Personalizzazione e riproducibilità nel disegno programmato: introduzione al pacchetto pgfkeys

Claudio Fiandrino

Sommario

Il paradigma chiave-valore è una delle principali potenzialità della famiglia di pacchetti PGF. Cambiando poche opzioni nel codice si ottiene alta personalizzazione e facile riproducibilità dei disegni. L'articolo illustra alcuni metodi per utilizzare in modo efficiente chiavi e stili di TikZ.

Abstract

The key-value paradigm is one of the most important features of the packages belonging to the PGF family. By changing few options it is possible to achieve high customization and easy reproducibility of pictures. This article shows the methods to efficiently use keys and styles with TikZ.

1 Introduzione

Come è ben noto, utilizzare i linguaggi di disegno programmato che LaTeX offre come Ti*k*Z (Tantau, 2013) permette di ottenere alta qualità nel disegno. In particolare, un segno distintivo è la coerenza tra i font del testo e delle figure, una caratteristica che è più difficile ottenere lavorando con software esterni.

Dal punto di vista del flusso di lavoro, il dibattito in merito alla qualità e potenzialità dei linguaggi di disegno programmato rispetto ad altri software dedicati, nell'ambito del disegno scientifico, ad oggi è ancora aperto. Un punto a favore di questi ultimi è sicuramente la rapidità nel realizzare disegni semplici, per i quali con pochi colpi di mouse si includono le parti essenziali della figura. Quando si parla di disegni complessi, *probabilmente* le tempistiche di realizzazione sono comparabili. L'incertezza di chi scrive, visibile nel corsivo, è motivata dall'assenza di letteratura scientifica comparativa a riguardo.

Molto spesso la rapidità di realizzazione è l'unica metrica considerata nel confronto tra le metodologie e probabilmente l'ostacolo più serio per chi si affaccia per la prima volta al mondo del disegno programmato. Tuttavia, ci si scorda sempre che le valutazioni basate su una metrica sola sono, quanto meno, mai completamente veritiere. Parlando del disegno programmato, oserei dire *bugiarde*. Il motivo è molto semplice: considerando altri parametri di confronto come la *personalizzazione* e la *riproducibilità*, probabilmente la bilancia pende verso il disegno programmato.

Con *personalizzazione* in questo articolo si intende la capacità di definire in modo consistente le proprietà degli oggetti in una figura e poterle cambiare in modo semplice. Per *riproducibilità* invece, si intende la capacità, in termini di codice e metodologia, di utilizzare interamente o singole parti di figure preesistenti ed eventualmente apportare modifiche o aggiunte.

Questi aspetti fondamentali appartengono ad ogni linguaggio di disegno programmato: non solo Ti*k*Z quindi, ma anche PSTricks, Asymptote e Metapost tra gli altri. In questo articolo ci si focalizza su Ti*k*Z, evidenziando i meccanismi con cui si ottiene alta personalizzazione e riproducibilità. In particolare, quindi, si prendono in esame le funzionalità base del pacchetto pgfkeys.

2 Personalizzazione e riproducibilità

In questa sezione analizziamo meglio le due metriche personalizzazione e riproducibilità considerando come ambito di riferimento disegni tecnici e scientifici.

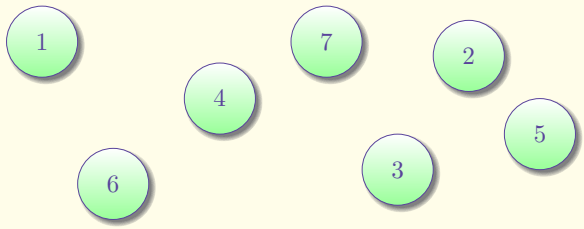
2.1 Personalizzazione

Questa indica la capacità di definizione e cambiamento delle proprietà estetiche dei vari elementi caratterizzanti la figura.

Prendiamo in esame la figura 1. Come si può notare, il gruppo di nodi è caratterizzato da uno stile omogeno che ne definisce il colore, la dimensione e l'ombreggiatura. Ogni proprietà è sempre sotto forma di un input di tipo *chiave-valore*. A volte sia il valore che la chiave possono essere sottointesi: per esempio, `circle` è una forma breve per `shape=circle` mentre in `draw` il valore non è specificato in quanto di default assume, dove presente, un colore indicato fra le opzioni (`violet` in questo caso), oppure il nero.

Prendiamo ora in esame la figura 2. Il risultato mostra i nodi con due differenti colorazioni, un trucco utile per distinguere elementi con proprietà differenti, come ad esempio gli utenti appartenenti a due piconet diversi in Bluetooth.

È importante notare il procedimento con cui si è ottenuta questa personalizzazione. Si osservi che gli stati, `stato 1` e `stato 2` sono del tutto analoghi a `stato` in figura 1, soltanto la colorazione è differente. La seconda osservazione riguarda il posizionamento: mentre in figura 1 si è usato un




```
\begin{tikzpicture}[stato/.style={%
  circle,top color=white,bottom color=green!40,
  circular drop shadow,draw,violet,minimum width=1cm}
]
\foreach \x/\text in {{0,2}/1,{1,0}/6, {2.5,1.2}/4,{4,2}/7,{5,0.2}/3,{6,1.8}/2,{7,0.7}/5}
\node[stato] at (\x) {\text};
\end{tikzpicture}
```

Figura 1: Un gruppo di nodi omogeneo

unico ciclo, in figura 2 è stato necessario spezzare il ciclo in due per poter assegnare i due diversi stili.

Per cambiare le proprietà degli elementi non è quindi stato necessario agire su ogni singolo elemento della figura.

2.2 Riproducibilità

Questa indica la capacità di riutilizzare interamente o parte di disegni realizzati precedentemente da estendere per creare la nuova figura. Quando si ha a disposizione un po' di materiale, la creazione di una figura risulta molto spesso il risultato di un opera di collage, non solo dei singoli elementi, ma soprattutto delle tecniche utilizzate durante il disegno. Il concetto è ben espresso da Andrew Stacey, di cui riporto un piccolo estratto di un intervento ben più lungo: [1]

> **Repetition** I do a lot of diagrams, but not so many that I'm a graphical designer. Every time that I do a new diagram there's a certain amount of *I've done something like this before, how did I do it?* with subsequent cut-and-pasting. When I use a graphical system then all of the *How did I do it?* information is lost. I can cut-and-paste actual objects, but it's rarely the objects that I want to copy - it's the how did I get that particular effect that I want to copy. When programming a diagram, all of this is laid out in an easily copy-able form.

Supponiamo ora di voler collegare fra loro gli elementi della figura 2 distinguendo i tipi di comunicazione. Il risultato è riportato in figura 3. Come si può notare, l'estensione al codice originale è molto semplice. Tuttavia, quello che è importante osservare è la tecnica utilizzata per l'inserimento delle frecce: il procedimento con i due cicli è del tutto analogo a quello già visto per il posizionamento dei nodi in figura 2.

1. http://tex.stackexchange.com/a/52547/13304d

3 Aspetti chiave del pacchetto pgfkeys

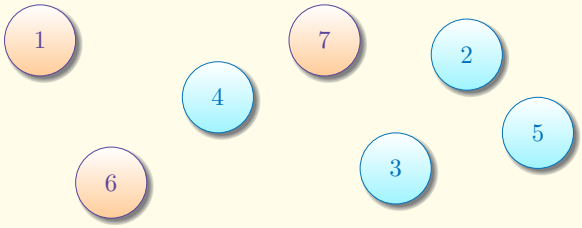
Numerosi pacchetti LaTeX implementano sistemi chiave-valore. In (Wright e Feueräsnger, 2009) gli autori forniscono una panoramica generale, evidenziando chiaramente le distinzioni fra il pacchetto pgfkeys e il resto dei pacchetti. Due sono le differenze sostanziali. La prima è la gestione delle chiavi gerarchica con struttura ad albero. La seconda è la capacità di definire *stili*, ossia raggruppare arbitrariamente un certo numero di parametri. Come abbiamo visto nel paragrafo 2, gli stili sono un aspetto fondamentale per garantire personalizzazione e riproducibilità dei disegni.

In questa sezione prendiamo in analisi gli aspetti più importanti di pgfkeys, utili non soltanto ad autori di pacchetti basati su Ti*k*Z, ma anche a tutti i normali utenti del pacchetto. Capire questi meccanismi, in modo particolare l'uso degli stili, permette di aumentare notevolmente la produttività nel disegno.

3.1 Le chiavi

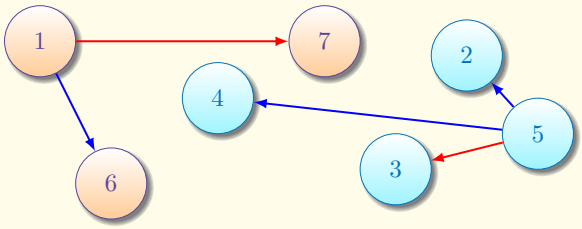
Il pacchetto pgfkeys definisce diversi tipi di chiavi che nel complesso permettono di creare interfacce anche molto complesse. In questo articolo si limita lo scopo dello studio agli strumenti per definire ed utilizzare nuove chiavi.

Le fasi di definizione e utilizzo, concettualmente distinte, devono in realtà essere affrontate insieme data la stretta relazione che intercorre fra le due. Infatti, a seconda di come si utilizza il valore della chiave nel codice, si possono distinguere due famiglie di metodi:




```
\begin{tikzpicture}[stato 1/.style={%
  circle,top color=white,bottom color=orange!40,
  draw,circular drop shadow,violet,minimum width=1cm,
  },
  stato 2/.style={%
    circle,top color=white,bottom color=cerulean!40,draw,
    circular drop shadow,royalblue,minimum width=1cm,
  },
]
\foreach \xa/\texta in {{0,2}/1,{1,0}/6, {4,2}/7}
\node[stato 1] (\texta) at (\xa) {\texta};
\foreach \xb/\textb in {{2.5,1.2}/4,{5,0.2}/3,{6,1.8}/2,{7,0.7}/5}
\node[stato 2] (\textb) at (\xb) {\textb};
\end{tikzpicture}
```

Figura 2: Un gruppo di nodi non omogeneo




```
\begin{tikzpicture}[stato 1/.style={%
  circle,top color=white,bottom color=orange!40,
  draw,circular drop shadow,violet,minimum width=1cm,
  },
  stato 2/.style={%
    circle,top color=white,bottom color=cerulean!40,draw,
    circular drop shadow,royalblue,minimum width=1cm,
  },
]
\foreach \xa/\texta in {{0,2}/1,{1,0}/6, {4,2}/7}
\node[stato 1] (\texta) at (\xa) {\texta};
\foreach \xb/\textb in {{2.5,1.2}/4,{5,0.2}/3,{6,1.8}/2,{7,0.7}/5}
\node[stato 2] (\textb) at (\xb) {\textb};

\begin{scope}[-latex,thick]
\foreach \sorgente/\destinazione in {1/7,5/3}
\draw[red] (\sorgente) edge (\destinazione);
\foreach \sorgente/\destinazione in {1/6,5/4,5/2}
\draw[blue] (\sorgente) edge (\destinazione);
\end{scope}
\end{tikzpicture}
```

Figura 3: Un gruppo di nodi non omogeneo con aggiunta

- la prima estrae il valore della chiave tramite il comando `\pgfkeysvalueof{`⟨*chiave*⟩`}`;
- la seconda inserisce direttamente nel codice una macro che assume il valore della chiave.

Utilizzando la prima famiglia, è preferibile effettuare la definizione attraverso:

```
\pgfkeys{/percorso/chiave/.initial = valore}
```

Utilizzando la seconda famiglia, invece, definire le chiavi con il precedente metodo potrebbe non andare bene. Per esempio, nelle prime versioni di S*a*-Ti*k*Z (Fiandrino, 2013), il meccanismo utilizzato era:

```
\pgfkeys{/tikz/.cd,%
      N/.initial=10,%
      N/.get=\N,%
      N/.store in=\N,%
}%
```

Come si può notare, anche qui era presente il costrutto `/.initial`. Tuttavia, questo metodo genera internamente tre macro:

- `\pgfk@/tikz/N` per `/.initial`, il quale a sua volta richiama internamente `\pgfkeyssetvalue`;
- `\pgfk@/tikz/N/.@cmd` per `/.get`, il quale a sua volta richiama internamente `\pgfkeysgetvalue`;
- `\N` generata da `.store in`.

Per evitare questo incoveniente, è sufficiente modificare di poco la definizione, in modo tale che ad ogni chiave venga associata una e una sola macro:

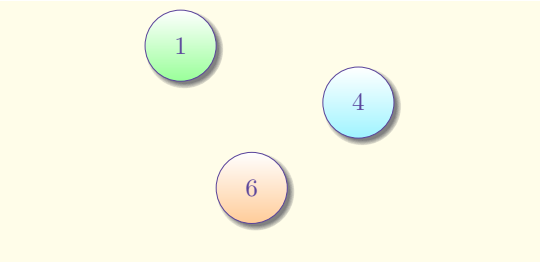
```
\pgfkeys{/tikz/.cd,%
      N=10,%
      N/.store in=\N,%
}%
```

L'ultima versione di S*a*-Ti*k*Z il meccanismo di gestione delle chiavi è implementato così.

3.2 Gli stili

Nel paragrafo 2 abbiamo già visto come gli stili permettano di ottenere personalizzazione e riproducibilità dei disegni. Tuttavia, le definizioni degli stili utilizzate per le figure 1, 2 e 3 sono estremamente semplici. In questa sezione, invece, analizziamo più a fondo come sfruttare meglio gli strumenti che il pacchetto pgfkeys mette a disposizione. Idealmente, questa sezione sviluppa ed estende i concetti già introtti in (Pantieri e Gordini, 2014).

L'aspetto chiave da tenere presente è che gli stili, come le macro in LaTeX *possono ricevere argomenti*. In questo modo, alcuni parametri dello stile possono variare volta per volta aumentando la flessibilità del codice. Per illustrare questo concetto, si utilizzerà come riferimento lo stile `stato` della figura 1:




```
\begin{tikzpicture}[stato/.style={%
  circle,top color=white,bottom color=#1,
  circular drop shadow,draw,violet,
  minimum width=1cm},
  stato/.default=green!40,
]
\node[stato] at (0,2) {1};
\node[stato=orange!40] at (1,0) {6};
\node[stato=cerulean!40] at (2.5,1.2) {4};
\end{tikzpicture}
```

Figura 4: Uso di uno stile con argomento

```
\tikzset{stato/.style={%
  circle,top color=white,
  bottom color=green!40,
  violet, circular drop shadow,
  draw,minimum width=1cm
  }
}
```

Supponiamo di voler essere liberi di cambiare il colore di riempimento, in questo caso `bottom color=green!40`. Logicamente, avrebbe poco senso creare n stili con n colori diversi. Invece, è molto più comodo fare in modo che la chiave `bottom color` venga impostata tramite argomento:

```
\tikzset{stato/.style={%
  circle,top color=white,
  bottom color=#1,
  violet, circular drop shadow,
  draw,minimum width=1cm
  },
  stato/.default=green!40,
}
```

La figura 4 illustra la flessibilità introdotta dal metodo. Si noti come grazie a `stato/.default=green`40! sia possibile dotare lo stile di un argomento di default: questo meccanismo permette di poter utilizzare:

```
\node[stato] at (0,2) {1};
```

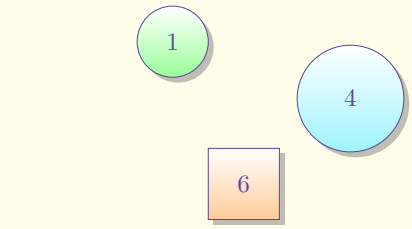
senza incorrere in errori.

In realtà, è perfettamente possibile inserire n argomenti agli stili, dove $n < 9$, con la sintassi:

```
stile/.style n args={num arg}{par}
```

dove il primo argomento è il numero di argomenti che si vogliono inserire e il secondo la lista di parametri che definiscono lo stile. Ad esempio:

```
\tikzset{stato/.style n args={3}{%
  circle,top color=#1,
```




```
\begin{tikzpicture}[%
stato/.style args={#1 of #2 colored in #3}{%
  forma e dimensione=#1 of #2, colore=#3,
},
stato/.default=%
circle of 1cm colored in green!40,
forma e dimensione/.style args={#1 of #2}{
  #1, drop shadow,
  minimum size=#2,
},
colore/.style={
  top color=white,
  bottom color=#1,
  violet, draw,
},]
\node[stato] at (0,2) {1};
\node[stato=%
rectangle of 1cm colored in orange!40]
 at (1,0) {6};
\node[stato=%
circle of 1.5cm colored in cerulean!40
] at (2.5,1.2) {4};
\end{tikzpicture}
```

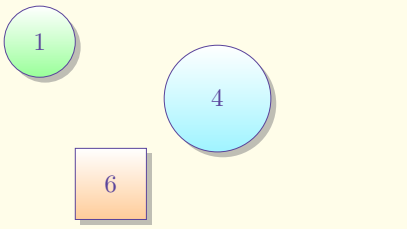
Figura 5: Stili con proprietà complesse

```
  bottom color=#2,
  violet, circular drop shadow,
  draw,minimum width=#3
  },
  stato/.default={white}{green!40}{1cm},
}
```

Attenzione: sebbene aumentando il numero di argomenti aumenta la flessibilità dello stile, usare un elevato numero di argomenti non è buona pratica in quanto il codice perde in leggibilità e facilità di utilizzo.

Un secondo metodo per migliorare personalizzazione e riproducibilità del codice è definire il più possibile *stili atomici*. Lo stile `stato`, per esempio non è *atomico* in quanto proprietà diverse come colore, forma e dimensione sono mescolate fra loro. Si può rendere più funzionale `stato` in questo modo:

```
\tikzset{stato/.style={%
    forma e dimensione,
    colore,
  },
  stato/.default=%
  circle of 1cm colored in green!40,
  forma e dimensione/.style args={#1 of #2}{
    #1, drop shadow,
```




```
\begin{tikzpicture}[stato/.style={%
    forma e dimensione,
    colore,
  },
  forma e dimensione/.style={
    circle, drop shadow,
    minimum size=1cm,
  },
  colore/.style={
    top color=white,
    bottom color=green!40,
    violet, draw,
  },]
\node[stato] at (0,2) {1};
\node[forma e dimensione/.append style={%
 rectangle}, colore/.append style={
 bottom color=orange!40,
},stato]
 at (1,0) {6};
\node[forma e dimensione/.append style={%
 minimum size=1.5cm},
colore/.append style={
bottom color=cerulean!40,
},stato] at (2.5,1.2) {4};
\end{tikzpicture}
```

Figura 6: Alterare le proprietà degli stili

```
    minimum size=#2,
  },
  colore/.style={
    top color=white,
    bottom color=#1,
    violet, draw,
  },
}
```

Il risultato di questa operazione è riportato in figura 5 utilizzando la proprietà degli stili di definire appositi *pattern* di argomenti. Occorre osservare che questa pratica è un primo passo verso la definizione di stili con proprietà atomiche, ma decisamente complesso. Oltretutto, fornisce soltanto una parvenza di flessibilità: per cambiare una delle proprietà, colore ad esempio, è infatti necessario riscrivere interamente il pattern di argomenti.

Invece, sfruttando un'altra proprietà degli stili si possono ottenere soluzioni molto più semplici. Come è noto, gli stili vengono definiti come una lista di parametri chiave-valore. È importante notare che in caso due parametri definiscano diversi valori per la stessa chiave, solo l'ultimo nella lista viene effettivamente considerato. Nel caso di riferimento, quindi, è sufficiente la sintassi:

```
\tikzset{stato/.style={%
    forma e dimensione,
    colore,
  },
  forma e dimensione/.style={
    circle, drop shadow,
    minimum size=1cm,
  },
  colore/.style={
    top color=white,
    bottom color=green!40,
    violet, draw,
  },
}
```

La figura 6 illustra come utilizzare nella pratica il metodo descritto per alterare le proprietà degli stili. Si noti che è necessario posporre lo stile `stato` come ultimo parametro. `stato`, infatti, esegue entrambi gli stili `forma e dimensione` e `colore`. Quindi, è necessario che le modifiche eseguite a due sotto-stili vengano rese operative *prima* dell'esecuzione dello stile di alto livello.

4 Conclusione

In questo articolo si sono analizzati gli strumenti principali che Ti*k*Z mette a disposizione per ottenere alta personalizzazione e riproducibilità dei disegni. In particolare modo, si è focalizzata l'attenzione sul pacchetto pgfkeys considerando sia il sistema chiave-valore che gli stili, una delle peculiarità del pacchetto.

Dal punto di vista dell'utente normale del pacchetto, imparare ad impostare correttamente gli stili è importante per migliorare la produttività nel disegno. Chi invece desidera *programmare* con Ti*k*Z, quindi creare nuovi pacchetti o librerie, deve quasi assolutamente confrontarsi con l'uso del sistema chiave-valore.

Riferimenti bibliografici

Fiandrino, C. (2013). «S*a*-TikZ: a library to draw switching architectures». *ArsTEXnica*, (15), pp. 4–10. URL `http://www.guit.sssup.it/arstexnica/`.

Pantieri, L. e Gordini, T. (2014). *L'arte di disegnare con LaTeX*. URL `http://www.lorenzopantieri.net/LaTeX_files/ArteTikZ.pdf`.

Tantau, T. (2013). *The TikZ and PGF Packages*. URL `http://www.ctan.org/pkg/pgf`. Consultabile con: `texdoc tikz`

Wright, J. e Feueräsnger, C. (2009). «Implementing key-value input: An introduction». *TUGboat*, **30** (1), pp. 110–122.

A Appendice

I colori `cerulean` e `royalblue` non sono presenti fra la collezione standard. Tramite l'opzione dvipsnames di xcolor si può accedere ai colori `Cerulean` e `Royalblue`, perciò si riporta in questa appendice le definizioni di `cerulean` e `royalblue`:

```
\definecolor{royalblue}{cmyk}{1,0.50,0,0}
\definecolor{cerulean}{cmyk}{0.94,0.11,0,0}
```


▷ Claudio Fiandrino
University of Luxembourg
`claudio dot fiandrino at uni dot lu`