidimensionali.

Tuttavia il programma completo è ancora troppo lungo per essere integralmente pubblicato. Provvederò ad inviare il *Notebook* a coloro che ne faranno richiesta con una mail a `romani@di.unipi.it` (per favore controllate che il campo `Reply to:` sia corretto; vi sono alcuni lettori che stanno aspettando ancora il programma dell'analisi di Fourier perché il mail server continua a darmi "`permanent fatal error`" sul loro indirizzo).

Il programma Java

Vediamo dapprima come può essere strutturato un programma **Java** che disegna un semplice grafico. Abbiamo il programma principale `finestra.java` che è fisso:

```
import java.awt.*;

public class finestra extends Frame {
       public static void main(String args[]) {
              new finestra();}

       public finestra() {
              super("grafico");
              setSize(grafico.DX,grafico.DY);
              setLayout(new BorderLayout());
              add("Center",new grafico());
              setVisible(true);}
}
```

Il programma `grafico.java` definisce la classe grafico e il metodo `paint`. Nell'esempio seguente, scritto a mano, viene disegnato un pallino rosso.

```
import java.awt.*;

public class grafico extends Component {
       public static int DX=500,DY=250;

       public void paint(Graphics g) {
              g.setColor(new
Color((float)1,(float)0,(float)0));
              g.fillOval(140,140,100,100);}
}
```

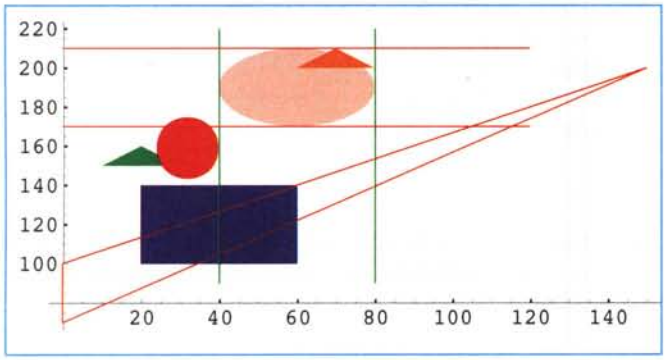

Figura 1

Le dimensioni del grafico in punti (le costanti `DX` e `DY`) sono stabilite nella classe `grafico` e utilizzate nel programma `finestra`. Il nostro scopo è scrivere una funzione *Mathematica* che dato un'espressione di tipo **Graphics** generi automaticamente il file `grafico.java` che eseguito in **Java** disegna **lo stesso** grafico.

Trattamento del grafico originario

Un grafico bidimensionale in *Mathematica* (scritto direttamente o prodotto, ad esempio, da una funzione **Plot**) è una espressione composta dalla funzione **Graphics** seguita da una lista (di liste) di primitive e da eventuali opzioni. Per esempio il grafico

In[1]:=

```
gl=Graphics[
{{Pink,
   {Blue,Rectangle[{20,100},{60,140}]},
  Disk[{60,190},20]},
 {Green,

Polygon[{{10,150},{20,160},{30,150},{10,150}
}]},
 {Orange,

Polygon[50+{{10,150},{20,160},{30,150},{10,1
50}}]},
 {Red,
    PointSize[0.1],Point[{32,159}],

Line[{{0,100},{150,200},{0,70},{0,100}}],
   {Green,
      Line[{{40,90},{40,220}}],
      Line[{{80,90},{80,220}}]},
   {Line[{{0,210},{120,210}}],
    Line[{{0,170},{120,170}}]}}},
{PlotRange->All,
Axes->True,
AspectRatio->1/2}];
```

può essere visualizzato con la funzione **Show**.

In[2]:=

```
Show[gl]
```

(Vedi Figura 1)

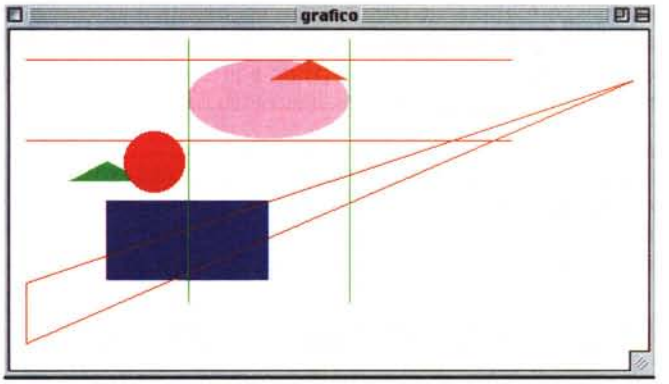

Figura 2

Le primitive grafiche possono essere solamente dichiarative (per esempio come **RGBColor** che determina il colore delle figure che seguono) oppure possono disegnare qualcosa. La possibilità di avere liste di liste permette di cambiare solo "localmente" il colore o la dimensione delle linee e poi di ritornare a quelli precedentemente stabiliti. Prima di porsi il problema della interpretazioni è necessario fare due operazioni:

- trasformare la lista di liste in una lista semplice che disegni lo stesso grafico;
- stabilire le dimensioni del grafico in punti e i fattori di scala.

La funzione `flatten`

La prima operazione non può essere effettuata applicando semplicemente la funzione **Flatten** al grafico: si perderebbe la struttura a sottografici e il disegno risultante potrebbe essere diverso. Provate per esempio ad eseguire

```
Show[Graphics[Flatten[gg[[1]]], gg[[2]]]].
```

e vedrete che molti colori cambiano.

La funzione **flatten** (qui non riportata) è un programma piuttosto complicato che fa uso della ricorsione e delle regole di riscrittura per tradurre un oggetto **Graphics** in uno equivalente senza liste annidate. Come effetto secondario converte le primitive **Gray** e **Hue** in **RGBColor** in modo che vi sarà solo da implementare la traduzione di quest'ultima. L'altra primitiva dichiarativa trattata è **PointSize**; per semplicità **Thickness**, **Dashing**, **AbsoluteThickness**, etc. sono ignorati (nel grafico tradotto tutte le linee saranno continue e avranno spessore 1 punto).

Scalatura

In *Mathematica* le coordinate possono variare in modo qualunque in **Java** bisogna esprimerle in punti. La funzione **scale** (qui non riportata) applicata ad un oggetto di tipo **Graphics** determina i valori **X0**, **X1**, **Y0**, **Y1** che rappresentano i limiti del grafico e i valori **DX** e **DY** che rappresenteranno le dimensioni in punti del grafico **Java**.

Servono anche definite altre quattro funzioni che durante la tra-

duzioni permetteranno di convertire coordinate e lunghezze in punti. La funzione **convy** effettua anche il rovesciamento in quanto in **Java** l'origine vale **(1,1)** ed è in alto a destra e quindi per valori crescenti di **y** scende verso il basso.

In[1]:=
```
convx[x_]:=Floor[1.5+(DX-1)(x-X0)/(X1-X0)];
convy[y_]:=Floor[DY+0.5-(DY-1)(y-Y0)/(Y1-Y0)];
diffx[r_]:=Floor[0.5+(DX-1)r/(X1-X0)];
diffy[r_]:=Floor[0.5+(DY-1)r/(Y1-Y0)];
```

Implementazione delle primitive

Vediamo nel seguito in dettaglio l'implementazione della traduzione delle primitive grafiche. Si tratta di una funzione **wr** che riceve una primitiva grafica con i suoi argomenti e scrive nel file le corrispondenti istruzioni **Java**.

RGBColor

RGBColor ha come equivalente **Java** il comando `setColor` e la traduzione è immediata.

In[1]:=
```
wr[RGBColor[x_,y_,z_]]:=
       WriteString[FILE,"\t\tg.setColor(new
Color((float)",
   x,",(float)",
   y,",(float)",
   z,"));\n"]
```

Rectangle

Rectangle ha come equivalente **Java** il comando `fillRect` ma il significato degli argomenti è diverso ed è necessaria una conversione.

In[2]:=
```
wr[Rectangle[{x1_,y1_},{x2_,y2_}]]:=
WriteString[FILE,"\t\tg.fillRect(",
  convx[Min[x1,x2]],",",
  convy[Max[y1,y2]],",",
  diffx[Abs[x2-x1]],",",
  diffy[Abs[y2-y1]]-1,");\n"];
```

Disk

Disk permette di disegnare sia un ellisse che in cerchio e bisogna distinguere i due casi. In *Mathematica* **Disk** può essere usato anche per disegnare uno spicchio, questa funzione non è stata implementata.

In[3]:=
```
wr[Disk[{x1_,y1_},r_]]:=
 WriteString[FILE,"\t\tg.fillOval(",
  convx[x1-r],",",
  convy[y1+r],",",
  diffx[2 r]-1,",",
  diffy[2 r]-1,");\n"];
```

In[4]:=
```
wr[Disk[{x1_,y1_},{rx_,ry_}]]:=
 WriteString[FILE,"\t\tg.fillOval(",
  convx[x1-r],",",
  convy[y1-r],",",
  diffy[2 rx]-1,",",
  diffy[2 ry]-1,");\n"];
```

PointSize

PointSize non scrive nulla sul file **Java**, si limita a modificare nel programma *Mathematica* il valore della variabile globale **POINTSIZE**

In[5]:=
```
wr[PointSize[x_]]:=POINTSIZE=x;
```

Point

La differenza tra **Point** e **Disk** consiste nel fatto che **Disk** ha una forma che dipende dai rapporti di scalatura (infatti nel grafico il cerchio rosa viene scalato e diviene un ellisse) mentre **Point** è sempre un cerchio di dimensione relativa **POINTSIZE**.

In[6]:=
```
wr[Point[{x_,y_}]]:=
 WriteString[FILE,"\t\tg.fillOval(",
    convx[x]-(rr=Floor[POINTSIZE/2  *  (DX-
1)]),",",
  convy[y]-rr,",",
  2 rr,",",
  2 rr,");\n"];
```

Circle

Circle si implementa come **Disk** con l'unica variante dell'uso di `drawOval` invece che `fillOval`.

Text

L'implementazione di **Text** è parziale ed approssimativa, per ottenere una reale corrispondenza si dovrebbe lavorare in modo molto più accurato sulla collocazione delle lettere in funzione della dimensione del font.

In[7]:=
```
wr[Text[s_,{x_,y_}]]:=
 WriteString[FILE,"\t\tg.drawString(\"",
  ss=ToString[s],"\",",
  convx[x]-2 StringLength[ss],",",
  convy[y]+3,");\n"];
```

Line e Polygon

Anche le funzioni che trattano **Line** e **Polygon** pongono molti problemi e sono troppo lunghe per essere riportate. In **Java** si può usare `drawLine` per un singolo segmento `drawPolyline` per una spezzata poligonale e `fillPolygon` per un poligono. le ultime due primitive richiedono come argomento due *array* di interi e bisogna generarne le definizioni. Per esempio il comando *Mathematica* :

```
Polygon[{{10,150},   {20,160},   {30,150},
```

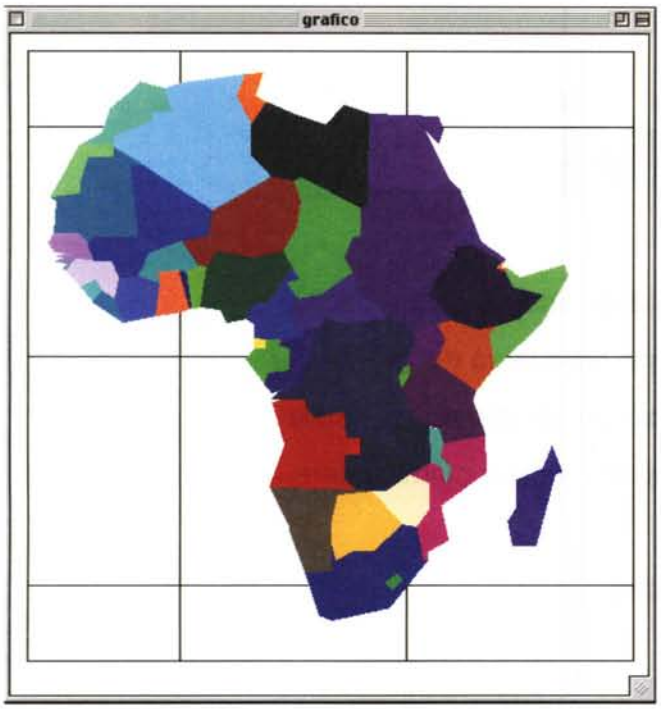

Figura 3

```
{10,150}}]
```

viene tradotto come:

```
{int[] x = {45,76,108,45};
 int[] y = {118,102,118,118};
 g.fillPolygon(x,y,4);}
```

I numeri sono diversi perché è stata effettuata la conversione da coordinate cartesiane a punti assoluti.

Generazione dei file Java

Il programma di traduzione ha una struttura molto semplice. Si applica il programma di scalatura, si apre il file del codice tradotto e si scrivono le intestazioni iniziali (tra cui i valori attuali di **DX** e **DY**). Quindi si applica **wr** a **flatten[g]** traducendo le singole componenti del programma grafico e infine si chiude il file.

In[1]:=

```
Java[g_Graphics] := (
 scale[g]];
 FILE=OpenWrite["grafico.java"];
 WriteString[FILE,
  "import java.awt.*;\n\n"];
 WriteString[FILE,
    "public class grafico extends Component
{\n"];
 WriteString[FILE,
                "\tpublic      static      int
DX=",DX,",DY=",DY,";\n\n"];
 WriteString[FILE,
   "\tpublic void paint(Graphics g) {\n"];
 WriteString[FILE,
                          "\t\tg.setFont(new
Font(\"Courier\",Font.PLAIN,10));\n"];
 Scan[wr, flatten[g][[1]]];

 WriteString[FILE,"\t}\n}\n"];
       Close[FILE]);
```

Esempi

Vediamo tre esempi abbastanza significativi. Il primo è il grafico **g1** definito prima.

In[1]:=

```
Java[g1];
```

(Vedi Figura 2)

I due grafici di Figura 1 e 2 non sono identici perché il disegno degli assi non è stato implementato.

Il secondo esempio è la mappa dell'Africa ottenuta col pacchetto **WorldPlot**.

In[2]:=

```
<<Miscellaneous`WorldPlot`;
Java[WorldPlot[{Africa,RandomColors},
     WorldToGraphics -> True]];
```

(Vedi Figura 3)

L'ultimo esempio è la carta dei fusi orari e delle linee alba-tramonto che avevo prodotto nell'articolo sul cambiamento di data (*Tempus Fugit*, MC n. 169 , gennaio 1997).

In[3]:=

```
Java[sole]
```

(Vedi Figura 4)

Termino con un doveroso ringraziamento ad Antonio Cisternino per la consulenza **Java** gentilmente prestatami.

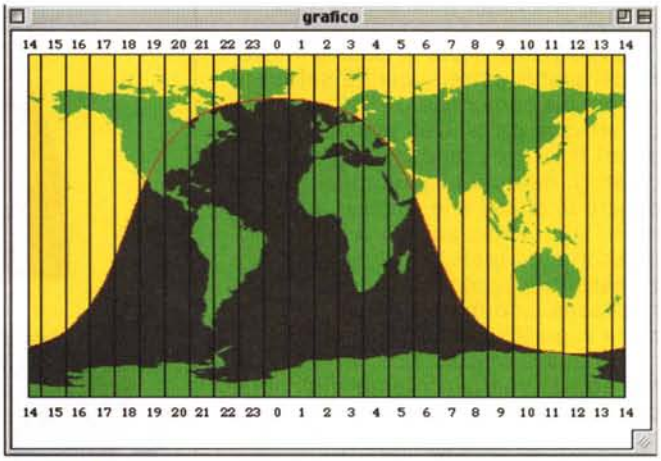

Figura 4

MC

Bibliografia

Stephen Wolfram, **The Mathematica Book, 4rd ed.** (Wolfram Media/Cambridge University Press, 1999)