Numero 13
Aprile 2012

ArsTEXnica

Rivista italiana di TEX e LATEX

GUIT

http://www.guit.sssup.it/arstexnica

TEXnica Ars

GUIT – Gruppo Utilizzatori Italiani di TEX

ArsTEXnica è la pubblicazione ufficiale del GUIT

ArsTEXnica è la prima rivista italiana dedicata a TEX, a LATEX ed alla tipografia digitale. Lo scopo che la rivista si prefigge è quello di diventare uno dei principali canali italiani di diffusione di informazioni e conoscenze sul programma ideato quasi trent'anni fa da Donald Knuth.

Le uscite avranno, almeno inizialmente, cadenza semestrale e verranno pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno conterrà gli Atti del Convegno Annuale del GUIT, che si tiene in quel periodo.

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file di esempio (`.tex`). Tutto il materiale è reperibile all'indirizzo web della rivista.

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di TEX e LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorials, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Chiunque, poi, volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo:

`arstexnica@sssup.it`.

Nota sul Copyright

Associarsi a GUIT

Fornire il tuo contributo a quest'iniziativa come membro, e non solo come semplice utente, è un presupposto fondamentale per aiutare la diffusione di TEX e LATEX anche nel nostro paese. L'adesione al Gruppo prevede una quota di iscrizione annuale diversificata: 30,00 € soci ordinari, 20,00 (12,00) € studenti (junior), 75,00 € Enti e Istituzioni.

Indirizzi

Gruppo Utilizzatori Italiani di TEX:
c/o Ufficio Statistica
Scuola Superiore Sant'Anna
Piazza Martiri della Libertà 33
56127 Pisa, Italia.
`http://www.guit.sssup.it`
`guit@sssup.it`

Redazione ArsTEXnica:
`http://www.guit.sssup.it/arstexnica/`
`arstexnica@sssup.it`

Codice ISSN 1828-2369

Stampata in Italia
Pisa: 15 Aprile 2012

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 13, Aprile 2012

Gruppo Utilizzatori Italiani di TEX

Editoriale

Gianluca Pignalberi

Avete il primo numero del 2012 di ArsTEXnica tra le mani. Lo avete ricevuto con ragionevole puntualità. Contiene pochi articoli di notevole qualità.

Ripercorriamo tutti gli avvenimenti interni ed esterni al GUIT che ci hanno condotto a questo numero.

Durante il GUIT *meeting 2011* tutti hanno posto l'attenzione sulla necessità di lavorare al sito nuovo. È stato costituito un comitato web coordinato da Roberto Giacomelli e questo ha lavorato bene, riempiendo in breve tempo il sito di contenuti. Ancora in questi giorni (siamo alla fine di marzo) Roberto e i suoi collaboratori stanno testando altre funzionalità di Joomla! per aggiungere caratteristiche necessarie al sito e all'associazione. Insomma, con calma ma metodicamente il GUIT sta effettuando la migrazione dei contenuti dal vecchio sito, ormai pressoché dismesso, al nuovo.

All'inizio dell'anno, con la pubblicazione di *Edizioni Critiche. Guida alla composizione con il proprio computer*, ha visto la luce la collana TEXNOLOGIE. L'editrice CompoMat di Silvia Maschio ha assunto l'onere di pubblicare una collana di testi perlopiù inediti dedicati a TEX. Mi pare superfluo dire che *tutto* il libro è composto in LATEX, copertina compresa (tranne il codice a barre, ahimè). La copertina è riprodotta di seguito.

Alla fine di febbraio è uscito in libreria *Microeconomia* di Hal R. Varian, edito dalla Libreria Editrice Cafoscarina. Un rappresentante dell'editrice ha contattato il GUIT per cercare un esperto di TEX per convertire il formato originale del libro (TEX) nel più adeguato LATEX e produrre contestualmente la nuova edizione (la settima italiana). Il GUIT è formato da ottimi elementi ed è in grado di catalizzare la richiesta di esperti di TEX e delle tecnologie derivate da parte del mondo editoriale. Spero solo che finalmente decolli la sezione Lavoro presente sul nuovo sito.

Arriviamo ai contenuti della rivista che vi accingete a leggere. Abbiamo detto pochi articoli... vedrete.

Apre il numero Claudio Fiandrino col suo *Graphviz e TikZ*. L'autore espone le tecniche per trasformare i grafi scritti nel linguaggio di Graphviz in figure TikZ. Non c'è bisogno di dire che Graphviz aumenta infinitamente la velocità di ottenimento dei grafi essendo un programma dedicato con un suo linguaggio di descrizione. L'esportazione verso il formato TikZ ne aumenta anche la qualità tipografica.

L'ormai affiatato trio Pignalberi, Schirone e Leal propone un'integrazione del precedente articolo dedicato ai grecisti classici. Sciolgono alcuni argomenti dati per irrisolti proponendo le soluzioni presenti sul CTAN relative proprio a quei problemi.

Claudio Beccari contribuisce al numero con un articolo sull'estensione di LATEX alle lingue romanze usate nelle regioni alpine. Tralasciando la completezza e l'erudizione dell'articolo sul versante linguistico (una perla tutta da leggere), l'articolo è fruibile per la moltitudine di contenuti: gestione delle lingue da parte di LATEX, descrizione della lingua e file di pattern, proposta di integrazione sia in babel che in polyglossia.

L'ultimo articolo del numero segna il "ritorno" in attività (se mai si è fermato) di Enrico Gregorio. L'editor incaricato dell'articolo lo ha definito, verosimilmente a ragione, uno dei primi vagiti italiani di LATEX3. Enrico ci spiega come scrivere un pacchetto per LATEX3. Il pacchetto in questione è kantlipsum che io ho scoperto per caso (dovevo effettuare dei test di impaginazione) proprio pochi giorni prima di ricevere l'articolo.

Non aggiungo altro per non togliere ulteriore tempo alla lettura. Alla prossima.

▷ Gianluca Pignalberi
`g dot pignalberi at alice dot it`

Graphviz e Ti*k*Z

Claudio Fiandrino

Sommario

Graphviz è un potente software per disegnare grafi. Questo articolo proverà a spiegare come esportare in maniera semplice tali grafi in figure Ti*k*Z.

Abstract

Graphviz is a very powerful tool to draw graphs. This article tries to explain how to export such graphs as a Ti*k*Z picture in a very simple way.

Premessa

Ubuntu 11.04 sarà il sistema operativo di riferimento nell'articolo. Tutti i comandi per l'installazione e la creazione del codice Ti*k*Z saranno impartiti dal terminale. Le istruzioni sono molto semplici e chi ha poca dimestichezza con tale strumento trova comunque in rete alcune guide introduttive molto ben curate. A tale proposito, si riporta `http://wiki.ubuntu-it.org/AmministrazioneSistema/RigaDiComando`.

1 Introduzione

La creazione di grafi con Graphviz avviene mediante un linguaggio specifico, chiamato *dot*, la cui sintassi è davvero semplice ed intuitiva. La sezione 3 ne darà una breve introduzione. L'uso di Graphviz è largamente diffuso per analizzare, creare ed utilizzare i grafi. Un esempio pratico è la creazione di nodi di rete per simulare attacchi di pirati informatici (Zhang e Wu, 2008).

Il software che *elabora* il linguaggio dot creando il relativo codice Ti*k*Z si chiama *dot2tex* e permette di ottenere anche il codice per PGF e PSTricks. I seguenti programmi sono requisiti fondamentali per *dot2tex*:

- python, versione 2.4 o successive;
- pyparsing versione 1.4.8 (raccomandata) o successive;
- Graphviz.

Sono anche richiesti i pacchetti LaTeX Preview e Ti*k*Z/PGF. Nella sezione 2 saranno illustrati i passi per installare velocemente tutti i componenti mentre nella sezione 4 sarà preso in esame il funzionamento di *dot2tex*.

2 Installazione

Python è già disponibile con l'installazione predefinita di Ubuntu (web, d); in caso contrario è sufficiente digitare da terminale:

```
sudo apt-get install python
```

Per installare *pyparsing* (web, c) e *dot2tex* (web, a) converrà usare uno strumento estremamente comodo, *easy_install*:

```
sudo apt-get install python-setuptools
```

I comandi per installare *pyparsing* e *dot2tex* sono rispettivamente:

```
sudo easy_install pyparsing
```

e

```
sudo easy_install dot2tex
```

Graphviz (web, b) necessita di una lunga lista di pacchetti: graphviz, graphviz-doc, graphviz-dev, libgraphviz-dev, libgraphviz-perl, libgv-lua, libgv-perl, libgv-php5, libgv-guile, libgv-python, libgv-ruby, libgv-tlc, libgv-ocaml. I suddetti possono essere installati con il solito comando:

```
sudo apt-get install <lista-pacchetti>
```

I pacchetti LaTeX Preview e Ti*k*Z/PGF sono supportati dalle distribuzioni TeX Live e MiKTeX. È consigliabile l'uso di TeX Live; una guida che ne illustra l'installazione su Ubuntu è (Gregorio, 2010).

3 Introduzione al linguaggio dot

Il codice 1 mostra un esempio di grafo realizzato con *dot*.

Codice 1: il primo esempio.

```
digraph G {
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
}
```

Il grafo `G` è composto da tre nodi e tre archi; dichiarato con la parola chiave `digraph`, tutto il codice che lo definisce deve essere racchiuso fra { }. È evidente l'assenza di *dichiarazioni*: i nodi e gli archi vengono creati direttamente. Una riga di codice finisce sempre con "`;`". Il simbolo `->` caratterizza l'arco che unisce due nodi. In questo caso l'arco ha una *direzione*, ma è possibile anche

definire archi *senza* direzione. Gli archi possono avere un'etichetta o meno; l'etichetta deve essere inserita fra parentesi quadre con la parola chiave `label`. L'esempio mostrato dal codice 1 (Il primo esempio) sarebbe notevolmente più semplice senza etichette, come mostra il codice 2 (Il primo esempio semplificato).

Codice 2: il primo esempio semplificato.

```
digraph G {
    1->2->3->1;
}
```

Per scrivere il codice del grafo è necessario creare un file con estensione `.dot`: qualsiasi editor di testo può andare bene; gli esempi qui riportati sono stati creati con *gedit*.

Ulteriori informazioni sono reperibili sul sito con la documentazione ufficiale: `http://www.graphviz.org/Documentation.php`

4 Come funziona

Per ottenere il codice di una figura non esiste un comando standard funzionale in ogni caso. A seconda dell'obbiettivo alcuni comandi sono più *funzionali* rispetto ad altri: per esempio, i comandi `circo` e `neato` producono un risultato diverso a partire dallo stesso codice.

4.1 Il primo esempio

La procedura che descrive la realizzazione del primo esempio è valida anche per tutti gli altri ed è la seguente:

1. Creare una directory di nome `ex1` sul desktop.
2. Creare all'interno di tale cartella un file nominato `ex1.dot` in cui incollare il codice mostrato nel codice 1 (Il primo esempio).
3. Aprire il terminale (fare click sul pulsante Applicazioni della dashboard, quindi digitare `terminale` nel campo *cerca*) e scrivere:

   ```
   cd Desktop/ex1/
   ```

   per *spostarsi* all'interno della directory.
4. Creare il file `.tex` attraverso il comando:

   ```
   dot2tex --preproc ex1.dot\
   | dot2tex > ex1.tex
   ```

   Il carattere \, usato qui per esigenze tipografiche, serve comunque a far capire al terminale che il comando è stato spezzato su più righe e l'invio dopo \ non indica l'esecuzione del comando.
5. Digitare:

   ```
   pdflatex ex1.tex
   evince ex1.pdf
   ```

   I due comandi compileranno il file `.tex` creato in precedenza e mostreranno il risultato, qui riportato nella figura 1, con il visualizzatore di pdf indicato.

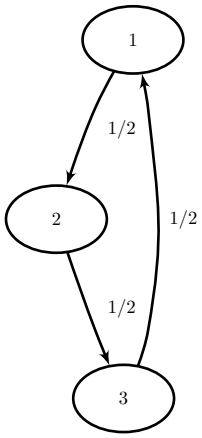


Figura 1: il primo esempio

Il documento LaTeX creato nel punto 4 è *completo*: utilizza la classe *article*, ha un proprio preambolo ed è composto da una sola pagina in cui viene inserita l'immagine.

È possibile esportare molto facilmente la figura copiando il codice, ma bisogna fare attenzione a copiare anche i pacchetti e le librerie scritti nel preambolo e necessari alla figura. In questo caso:

```
\usepackage{tikz}
\usetikzlibrary{snakes,arrows,shapes}
```

4.2 Le opzioni sul tipo di codice

Il comando *dot2tex* permette di ottenere:

- il codice PGF – opzione `fpgf`;
- il codice Ti*k*Z – opzione `ftikz`;
- il codice PSTricks – opzione `fpst`.

In generale, non specificare un'opzione fa sì che la figura sia *scritta* in PGF. Tuttavia, il medesimo risultato è ottenibile digitando:

```
dot2tex -fpgf ex1.dot > ex1_pgf.tex
```

La generazione del codice Ti*k*Z avviene con:

```
dot2tex -ftikz ex1.dot > ex1_tikz.tex
```

mentre il seguente comando:

```
dot2tex -fpst ex1.dot > ex1_pst.tex
```

è necessario per ottenere un'immagine PSTricks.

I comandi riportati possono essere applicati anche al primo esempio.

Codice 3: il secondo esempio.

```
digraph G {
    a_1->b_2 [label="1/2"];
    b_2->c_3 [label="1/2"];
    c_3->a_1 [label="1/2"];
}
```

4.3 Le opzioni sulle etichette dei nodi

Si consideri il codice mostrato in codice 3 (Il secondo esempio).

La procedura indicata nella sezione 4.1 implica che il file `ex2.dot` sia presente in una directory di nome `ex2` nel desktop. Dal codice evinciamo come sia facile assegnare un'etichetta ad un nodo: è sufficiente inserirla all'atto di creazione dell'arco (un secondo metodo sarà analizzato più avanti).

Esistono tre opzioni per caratterizzare l'output:

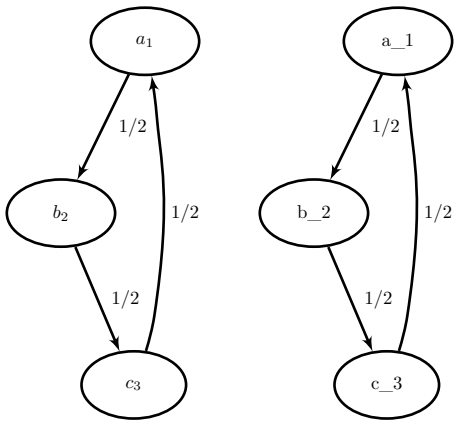
- in modalità matematica (opzione `tmath`) l'etichetta sarà inserita fra $ $;
- in modalità *verbatim* (opzione `tverbatim`) l'interprete farà sì che i caratteri speciali di LaTeX vengano riconosciuti;
- in modalità *raw* (opzione `traw`) la label non viene processata in alcun modo; se sono presenti caratteri speciali è possibile incorrere in errori.

Il comando:

```
dot2tex -tmath ex2.dot > ex2.tex
```

genera il risultato mostrato nella figura 2a, diverso da quello riportato in figura 2b ottenuto invece con:

```
dot2tex -tverbatim ex2.dot > ex2.tex
```



(a) label opzione *tmath* (b) label opzione *tverbatim*

Figura 2: esempi con le diverse opzioni label

Il comando:

```
dot2tex -traw ex2.dot > ex2.tex
```

fornisce invece un errore perché il carattere _ è un carattere speciale.

Il secondo metodo che permette di specificare delle etichette per i nodi è più *pesante* di quello visto ora in quanto è necessario scrivere più righe di codice nel file `.dot` per ottenere lo stesso risultato. Ha però il grande vantaggio di essere indipendente dalle opzioni precedentemente elencate. Il motivo è molto semplice: grazie alla parola chiave `texlbl`, l'opzione da utilizzare viene definita a priori nel file `.dot` quindi è indispensabile scrivere più righe di codice.

Il codice 4 (Una variante del secondo esempio) riporta la variante del secondo esempio con le modifiche illustrate. In questo caso è sufficiente

Codice 4: una variante del secondo esempio.

```
digraph G {
    1 [texlbl="$a_1$"];
    2 [texlbl="$b_2$"];
    3 [texlbl="$c_3$"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
}
```

digitare

```
dot2tex ex2.dot > ex2.tex
```

per creare il file `ex2.tex` ed ottenere il risultato mostrato nella figura 2a.

4.4 Personalizzare le etichette

È possibile personalizzare il colore delle etichette attraverso la parola chiave `lblstyle`. Più in generale, è possibile introdurre tutti i comandi standard usati in Ti*k*Z. La modifica del file `ex2.dot` come riportato nel codice 5 (Il secondo esempio con label rossa) e l'immissione del comando:

```
dot2tex ex2.dot > ex2.tex
```

genera il risultato mostrato nella figura 3.

Codice 5: il secondo esempio con label rossa.

```
digraph G {
  1 [texlbl="$a_1$", lblstyle="red"];
  2 [texlbl="$b_2$"];
  3 [texlbl="$c_3$"];
  1->2 [label="1/2"];
  2->3 [label="1/2"];
  3->1 [label="1/2"];
}
```

La modifica del codice 5 riportata nel codice 6 (Il secondo esempio con comandi Ti*k*Z) illustra come è possibile introdurre altri comandi Ti*k*Z.

Come prima, il comando:

```
dot2tex ex2.dot > ex2.tex
```

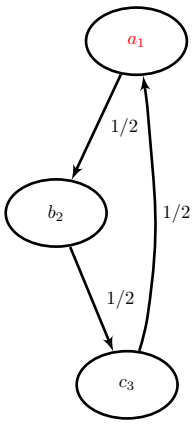


FIGURA 3: il secondo esempio con label rossa

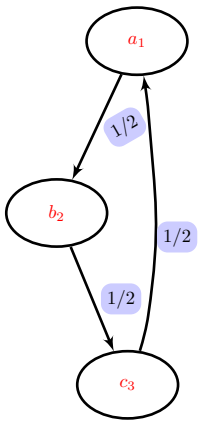


FIGURA 4: il secondo esempio con comandi TikZ

CODICE 6: il secondo esempio con comandi TikZ.

```
digraph G {
1 [texlbl="$a_1$",lblstyle="red"];
2 [texlbl="$b_2$",lblstyle="red"];
3 [texlbl="$c_3$",lblstyle="red"];
1->2 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20,rotate=30"];
2->3 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20"];
3->1 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20, below=0.1cm"];
}
```

genera il risultato mostrato nella figura 4.

4.5 Personalizzare i nodi

La personalizzazione dei nodi può avvenire caratterizzandone:

- la forma:
  - circonferenza;
  - ellisse;
  - rettangolo;
- i colori:
  - dei bordi;
  - del riempimento;
- i font.

La parola chiave `style` e il contestuale uso dei comandi usuali di TikZ consentono di personalizzare i colori e il font. La forma viene definita, invece, con la parola chiave `shape`. L'esempio `ex3.dot` illustra tali caratteristiche attraverso il codice 7 (IL TERZO ESEMPIO). C'è da notare che le impostazioni date con la parola chiave `node` diventano le impostazioni di default; possono comunque essere cambiate per un singolo nodo.

CODICE 7: il terzo esempio.

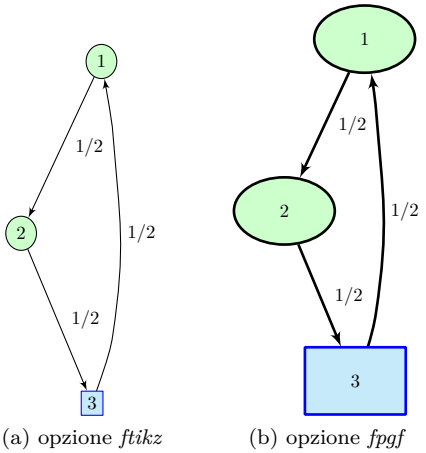
```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

È utile mostrare come la compilazione del codice 7 mediante i comandi TikZ o PGF faccia ottenere due risultati diversi. Le opzioni da utilizzare sono quelle descritte nella sottosezione 4.2:

```
dot2tex -ftikz ex3.dot > ex3.tex
```

```
dot2tex ex3.dot > ex3_pgf.tex
```

Le figure 5a e 5b mostrano tale diversità, nonostante entrambe presentino lo stesso fattore di scala (non sono chiare le motivazioni di tale comportamento).



(a) opzione *ftikz* (b) opzione *fpgf*

FIGURA 5: esempi delle le diverse opzioni di output

Si ipotizzi sia necessario caratterizzare diversi tipi di nodi in un grafo, ad esempio una rete di sensori in un'area geografica che monitori la tempe-

ratura oppure una rete di antenne accese e spente. *Colorando* in modo diverso i nodi sarebbe possibile visualizzare immediatamente quali antenne sono attive oppure quali sensori misurano un aumento della temperatura.

Risolvere questo problema caratterizzando i nodi come fatto in precedenza sarebbe poco pratico. L'impostazione di default unica implicherebbe di dover agire manualmente per modificare i nodi del secondo tipo. Questo problema non viene affrontato nella documentazione ufficiale, anche se è opinione dell'autore che rivesta notevole importanza. La soluzione proposta utilizza la parola chiave `d2tfigpreamble`; il codice 8 (Il quarto esempio) riporta un esempio in cui viene illustrato tale metodo.

Codice 8: il quarto esempio.

```
digraph G {
 d2tfigpreamble="
 \tikzstyle{cold}=\ [draw=blue!50,
 very thick,fill=blue!20],
 \tikzstyle{hot}=\ [draw=red!50,
 very thick,fill=red!20]";
 A [style="cold"];
 B [style="cold"];
 C [style="hot"];
 D [style="hot"];
 A->B->D;
 A->C->D;
 D->C;
 D->B->A;
 B->B [topath="loop left"];
 C->B;
}
```

È molto importante notare la sintassi: entrambe le definizioni degli stati sono racchiuse fra *doppi apici*; è necessario inserire un *backslash* prima della parola chiave `tikzstyle` e prima di definire le caratteristiche del nodo fra [ ]. Inoltre, come succede per Ti*k*Z, i vari comandi devono essere separati da una *virgola*.

Naturalmente è possibile definire anche più di due *stati* con questo metodo.

Rimane da analizzare come comandi diversi di Graphviz generino risultati diversi partendo dal medesimo esempio. Il comando:

```
circo -Txdot ex4.dot |\
dot2tex -ftikz > ex4.tex
```

genera il grafo riportato nella figura 6.

Utilizzando invece il comando:

```
circo -Txdot ex4.dot |\
dot2tex -ftikz --styleonly > ex4.tex
```

e seguendo sempre la procedura illustrata nella sottosezione 4.1 il risultato ottenuto, riportato nella figura 7, sarà diverso.

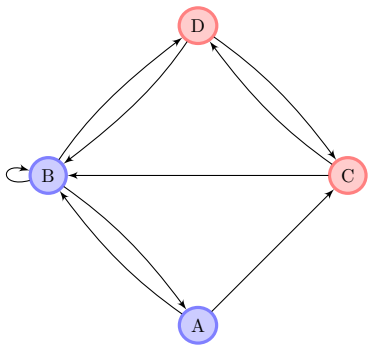


Figura 6: il quarto esempio comando *circo*

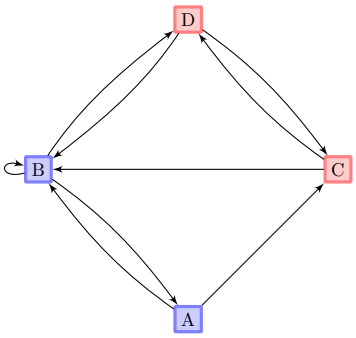


Figura 7: il quarto esempio opzione *styleonly*

È facile intuire che il motivo di tale differenza è dovuto all'opzione `styleonly`. Infatti, se l'opzione è assente, il codice Ti*k*Z che definisce un nodo è:

```
\node (A) at (108.21bp,19.5bp)
  [draw,ellipse,cold] {A};
```

Quando invece l'opzione è attiva il codice è:

```
\node (A) at (108.21bp,19.5bp)
  [cold] {A};
```

In ultimo, il comando:

```
neato -Txdot ex4.dot |\
dot2tex -ftikz --styleonly > ex4.tex
```

genera il grafo visibile nella figura 8.

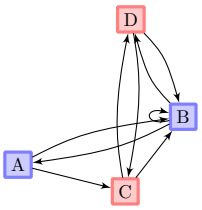


Figura 8: il quarto esempio comando *neato*

Il confronto tra le figure 6 e 8 fa dedurre che il comando `circo` è preferibile per i grafi con topologie circolari perché utilizza un algoritmo che *posiziona* i nodi a dofferenza di `neato`. Il manuale di Graphviz conferma l'osservazione.

4.6 Personalizzare gli archi

Esistono due metodi per personalizzare gli archi: mediante la sintassi `edge[style="..."]` oppure attraverso la parola chiave `topath`.

Il codice 9 (Il quinto esempio) analizza il primo approccio.

Codice 9: il quinto esempio.

```
digraph G {
 d2tfigpreamble="
 \tikzstyle{cold}=\ [draw=blue!50,
 very thick,fill=blue!20],
 \tikzstyle{hot}=\ [draw=red!50,
 very thick,fill=red!20]";
 A [style="cold"];
 B [style="cold"];
 C [style="hot"];
 D [style="hot"];
 edge [style="snake=zigzag"];
 A->B->D->C->A;
 edge [style="snake=snake"];
 A->D;
 C->B;
}
```

Si noti come la sintassi che caratterizza l'arco deve essere inserita prima del codice di definizione. Il risultato grafico mostrato nella figura 9 è stato ottenuto con il comando:

```
circo -Txdot ex5.dot |\
dot2tex -ftikz -s > ex5.tex
```

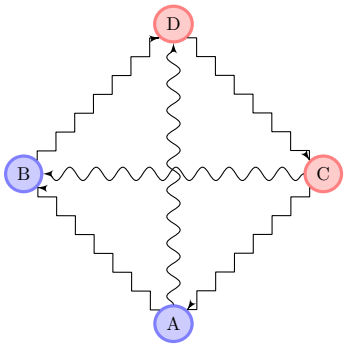


Figura 9: il quinto esempio

L'opzione `-s` deve essere specificata per poter riconoscere i percorsi di tipo *snake*. Il comando

```
\usetikzlibrary{snakes}
```

richiede la libreria di Ti*k*Z *snakes*, obsoleta e recentemente sostituita da *decorations*. Questo è il motivo per cui le forme selezionabili sono poche. Essere vincolati a una libreria obsoleta non permette neanche l'uso di opzioni estremamente interessanti come *pre/post length*. Queste permettono di personalizzare il punto di inizio e fine del cambiamento di forma dell'arco. Ovviamente è possibile editare il codice Ti*k*Z in un secondo momento, ad esempio inserendo la libreria *decorations* e il codice relativo alla forma dell'arco. È sufficiente individuare la parte di codice giusta:

```
\draw [->,snake=zigzag] (C) -- (A);
\draw [->,snake=snake] (A) -- (D);
```

Il codice 10 (Il sesto esempio) permette invece l'analisi del metodo che fa uso della parola chiave `topath`.

Codice 10: il sesto esempio.

```
digraph G {
 d2tfigpreamble="
 \tikzstyle{cold}=\ [draw=blue!50,
 very thick,fill=blue!20],
 \tikzstyle{hot}=\ [draw=red!50,
 very thick,fill=red!20]";
 A [style="cold"];
 B [style="cold"];
 C [style="hot"];
 D [style="hot"];
 A->B->D->C->A[topath="bend left=20"];
 A->C->D->B->A[topath="bend left=20"];
 B->B [topath="loop left"];
}
```

Questo esempio è diverso dal precedente poiché mostra una proprietà diversa: non viene cambiata la forma dell'arco ma l'angolazione di partenza ed arrivo dell'arco dai/nei nodi. Entrambi gli approcci permettono l'uso dei classici comandi Ti*k*Z essendo, pertanto, equivalenti. È dunque possibile inserirli contemporaneamente nello stesso codice.

Il comando:

```
circo -Txdot ex6.dot |\
dot2tex -ftikz > ex6.tex
```

permette di ottenere il risultato riportato nella figura 10.

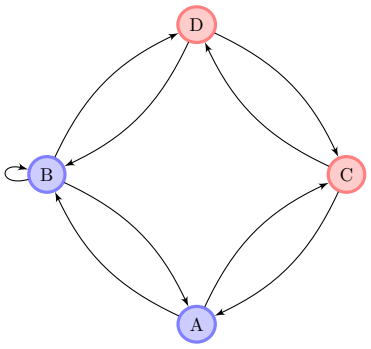


Figura 10: il sesto esempio

5 Considerazioni finali

Questa sezione cercherà di rispondere alla domanda "perché non scrivere il codice del grafo direttamente con Ti*k*Z?"

Non esiste una risposta univoca. I vantaggi del metodo illustrato sono evidenti quando occorra importare in un documento dei grafi con un certo numero di nodi ed archi. Se invece il grafo ha due o tre nodi è forse più conveniente scrivere direttamente il codice Ti*k*Z. La figura 11 è un esempio di questo tipo e raffigura una semplice catena di Markov. Il sorgente è riportato in codice 11 (GRAFO COMPOSTO CON TI*K*Z).

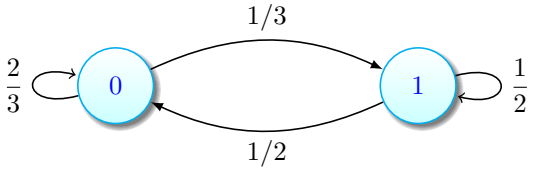


FIGURA 11: un esempio di grafo composto direttamente con Ti*k*Z

CODICE 11: un grafo composto direttamente con Ti*k*Z.

```
\begin{tikzpicture}[-latex,auto,
  node distance=4cm,on grid,semithick,
  state/.style={circle, draw,
  top color=white,
  bottom color=cyan!20, cyan,
  circular drop shadow,text=blue,
  minimum width=1cm}]
% Nodes
\node[state] (A) {$0$};
\node[state] (B) [ right=of A] {$1$};
% Edges
\path (A) edge [loop left]
  node{$\dfrac{2}{3}$} (A);
\path (B) edge [loop right]
  node{$\dfrac{1}{2}$} (B);
\path (B) edge [bend left=25]
  node[below]{$1/2$} (A);
\path (A) edge [bend right=-25 ]
  node[above]{$1/3$} (B);
\end{tikzpicture}
```

C'è da notare, contrariamente a quanto necessario col codice *dot*, la necessità di indicare la posizione dei nodi e che la sintassi del codice è leggermente più complicata. Si può quindi affermare che il linguaggio *dot*:

- è più intuitivo;
- non richiede di posizionare manualmente i nodi perché lo fa autonomamente in base al comando usato (quindi all'algoritmo opportuno);
- permette di ottenere la stessa qualità di Ti*k*Z grazie alla possibilità di personalizzare gli *stati* attraverso i preamboli.

6 Conclusioni

Questo articolo ha analizzato un metodo per esportare grafi creati con Graphviz in immagini prodotte con codice Ti*k*Z, PGF e PSTricks.

Dopo aver illustrato come installare i programmi necessari, sono stati presentati numerosi esempi (codice e risultato grafico) per discutere i punti di forza e di debolezza del metodo.

L'articolo non è esaustivo perché ha discusso solo gli aspetti di base dei comandi introdotti e non ha preso in considerazione il pacchetto *dot2texi*. In teoria, grazie all'ambiente dot2tex, l'uso del pacchetto dovrebbe essere preferibile al metodo illustrato in questo articolo: il codice dot viene inserito nell'ambiente citato e produce automaticamente la figura. Le compilazioni fatte sugli esempi tratti dal manuale di *dot2texi* danno sempre un errore in corrispondenza della linea di apertura dell'ambiente, errore che comunque non pregiudica l'ottenimento del grafo finale, seppur leggermente diverso da quanto riportato nel citato manuale.

Riferimenti bibliografici

(a). «dot2tex». URL `http://www.fauskes.net/code/dot2tex/`.

(b). «Graphviz». URL `http://www.graphviz.org/Download.php`.

(c). «Pyparsing». URL `http://pyparsing.wikispaces.com/`.

(d). «Python». URL `http://www.python.org/`.

GREGORIO, E. (2010). «Installare TEX live 2010 su ubuntu». *ArsTEXnica*, (10), pp. 7–13. URL `http://www.guit.sssup.it/arstexnica/`.

ZHANG, T. e WU, C. (2008). «Network security analysis based on security status space». *Web-Age Information Management*. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4597065`.

▷ Claudio Fiandrino
`http : / / claudiofiand rino . altervista.org`
`claudio dot fiandrino at gmail dot com`

Ancora sugli strumenti per grecisti classici

Gianluca Pignalberi, Salvatore Schirone, Jerónimo Leal

Sommario

L'articolo esamina due argomenti lasciati aperti nel nostro precedente *Introduzione agli strumenti per grecisti classici*, segnatamente la traslitterazione di testi greci con XeLaTeX e la composizione parallela di testi con traduzione a fronte. Verranno date una o più soluzioni per ognuno dei due argomenti.

Abstract

This paper inspects two topics left unsolved in our previous one, *Introduzione agli strumenti per grecisti classici*, namely the transliteration of Greek texts with XeLaTeX and the composition and the translation with parallel text into another language. One or more solutions to both topics will be given.

1 Introduzione

Nel nostro precedente articolo *Introduzione agli strumenti per grecisti classici* (Pignalberi *et al.*, 2011) abbiamo descritto alcuni metodi e strumenti per comporre i testi tipici dei filologi, in particolare dei grecisti. Non siamo stati però in grado di fornire i metodi per trattare il greco traslitterato con XeLaTeX e per ottenere le note a piè di pagina posizionate correttamente nei documenti con testi paralleli su due colonne.

Durante la stesura del libro Leal e Pignalberi (2012) e l'elencazione dei pacchetti necessari per comporre la tesi di dottorato dal titolo provvisorio «La "Città di vita" di Matteo Palmieri con le "Expositiones" di Leonardo Dati: edizione commentata del ms. XL 53 della Biblioteca Laurenziana di Firenze»[1] siamo venuti a conoscenza di due soluzioni a quei problemi per noi irrisolti. Non erano problemi irrisolti per la comunità TUG. In questo breve articolo descriveremo finalmente come possiamo trattare anche quei casi usando due pacchetti preesistenti all'articolo di cui sopra.

2 XeLaTeX e traslitterazioni

Come sappiamo, XeLaTeX è un pacchetto di macro in grado di leggere e comporre dei testi codificati Unicode. Per questo lo abbiamo sempre supposto incapace di comporre dei testi traslitterati: sarebbe un inutile esercizio di stile più che una necessità reale.

Contrariamente a quanto affermato in Pignalberi *et al.* (2011) anche XeLaTeX è in grado di comporre un testo greco traslitterato, basta usare il pacchetto `lgreek` (Levy e Murphy, 2010).

Il manuale a esso relativo mostra un documento minimale: nel preambolo c'è solo `\documentclass` e l'inclusione del pacchetto `lgreek`. Quindi c'è il corpo del documento. Qui il testo da comporre in greco è distinto dal testo normale perché è racchiuso nell'ambiente `greek`. Tutto il testo al di fuori di questo ambiente viene composto coi caratteri latini. La corrispondenza tra caratteri latini e greci è la stessa data nell'articolo Pignalberi *et al.* (2011) e non la ripeteremo.

Nella sezione "Accents and Breathings"[2] del manuale citato troviamo le regole da seguire per ottenere accenti e spiriti corretti e troviamo anche un errore che evidenzieremo a breve. Questa è la traduzione letterale della sezione di nostro interesse:

> Per ottenere un accento acuto, grave o circonflesso su una vocale digita rispettivamente `'`, `‘` o `~` prima della vocale. Per ottenere uno spirito aspro o dolce digita `<` o `>` prima della vocale (o rho) e di ogni suo eventuale accento. Per ottenere uno iota sottoscritto digita `|` *dopo* la vocale. Una dieresi è rappresentata da `"`, e se è accompagnata da un accento essa può stare prima o dopo l'accento stesso.
>
> Per esempio, `>en >arq\~h| >\~hn <o l'ogos` dà ἐν ἀρχῇ ἦν ὁ λόγος. Ordinato, no?
>
> Anche gli accenti e gli spiriti sono composti per mezzo di legature: una vocale con uno spirito, un accento e uno iota sottoscritto, per esempio, è realizzata come una legatura di quattro caratteri. L'unica eccezione si ha quando uno spirito è seguito da un accento grave, nel qual caso la combinazione spirito + accento è composta come un `\accent` di TeX sulla vocale. Ciò significa che le parole contenenti queste combinazioni non possono essere sillabate in TeX standard; questo non è un problema perché, con l'eccezione dei rarissimi casi di crasi, tutte queste parole sono monosillabiche.

1. Il dottorando Fabrizio Crasta sta lavorando alla tesi la cui discussione è prevista per maggio 2013 presso l'Università di Firenze.

2. I lettori con scarse conoscenze di greco antico trovano un'interessante introduzione ai segni diacritici dell'alfabeto greco in Wikipedia (2012).

In principio era il Verbo, il Verbo era presso Dio e il Verbo era Dio.
᾽Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος.

FIGURA 1: Il primo versetto del Vangelo di Giovanni ottenuto mediante traslitterazione e composto con XeLaTeX e lgreek

L'errore menzionato sta nel secondo capoverso della traduzione. Il manuale dà erroneamente un testo traslitterato con il carattere di *escape* prima della tilde. Mentre il comando `\~` genera l'accento circonflesso in LaTeX, ciò non è più vero col pacchetto in esame. Se compiliamo l'esempio proposto otteniamo un errore proprio in corrispondenza delle sequenze `\~`. Leggendo attentamente il primo capoverso di LEVY e MURPHY (2010) qui tradotto non troviamo affermata da nessuna parte la necessità di far precedere la tilde da `\`, tutt'altro. Togliamo il backslash e la compilazione va a buon fine e l'accento viene posto correttamente.

Il codice relativo al primo versetto del Vangelo di Giovanni è il seguente:

```
\documentclass{minimal}

\usepackage{lgreek}

\begin{document}
In principio era il Verbo, il
    Verbo era presso Dio
e il Verbo era Dio.

\begin{greek}
>En >arq~h| >~hn <o L'ogos, ka'i
    <o L'ogos >~hn
pr'os t'on Je'on, ka'i Je'os >~
    hn <o L'ogos.
\end{greek}
\end{document}
```

e genera il testo visibile nella Figura 1. Oltre che con XeLaTeX, possiamo compilare il sorgente appena visto anche con LaTeX e pdfLaTeX.

Un altro metodo per risolvere il problema dei testi traslitterati con XeLaTeX, suggeritoci da Massimiliano Dominici in corrispondenza privata, è la scrittura di una mappa.

All'indirizzo `http://tex.stackexchange.com/questions/30829/how-to-insert-greek-with-ascii-keyboard-and-xetex-polyglossia` troviamo una possibile istanza, a cura di Enrico Gregorio e quindi molto dettagliata, del metodo propostoci da Dominici. La prima cosa da fare è copiare il file di testo `asciitogreek.map` sul proprio computer. Ne riportiamo di seguito le prime righe.

```
; TECkit mapping for ASCII Greek
    <-> Unicode characters

LHSName "TeX-text"
RHSName "UNICODE"

pass(Unicode)

; Class "letter" consists of
    letters and characters
    denoting accents
; in order to cope with "sigma
    finalis", i.e., A-Z a-z ' ' >
    < ~
; we don't need the double quote
    because it goes only after a
    vowel
UniClass[letter] = ( U+0041..U
    +005A U+0061..U+007A U+0060 U
    +0027 U+003E U+003C U+007E )


; ligatures from Knuth's
    original CMR fonts
U+002D U+002D <> U+2013 ; -- ->
    en dash
U+002D U+002D U+002D <> U+2014 ;
    --- -> em dash

; Greek (according to C. Beccari
    conventions)

; (-1)
U+0022 <> U+2019 ; APOSTROPHE
```

Bisogna poi convertire il file col comando

```
teckit_compile asciitogreek.map
```

ottenendo in output un file binario con lo stesso nome ma estensione `.tec`. Questo file è la mappa usata da fontspec (ROBERTSON e HOSNY, 2011) per il suo lavoro. Mentre è facile ottenere la maggior parte dei caratteri greci a partire dalla traslitterazione latina, non è immediato ottenere l'accento circonflesso. Sul sito citato troviamo un file di esempio per capire come usare correttamente la mappa. Ne diamo di seguito un esempio riadattato il cui risultato è visibile nella figura 2.

```
\documentclass{minimal}

\usepackage{fontspec}
\setmainfont{Palatino Linotype}
\usepackage{polyglossia}
\setmainlanguage{italian}
\setotherlanguage{greek}

\XeTeXinputnormalization=1

\newenvironment{transgreek}
  {\catcode'\~=12 \begin{
      otherlanguage}{greek}%
```

```
13     \addfontfeature{Mapping=
           asciitogreek}}
14   {\end{otherlanguage}}
15 \newenvironment{transgreek*}
16   {\catcode`\~=12 \begin{
         otherlanguage*}{greek}%
17    \addfontfeature{Mapping=
           asciitogreek}}
18   {\end{otherlanguage*}}
19
20 \newcommand{\texttransgreek}{\
       begingroup
21   \catcode`\~=12 \
         innertexttransgreek}
22 \newcommand*{\
       innertexttransgreek}[1]{%
23   \begin{transgreek*}#1\end{
         transgreek*}\endgroup}
24
25 \begin{document}
26 \textitalian
27 In principio era il Verbo, il
       Verbo era presso Dio
28 e il Verbo era Dio.
29
30 \begin{transgreek}
31 >En >arq~h| ~>hn <o L'ogos, ka`i
       <o L'ogos ~>hn
32 pr`os t`on Je'on, ka`i Je`os ~>
       hn <o L'ogos.
33 \end{transgreek}
34 \end{document}
```

Nello stesso documento possono coesistere parti di testo scritto in greco e parti di testo traslitterato perché il pacchetto lgreek non è antagonista di polyglossia

Con questo riteniamo chiuso l'argomento traslitterazioni in XeLaTeX. Nel prossimo paragrafo torniamo a lavorare con i testi codificati Unicode, quindi scritti sia con caratteri latini che greci. Rimandiamo alla lettura di Pignalberi *et al.* (2011) per ripassare come Emacs e Yudit permettono di digitare tali testi pur non avendo una tastiera dedicata o il sistema operativo regolato sull'uso delle tastiere diverse dall'italiana.

3 Traduzioni a fronte e note a piè di pagina

Anche il problema dei testi su colonne parallele con le note a piè di pagina posizionate male ha una soluzione. Il pacchetto paracol (Nakashima, 2011) serve a comporre testi paralleli su più colonne e risolve brillantemente il posizionamento delle note a differenza di parallel e ledpar.

L'algoritmo essenziale di impaginazione parallela con paracol, per comodità chiamato ae da qui in poi, è semplicissimo:

1. bisogna aprire l'ambiente paracol dopodiché...
2. ... è sufficiente scrivere il testo della prima colonna e le eventuali note a piè di pagina.
3. Al termine del testo bisogna usare un comando indicante il cambio di colonna...
4. ... seguito dal testo della successiva colonna.
5. Reiteriamo il procedimento dal passo 3 fin quando, dopo aver scritto il testo dell'ultima colonna, ...
6. ... chiudiamo l'ambiente paracol.

Diamo un esempio breve ma non banale in cui riproduciamo due stralci di bibbie con traduzione a fronte e relative annotazioni. Gli stralci sono posti su quattro colonne parallele. Le prime due colonne riproducono i primi cinque versetti del Vangelo di Giovanni e le relative note così come compaiono in Merk e Barbaglio (1993), le altre due colonne riproducono invece lo stesso brano ma tratto da Nestle *et al.* (1984). Possiamo vedere il risultato nella figura 3.

A pagina 15 vediamo il codice necessario a realizzare quanto mostrato. La tabella 1 mostra le corrispondenze tra il passo di ae e la riga o le righe del sorgente. Lo stesso codice fornisce inoltre un metodo semplice per numerare tutti i versetti e le eventuali relative note mediante l'uso di `\footnotemark`.

Il documento pone il comando `\footnote` all'inizio di un versetto per numerare il versetto stesso e aggiungere le relative note. Usare `\footnote` per numerare quei versetti che non abbiano però alcuna nota genererebbe delle righe numerate e vuote nelle note, cosa che farebbe decadere la qualità del documento finale e della sua lettura. Per simulare un comportamento adeguato abbiamo definito il comando `\blindfootnote`, di fatto un *alias* a `\footnotemark`. Questo comando non prende argomenti e serve solo a porre il numero di richiamo della nota nel punto del testo indicato. Non viene emessa alcuna nota a piè di pagina. Il contatore del numero di nota viene incrementato automaticamente così come avviene con `\footnote`. Manca un meccanismo per non mandare a capo ogni nota a piè di pagina, ma ciò esula dallo scopo di questo articolo.

4 Conclusioni

L'articolo ha presentato due pacchetti e un metodo per risolvere due problemi presentati come irrisolti nell'articolo Pignalberi *et al.* (2011). I due problemi in questione sono la composizione di testi in greco traslitterato con XeLaTeX e il posizionamento corretto delle note a piè di pagina nei testi con traduzione a fronte posti su colonne

In principio era il Verbo, il Verbo era presso Dio e il Verbo era Dio.
Ἐν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος.

FIGURA 2: Il solito brano ottenuto per traslitterazione tramite una mappa realizzata *ad hoc*

TABELLA 1: Tabella delle corrispondenze tra le righe del sorgente alla pagina 15 e l'algoritmo qui chiamato AE

Passo di AE	Righe del codice
1	27
2	28–44
3	45, 63, 73
4	46–62, 64–72, 74-95
6	96

parallele. Il pacchetto lgreek risolve completamente il primo problema così come lo risolve ridefinire una mappa, mentre il pacchetto paracol risolve il problema del posizionamento delle note comune ai pacchetti parallel e ledpar. paracol consente la presenza di più di due colonne parallele sulla stessa pagina. L'esempio a corredo dell'ultima soluzione fornisce un metodo semplice ma efficace, sebbene non completamente automatico, per numerare tutti i versetti della Bibbia e le eventuali relative note.

Riferimenti bibliografici

LEAL, J. e PIGNALBERI, G. (2012). *Edizioni Critiche. Guida alla composizione con il proprio computer*. TEXNOLOGIE. CompoMat, Rieti, Italia.

LEVY, S. e MURPHY, T. (2010). *Using Greek Fonts with LaTeX*.

MERK, A. e BARBAGLIO, G. (a cura di) (1993). *Nuovo Testamento. Greco e italiano*. EDB, Bologna, terza edizione.

NAKASHIMA, H. (2011). *Package paracol: Yet Another Multi-Column Package to Typeset Columns in Parallel*.

NESTLE, E., NESTLE, E. e ALAND, K. (a cura di) (1984). *Novum Testamentum Graece et Latine*. Deutsche Bibelgesellschaft, Stuttgart, 26ª edizione.

PIGNALBERI, G., SCHIRONE, S. e LEAL, J. (2011). «Introduzione agli strumenti per grecisti classici». *ArsTEXnica*, (11).

ROBERTSON, W. e HOSNY, K. (2011). *The fontspec package*.

WIKIPEDIA (2012). «Segni diacritici dell'alfabeto greco». URL `http://it.wikipedia.org/wiki/Segni_diacritici_dell%27alfabeto_greco`.

▷ Gianluca Pignalberi
`g dot pignalberi at alice dot it`

▷ Salvatore Schirone
`schirone at gmail dot com`

▷ Jerónimo Leal
Dipartimento di Storia della Chiesa
Pontificia Università
della Santa Croce
Piazza Sant'Apollinare 49
00186 Roma
`jleal at pusc dot it`

[1]In principio era il Verbo, il Verbo era presso Dio e il Verbo era Dio. [2]Egli era in principio presso Dio: [3]tutto è stato fatto per mezzo di lui, e senza di lui niente è stato fatto di tutto ciò che esiste. [4]In lui era la vita e la vita era la luce degli uomini; [5]la luce splende nelle tenebre, ma le tenebre non l'hanno accolta.

[1]*presso* tr (altra) rivolto a (cf. anche v. 2) | tr e Dio era il Verbo // e + art (= Iddio)…
[3]tr senza di lui fu fatta neppure una cosa sola (P^{75})] …fu fatto niente (P^{66} vg) / punteggiatura: qui si chiude la frase (P^{75} e prob P^{66}, Nestle-Aland) | tr di tutto ciò che è stato fatto] (come inizio frase) ciò che… // ciò + però
[4]*In lui era*] …è | *la vita era*] …è | *degli uomini* >
[5]*splende*] splendeva | tr nella tenebra, ma la tenebra non l'ha compresa] …non l'ha vinta

[1]Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. [2]οὗτος ἦν ἐν ἀρχῇ πρὸς τὸν θεόν. [3]πάντα δι᾽ αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. [4]ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. [5]καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

[1]και^{2} + LW Ny
[3]ουδε εν Ta^{ae} sycp ar gg sa Val Thph Hipp] ουδεν S^{*} D 1^{s} 139 71 Thdot $Ω_{Q}^{1}$ Eus^{l} | ἓν. Ta^{ein} C^{*} LW D Θ e a b f ff q syci sa Naass Val Her Thdot Thph Ptol Iϙ Cl Ωϙ Eus Ath Tert Hil Amb Aug LG | ο + δε b syc Naass Iϙ Ef(Val)
[4]ην^{1}] εστιν Ta^{1} S D a b c f ff z vg^{1} syc sa Naass Val Thdot^{1} Iϙ Cl^{1} Aug | ην^{2}] εστιν Ta b sycp sa | τ. ανθϙ. > B*
[5]εφαινεν Ef | κατελαβεν] vicit Ef

[1]In principio erat Verbum, et Verbum erat apud Deum, et Deus erat Verbum. [2]Hoc erat in principio apud Deum. [3]Omnia per ipsum facta sunt, et sine ipso factum est nihil; quod factum est [4]in ipso vita erat, et vita erat lux hominum, [5]et lux in tenebris lucet, et tenebrae eam non comprehenderunt.

[1]Εν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος. [2]οὗτος ἦν ἐν ἀρχῇ πρὸς τὸν θεόν. [3]πάντα δι᾽ αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο ⸂οὐδὲ ἕν⸃· ὃ γέγονεν·^{1} [4]ἐν αὐτῷ ζωὴ ⸂ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς ◻τῶν ἀνθρώπων⸣· [5]καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

[3] ⸂ουδεν $\mathfrak{P}^{66}$ $\aleph^{*}$ D f^{1} *pc*; Ir | · † - *et*$^{-1}$. $\aleph^{c}$ (Θ) Ψ 050^{c} f^{1-13} 𝔐 $sy^{p.h}$ bo ¦ *txt* $\mathfrak{P}^{75c}$ C D L W^{s} 050* *pc* b vg^{s} sy^{c} sa; Ir Tert Cl Or (*sine interp. vl incert.* $\mathfrak{P}^{66.75*}$ $\aleph^{*}$ A B Δ 063 *al*)
[4] ⸂εστιν ℵ D it sa?; Cl^{pt} Or^{mss} ¦ - W^{s} | ◻B*

FIGURA 3: Vangelo secondo Giovanni, versetti 1–5. Da sinistra a destra il testo è in italiano, in greco (testo e note tratti da MERK e BARBAGLIO (1993)), in latino e in greco (testo e note tratti da NESTLE *et al.* (1984)). La parte greca di quest'ultimo testo fa ampio uso dei segni di interpunzione inclusi nella tabella Unicode 2E00–2E7F (Supplemental Punctuation). Tali glifi non sono compresi in alcuno dei font usati per realizzare gli esempi. Li abbiamo approssimati al meglio delle nostre capacità con i comandi visibili alle righe 16–22 del sorgente alla pagina 15

```
\documentclass[a4paper]{article}

\usepackage{fontspec}
\setmainfont[Mapping=tex-text]{Palatino Linotype}
\usepackage{polyglossia}
\setmainlanguage{italian}
\setotherlanguage{greek}
\setotherlanguage{latin}
\usepackage{ledmac}
\usepackage{paracol}
\usepackage{latexsym}
\usepackage{amssymb}
\usepackage{relsize}
\usepackage[body={20cm,12cm}]{geometry}

\newcommand{\cmexchar}{\usefont{OMX}{cmex}{m}{n}\selectfont\char}
\newcommand{\rasm}{$\ulcorner$}
\newcommand{\lsb}{\raisebox{1.5ex}{\textsmaller[4]{\cmexchar'70}}\hskip-.1em}
\newcommand{\rsb}{\hskip-.1em\raisebox{1.5ex}{\textsmaller[4]{\cmexchar'71}}}
\newcommand{\rs}{$^\Box$}
\newcommand{\lrob}{\rotatebox{30}{\footnotesize\textbackslash}}
\newcommand{\rd}{\raisebox{.25ex}{.}}
\newcommand\blindfootnote{\footnotemark}

\begin{document}
\thispagestyle{empty}
\begin{paracol}{4}
\footnote{\emph{presso} tr (altra) rivolto a (cf. anche v. 2) | tr e Dio era il
Verbo // e + art (=~Iddio)\ldots}In principio era il Verbo, il Verbo era presso
Dio e il Verbo era Dio.
\blindfootnote{}Egli era in principio presso Dio:
\footnote{tr senza di lui fu fatta neppure una cosa sola
(P\textsuperscript{75}) ] \ldots{}fu fatto niente (P\textsuperscript{66} vg) /
punteggiatura: qui si chiude la frase
(P\textsuperscript{75} e prob P\textsuperscript{66}, Nestle-Aland) | tr di
tutto ciò che è stato fatto ] (come inizio frase) ciò che\ldots{} // ciò $+$
però}tutto è stato fatto per mezzo di lui, e senza di lui niente è stato fatto
di tutto ciò che esiste.
\footnote{\emph{In lui era} ] \ldots{}è | \emph{la vita era} ] \ldots{}è |
\emph{degli uomini} \textgreater{}}In lui era la vita e la vita era la luce
degli uomini;
\footnote{\emph{splende} ] splendeva | tr nella tenebra, ma la tenebra non
l'ha compresa ] \ldots{}non l'ha vinta}la luce splende nelle tenebre, ma le
tenebre non l'hanno accolta.
\switchcolumn
\begin{greek}[variant=ancient]
\footnote{και\textsuperscript{2} $+$ LW Ny}Ἐν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν
πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ Λόγος.
\blindfootnote{}οὗτος ἦν ἐν ἀρχῇ πρὸς τὸν θεόν.
\footnote{ουδε εν Ta\textsuperscript{ae} sycp ar gg sa Val Thph Hipp] ουδεν S*
D 1\textsuperscript{s} 139~71~Thdot Ωρ\textsuperscript{l}
Eus\textsuperscript{l} | ἔν. Ta\textsuperscript{ein} C* LW D Θ e a b f ff q
syci sa Naass Val Her Thdot Thph Ptol Ιρ Cl Ωρ Eus Ath Tert Hil Amb Aug LG | o
$+$ δε b syc Naass Ιρ Ef(Val)}πάντα δι᾽ αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο
οὐδὲ ἓν ὃ γέγονεν.
\footnote{ην\textsuperscript{1}] εστιν Ta\textsuperscript{1} S D a b c f ff z
vg\textsuperscript{l} syc sa Naass Val Thdot\textsuperscript{l} Ιρ
Cl\textsuperscript{l} Aug | ην\textsuperscript{2}] εστιν Ta b sycp sa | τ.
ανθρ. \textgreater{} B*}ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων.
\footnote{εφαινεν Ef | κατελαβεν] vicit Ef}καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ
ἡ σκοτία αὐτὸ οὐ κατέλαβεν.
\end{greek}
\switchcolumn
\begin{latin}
\blindfootnote{}In principio erat Verbum, et Verbum erat apud Deum, et Deus
erat Verbum.
\blindfootnote{}Hoc erat in principio apud Deum.
\blindfootnote{}Omnia per ipsum facta sunt, et sine ipso factum est nihil;
quod factum est
\blindfootnote{}in ipso vita erat, et vita erat lux hominum,
\blindfootnote{}et lux in tenebris lucet, et tenebrae eam non comprehenderunt.
\end{latin}
\switchcolumn
\begin{greek}[variant=ancient]
\blindfootnote{}Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς
```

```
ἦν ὁ λόγος.
\blindfootnote{}οὗτος ἦν ἐν ἀρχῇ πρὸς τὸν θεόν.
\footnote{\lsb ουδεν $\mathfrak{P^{66}} \aleph^*$ D
\emph{f}$^\mathfrak{1}$ \emph{pc}; Ir | \textsuperscript{:} \dag{} -
\emph{et}$^{:\mathfrak{1}}$. $\aleph^{\mathrm{c}}$ (θ) Ψ
050\textsuperscript{c} \emph{f}$^\mathfrak{1.13} \mathfrak{M}$
sy\textsuperscript{p.h} bo \textbrokenbar{} \emph{txt}
$\mathfrak{P}^{\mathfrak{75}c}$ C D L
W\textsuperscript{s} 050$^*$ \emph{pc} b vg\textsuperscript{s}
sy\textsuperscript{c} sa; Ir Tert Cl Or (\emph{sine interp. vl incert.}
 $\mathfrak{P^{66.75{\mathsmaller{\mathsmaller{*}}}}} \aleph^*$ A B Δ 063
\emph{al})}πάντα δι᾽ αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο \lsb οὐδὲ
ἕν\rsb\textsuperscript{:}. ὃ\linebreak γέγονεν$^{:\mathfrak{1}}$
\footnote{\rasm εστιν $\aleph$ D it sa?; Cl\textsuperscript{pt}
Or\textsuperscript{mss} \textbrokenbar{} - W\textsuperscript{s} |
\rs{}B$^*$}ἐν αὐτῷ ζωὴ \rasm ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς \rs{}τῶν\linebreak
ἀνθρώπων\lrob\rd{}
\blindfootnote{}καὶ τὸ φῶς ἐν\linebreak τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ
κατέλαβεν.
\end{greek}
\end{paracol}
\end{document}
```

LaTeX e le lingue romanze alpine

Claudio Beccari

Sommario

LaTeX può gestire molte lingue, attualmente 74 più diverse varianti. Alcune delle lingue gestibili da LaTeX sono usate da poche persone, altre da milioni. Questo articolo vorrebbe essere un primo passo verso l'estensione di LaTeX alle lingue romanze usate nelle regioni alpine d'Europa.

Abstract

LaTeX is used to typeset in a variety of languages: at the time of writing, it can handle 74 different languages (plus many variants), some of them used by very few people, some by millions. At the moment, there are no facilities for typesetting documents in the various more or less official Alpine Romance languages. This paper would be a first step in order to extend the LaTeX language capabilities to the various romance languages that are being used by large communities in the European Alps.

1 Introduzione

Le Alpi sono una barriera naturale fra varie nazioni, ma non lo sono state per le popolazioni che le hanno attraversate portando di qua e di là delle Alpi le loro tradizioni, i loro costumi e specialmente le loro lingue. Gli stessi antichi romani hanno conquistato le regioni alpine annettendo i territori e le popolazioni autoctone e, successivamente, estendendo loro la cittadinanza romana. In questo modo la lingua del potente divenne anche la lingua del sottomesso, come purtroppo è successo nei secoli fino ai giorni nostri; questo ha fatto estinguere le antiche lingue autoctone, ma ha trasferito molte delle loro caratteristiche nelle nuove lingue che si sono formate.

Ora procedendo da est a ovest sull'arco delle Alpi si parlano diverse lingue romanze che hanno avuto riconoscimenti ufficiali come il romancio in Svizzera, o come il friulano nella regione autonoma Friuli – Venezia Giulia e il ladino delle regioni Trentino – Alto Adige e del Veneto. Le province autonome di Trento e di Bolzano hanno stabilito leggi particolari per il plurilinguismo che viene praticato nei loro territori; immagino che anche la regione Veneto e la provincia di Belluno abbiano provveduto nello stesso modo.

Nel canton Grigioni in Svizzera c'è un marcato plurilinguismo che comprende non solo una comunità maggioritaria di lingua tedesca, circa il 68%, e una comunità di lingua italiana di circa il 10%, ma anche una varietà di parlate romanze non italiche che rappresentano le varietà di una lingua "comune", il romancio dei Grigioni, che è stata ricostruita a tavolino e che per legge è ufficiale in tutti i rapporti fra i cittadini e il governo cantonale e con quello federale.

Esisterebbero anche le lingue ticinese, lombarda e piemontese, di cui non parlerò in questo articolo, ma arrivando alle Alpi occidentali abbiamo ancora il franco-provenzale e l'occitano.

Il franco-provenzale è poco conosciuto e spesso viene confuso con il provenzale, ma è parlato da decine di migliaia di parlanti in tre nazioni: in Italia dal Piemonte occidentale e in tutta la Valle d'Aosta; in Svizzera nel Vallese; in Francia ad est delle Alpi nella Savoia, nel Delfinato e nella Franca Contea.

L'occitano si estende su un territorio vastissimo, che comprende il sud ovest del Piemonte, tutta la Francia meridionale per arrivare fino alla Val d'Aran in Spagna. Probabilmente l'occitano, fra queste lingue romanze minoritarie è quella parlata dal maggior numero di persone.

Fra le lingue citate l'occitano e il franco-provenzale hanno avuto uno sviluppo diverso dal francese da una parte e dall'italiano dall'altra e sotto certi aspetti grammaticali assomigliano più al francese e al catalano, mentre le lingue romanze minoritarie della Svizzera sud orientale e dell'Italia nord orientale hanno avuto uno sviluppo assolutamente staccato da ogni influenza dell'italiano come, sotto certi aspetti, hanno avuto invece le lingue ticinese, lombarda e piemontese. La differenza sostanziale è che la formazione del plurale dei nomi e degli aggettivi in italiano e nelle lingue minoritarie dell'areale italiano avviene mutando la vocale finale, se detti nomi ed aggettivi finiscono con una vocale, mentre nelle lingue romanze che accomunano i grigionesi, i ladini del Trentino – Alto Adige, e i friulani formano il plurale con l'aggiunta della 's'. Tolta questa caratteristica particolare devo dire che tutte le lingue romanze citate, dal friulano all'occitano, sono comprensibili per una persona di cultura e di lingua madre italiana, e i ceppi "ladini" da una parte, e "occitani" dall'altra appaiono come varietà di due lingue principali.

Certamente come persona di formazione tecnico-scientifica non ho molto da dire su queste lingue, salvo che riesco a leggerle tutte, comprendendo il 90% delle parole e ne riconosco delle similitudini che vanno al di là degli areali dove sono sviluppate.

In Italia esiste una legge dello Stato che tutela le lingue minoritarie. In Italia sono numerosissime:

oltre a quelle citate sopra, si va dallo sloveno, al tedesco, che nella provincia di Bolzano ha lo status di lingua ufficiale accanto all'italiano, al francese, che in Valle d'Aosta ha lo status di lingua ufficiale accanto all'italiano, alle lingue dei Walser del Piemonte e della Valle d'Aosta, al catalano di Alghero, all'albanese arbaresh in molte regioni del sud, al grecanico, al croato, e probabilmente ne ho dimenticata qualcuna. In Calabria esiste persino un'enclave occitana nella zona di Guardia Piemontese, la cui lingua prende il nome di gardiol.

Come ho detto, però, solo il tedesco e il francese hanno uno status ufficiale.

A livello regionale o provinciale tutte le regioni che insistono sull'arco alpino in Italia hanno provveduto alla emanazione di leggi che hanno valorizzato le lingue locali. In Piemonte esistono leggi regionali che forniscono supporto alla conservazione delle lingue locali e in particolare la Giunta Regionale ha avanzato una petizione all'UNESCO per il riconoscimento dell'occitano come patrimonio culturale dell'umanità. In Tentino – Alto Adige esistono disposizioni delle due provincie autonome per il supporto al ladino della Val Gardena (Bz) e delle Val di Non e Val di Fassa (Tn); in Friuli – Venezia Giulia esiste sia una legge regionale, sia quella della provincia di Udine per il sostegno al friulano. Della Svizzera ho già detto e il romancio ufficiale (rumantsch grischun) è riconosciuto come lingua federale.

In questo articolo mi riferirò al romancio svizzero (chiamato ladino in alcuni dei sotto areali dei Grigioni) e al friulano (chiamato anche ladino orientale); questa scelta riduttiva dipende dal fatto che solo il rumantsch grischun e il furlan hanno per legge una grafia ufficiale e riconosciuta dalle autorità locali; ma ricordiamo che, a parte la grafia di queste lingue, i parlanti possono comprendersi reciprocamente con maggiore o minore facilità.

Le differenze di grafia sono sostanziali; il rumantsch ha molti suoni rappresentati mediante trigrafi o tetragrafi imprestati dal tedesco, come si vede anche nel nome stesso della lingua dove compaiono "tsch" e "sch" con il medesimo valore fonetico che hanno in tedesco. Il furlan invece ha una grafia più simile a quella della lingua italiana, sia pure con l'uso della 'ç' e di alcuni digrafi, che non compaiono in italiano come "cj" e "gj", oltre all'uso diffusissimo dell'accento circonflesso per distinguere le vocali lunghe che in furlan hanno una rilevanza semantica.

Riduco ulteriormente il campo di azione: mi occuperò solamente del rapporto fra il romancio e LaTeX, ma mi riprometto di affrontare questa tematica anche per altre lingue alpine. Perché questa scelta riduttiva?

Per alcuni motivi, alcuni validi, altri molto meno: in primo luogo perché il romancio è una lingua nazionale, mentre il friulano ancora ha rilevanza solo regionale, benché sia parlato da un numero molto più grande di persone; in secondo luogo perché mi sono dovuto occupare di problemi di sillabazione per le lingue che usano l'elisione degli articoli e delle preposizioni articolate quando precedono una parola che comincia per vocale; infatti per conoscenza diretta fino a poco tempo fa conoscevo solo l'italiano, il francese, il catalano e il romancio come lingue che usavano questa particolarità grafica; ora so che questo uso riguarda di fatto tutte le lingue romanze tranne il portoghese, lo spagnolo/castigliano e il rumeno. In terzo luogo la grafia del romancio non fa uso di consonanti con diacritici e per questo mio primo approccio alle lingue romanze alpine questa caratteristica mi torna molto utile. Infine ho una grande simpatia per la Svizzera e i suoi abitanti di ogni lingua e parlata, e ammiro questa loro capacità di convivere malgrado le differenze linguistiche che riescono a superare, perché quando ci si vuol capire, ci si capisce con la buona volontà; nonostante le differenze hanno un senso della cittadinanza e dello Stato dal quale noi abbiamo molto da imparare. Li ammiro perché nella loro Confederazione hanno realizzato l'*Unità nella diversità*, che sarebbe il motto dell'Unione Europea, ma che noi cittadini dell'Unione non abbiamo ancora sentito come nostro. Per questi motivi sentimentali, ho quindi dato la preferenza al romancio.

Con il romancio mi sono trovato inizialmente in difficoltà ad applicare il piccolo strumento di correzione per superare i problemi dell'elisione vocalica quando ho scoperto che LaTeX ancora non è in grado di gestire il romancio; perciò mi sono dovuto creare l'intera struttura per la gestione di questa lingua.

2 Fonemi e grafemi del romancio

Il romancio si scrive con l'alfabeto latino e l'uso degli accenti grave e acuto su alcune vocali. L'insieme di fonemi e grafemi è mostrato nella tabella 1, con i fonemi rappresentati mediante i caratteri IPA (International Phonetic Alphabet)[1].

Come si vede la grafia del romancio non è strettamente fonetica, me le differenze rispetto alla grafia puramente fonetica sono molto simili a quelle dell'italiano e riguardano principalmente la pronuncia della 'c' e della 'g' diversa a seconda che siano seguite dalle vocali 'e' e 'i' oppure dalle altre vocali.

1. Con *pdflatex* i fonemi IPA sono accessibili mediante il pacchetto tipa, (FUKUI, 2004) fornito di una discreta documentazione; sfortunatamente questi caratteri, disegnati in accordo con la famiglia dei font Computer Modern, possono essere usati solo con questi font e non sono usabili con il font Latin Modern di questo articolo. Non è un problema eseguire gli adattamenti necessari, ma certamente è una cosa che non mi sarei aspettato. Con XeLaTeX e LuaLaTeX, che usano i caratteri OpenType, i grafemi sono tutti direttamente accessibili, sempre che il font di testo scelto come "main font" ne sia dotato.

Tabella 1: Corrispondenza fra fonemi e grafemi

Fonemi IPA	Grafemi
a, ɑ, aː, ə	a,à
e, eː, ɛ, ɛː	e, è, é
i, iː	i, ì
ɔ, ɔː	o, ò
o, oː, u, uː	u
aɪ	ai
aʊ	au
iə	ie
jeʊ	ieu
wa, waː, uːɑ	ua
k	c, q
kw	qu
tʃ,	c, ch, tg
ttʃ	tsch
χ	ch
g, dʒ	g, gh
ʎ	gl, gli
ɲ	gn
h	h
s	s, ss
ts	z
z	s
ʃ	s, sch
ʒ	s, sch
ʃtʃ	stg
j	i, j
b	b
d	d
f	f
l	l
m	m
n	n
p	p
r	r
v	v

Anche il suono della ‘c’ palatale viene rappresentato in modo diverso a seconda che sia all’inizio, al centro o alla fine della parola. Ma tutto sommato le differenze rispetto all’italiano non sono moltissime; molte dipendono dal fatto che le terminazioni consonantiche delle parole sono molto frequenti.

Pertanto questa grafia può dare qualche problemino per la sillabazione (vedi più avanti), ma non dà nessun problema per scrivere un testo con una tastiera svizzera o italiana.

3 Come LaTeX gestisce le lingue

LaTeX, come anche gli altri mark-up del sistema TeX, da XeLaTeX, a ConTeXt a LuaTeX, gestisce le lingue in due modi distinti.

3.1 La divisione in sillabe

In primo luogo il file di formato per ogni mark-up deve contenere le regole di sillabazione per ogni lingua da gestire. I file di formato sono file creati apposta su ogni macchina al momento dell’installazione del sistema TeX, oppure ricreati dall’amministratore della macchina, che contengono tutto il mark-up, le configurazioni, le regole di sillabazione delle varie lingue, e molto altro, direttamente tradotti nel formato interno di ciascun motore di composizione, in modo che il grosso delle funzioni che servono per comporre con i programmi del sistema TeX siano già immediatamente utilizzabili, senza bisogno di tradurle daccapo ogni volta che si lancia uno di questi programmi.

Le regole di sillabazione sono costituite da frammenti di parole, chiamati *pattern*, che contengono una sola lettera, due lettere, tre lettere, eccetera, dove a sinistra e a destra di ogni lettera è segnata una cifra che indica con il valore pari che la cesura in quella posizione è vietata, mentre con il valore dispari è permessa; ogni cifra passa la sua informazione specifica mediante il suo valore, quindi 0 indica una blanda proibizione di dividere mentre 1 indica un blando consenso alla divisione; i corrispondenti valori 8 e 9 indicano la massima proibizione o il massimo consenso a dividere. Se in una parola divisa in pattern se ne trovano alcuni che contengono alcune lettere nella stessa sequenza e contengono cifre diverse, prevale il valore della cifra più alta: per esempio, prendiamo il caso della ‘s’ pura e impura in italiano che non può mai essere divisa da quanto segue a meno che non si tratti di un’altra ‘s’ oppure che sia la seconda di due ‘s’ seguita da un’altra consonante (per esempio mass-mediologo); ecco allora che ci sarà un pattern ‘1m’ che dice che si può dividere prima della ‘m’, e un altro pattern ‘1s2’ che dice si può dividere prima della ‘s’ ma non dopo; poi c’è ‘2s3s’ per consentire la divisione della doppia ‘s’, ma c’è anche ‘s4s3m’ per dividere correttamente questa parola italiana composta con una radice inglese. Nel nesso ‘ssm’ i pattern di una lettera consentirebbero la cesura solo prima della prima ‘s’ ma non prima della ‘m’, perché il pattern della ‘s’ vieta di dividere dopo la ‘s’ stessa; i pattern con due lettere consentirebbero la cesura solo fra le due ‘s’, mentre il pattern con tre lettere vieta la cesura fra le due ‘s’ ma la consente fra la ‘s’ e la ‘m’; in definitiva, mettendo assieme i vari pattern i codici numerici di sillabazione corrisponderebbero a 2s4s3m e questa è la regola finale che consente la cesura solo fra la seconda ‘s’ e la ‘m’.

Questo è il meccanismo, ma è facile immaginare che per dividere in sillabe una intera lingua ci vogliano moltissimi pattern; per l’italiano ne occorrono circa 330 e virtualmente non producono mai errori di sillabazione; per l’inglese americano ne occorrono un po’ meno di 5000 ma circa il 10%

delle cesure di tutte le parole americane sarebbero divise in modo scorretto[2] (parole di Liang e Knuth); per il tedesco ne occorrono un numero enorme che supera le 10 000 unità. Perché LaTeX possa fare il suo lavoro per bene con questi grandi numeri di pattern è necessario che i pattern stessi siano memorizzati nel file di formato in strutture dati molto rapide da scorrere per la loro ricerca con i rispettivi valori numerici; per questo motivo essi devono essere predigeriti una volta per tutti da una versione "vergine" del motore di composizione (oggi quasi sempre *pdftex*, oppure *xetex* o *luatex*) per assemblarli in strutture chiamate *hash* perché LaTeX le possa analizzare con la massima rapidità. Assemblati gli hash di ogni lingua, il programma associa loro un numero univoco da legare al nome della lingua che verrà usato da babel o da polyglossia per selezionare le regole giuste per ogni lingua.

3.2 La lingua nella tipografia

In secondo luogo LaTeX deve provvedere a scrivere le parole fisse nella lingua giusta, deve scegliere l'hash giusto per la lingua selezionata, deve realizzare le strutture tipografiche particolari di ogni lingua: un semplice esempio: in italiano si possono usare le virgolette alte o quelle uncinate, ma non deve essere lasciato nessuno spazio fra le virgolette e quanto esse racchiudono: "parola" o «parola»; in francese si usano solo le virgolette uncinate con uno spazio fra ognuna di esse e il loro contenuto: « mot »; in tedesco si usano sia quelle a forma di piccole virgole diritte alte o basse, ma quelle di apertura sono al piede, non sono alzate: „Wort", oppure quelle uncinate a rovescio, rispetto a quello che si fa in Italia, e senza spazio: »Wort«.

Inoltre babel e polyglossia sono in grado di distinguere alcune varietà ortografiche, come per esempio in tedesco gennaio si chiama *Januar*, mentre in austriaco si chiama *Jänner*.

Questi sono pochi esempi, ma ovviamente le tradizioni tipografiche nazionali sono molto diverse anche fra nazioni che usano la stessa lingua. I pacchetti babel e polyglossia tengono conto di tutte queste varianti e della selezione delle lingue.

Un punto importante: i pattern non sono caricati da babel o da polyglossia; questo pacchetto sceglie solo fra gli hash *già presenti nel file di formato* e se non ne trova nessuno associato alla lingua specifica, usa la sillabazione americana che, evidentemente, non ha nulla a che vedere con la sillabazione delle lingue romanze, tanto per citarne un gruppo piuttosto ampio, ma nemmeno con le lingue scandinave o germaniche o slave; non parliamo delle lingue che si compongono con alfabeti diversi da quello latino.

Oggigiorno le distribuzioni ridotte del sistema TeX spesso fanno disperare i principianti perché non riescono a ottenere le cesure in fin di riga in modo corretto anche se hanno caricato babel con l'opzione di lingua giusta. Solo con le distribuzioni complete si hanno le cesure giuste per tutte le lingue che il sistema riesce a gestire; in caso contrario bisogna essere capaci di ricreare i file di formato con i pattern delle lingue che interessa usare; questa è una operazione non difficile, ma delicata, e richiede di procedere in modi diversi a seconda della distribuzione del sistema TeX di cui si dispone, a seconda che si vogliano creare i formati per il motore *pdftex* oppure per *xetex* oppure per *luatex*; spesso le distribuzioni dispongono di opportuni programmi di configurazione con i quali l'operazione diventa più semplice; tuttavia queste operazioni esulano dall'argomento di questo articolo.

4 File di descrizione della lingua e file di pattern

La funzione descrittiva della lingua esposta nella sezione precedente è svolta da (generalmente un solo) file con estensione `.ldf`. Invece i pattern possono richiedere diversi file, diversi perché devono essere usati solo con font aventi codifiche del tipo T1, oppure con codifiche UNICODE; c'è modo di costruire file che soddisfino entrambe le esigenze, ma la predisposizione dei pattern è piuttosto complessa.

4.1 Il file di descrizione della lingua

I file `.ldf` per babel hanno il nome formato con quello della lingua e con l'estensione indicata. I file per polyglossia hanno il nome formato agglutinando il nome della lingua con il prefisso `gloss-`; il nome della lingua è solitamente il nome in inglese e in lettere minuscole; per il romancio avremo perciò i file `romansh.ldf` e `gloss-romansh.ldf`.

La funzione del file `.ldf` nei due casi è sostanzialmente la stessa, ma mentre con babel la lingua è una opzione per il pacchetto, con polyglossia questo è un file di estensione autonomo che permette di definire alcuni comandi con i quali precisare le singole lingue da usare nel documento specificandone mediante opzioni anche alcune particolarità che sono disponibili grazie all'uso dei font OpenType. Non scenderò in questi dettagli; mi limiterò a descrivere il file `romansh.ldf` che contiene le definizioni importanti e assolutamente necessarie in ogni file di questo genere.

```
\LdfInit{romansh}{captionsromansh}%
\ifx\l@romansh\@undefined
    \@nopatterns{romansh}%
```

2. Le regole di sillabazione dell'inglese americano tengono conto dell'accento tonico delle parole: il nome *rècord* e il verbo *to recòrd* hanno la stessa ortografia ma diversi accenti tonici (che in inglese non si scrivono); a causa di questo fatto le divisioni sillabiche sono rispettivamente *rec-ord* e *re-cord*; LaTeX conosce l'ortografia, ma non la pronuncia quindi se divide correttamente una delle due parole sbaglia l'altra. In inglese le parole omografe ma non omofone sono moltissime, quindi LaTeX si trova in grande difficoltà.

```
4     \adddialect\l@romansh\l@italian\fi
5 \addto\captionsromansh{%
6   \def\prefacename{Prefaziun}%
7   \def\refname{Bibliografia}%
8   \def\abstractname{Recapitulaziun}%
9   \def\bibname{Index bibliografic}%
10  \def\chaptername{Chapitel}%
11  \def\appendixname{Appendix}%
12  \def\contentsname{Tavla dal cuntegn}%
13  \def\listfigurename{Tavla da las figuras}%
14  \def\listtablename{Tavla da las tabellas}%
15  \def\indexname{Register da materias}%
16  \def\figurename{Figura}%
17  \def\tablename{Tabella}%
18  \def\partname{Part}%
19  \def\enclname{Agiunta(s)}%
20  \def\ccname{Copia a}%
21  \def\headtoname{A}%
22  \def\pagename{pagina}%
23  \def\seename{vesair }%
24  \def\alsoname{vesair era}%
25  \def\proofname{Demonstraziun}%
26  \def\glossaryname{Glossari}%
27  }%
28 \def\dateromansh{%
29  \def\today{\ifcase\day\or 1.\else
30    ils~\number\day\fi~da~\ifcase\month\or
31    schaner\or favrer\or mars\or avrigl\or matg\or
32    zercladur\or fanadur\or avust\or settember\or
33    october\or november\or december\fi\space
34    \number\year}}%
35 \providehyphenmins{\CurrentOption}{\tw@\tw@}
36 \addto\extrasromansh{%
37  \babel@savevariable\clubpenalty
38  \babel@savevariable\widowpenalty
39  \babel@savevariable\@clubpenalty
40  \clubpenalty3000\widowpenalty3000%
41  \@clubpenalty\clubpenalty}%
42 \addto\extrasromansh{%
43  \babel@savevariable\finalhyphendemerits
44  \finalhyphendemerits50000000}%
45 \addto\extrasromansh{%
46  \lccode‘’=‘’ }%
47 \addto\noextrasromansh{%
48  \lccode‘’=0 }%
49 \ldf@finish{romansh}%
50 \endinput
```

Credo che siano opportuni alcuni commenti. Innanzi tutto la prima linea di codice serve per l'inizializzazione, ma fra le altre cose definisce `\captionsromansh` che serve a babel tra l'altro per controllare se il file sia già stato caricato.

Le righe da 2 a 4 verificano se esista un hash collegato alla lingua romancia; se un tale hash non esistesse, sarebbe necessario comunque dare un significato al comando `\l@romansh` associandolo all'hash di un'altra lingua come se ne fosse una varietà. Io ho preferito associarlo all'hash dell'italiano, perché sulla mia macchina lavoro con una distribuzione completa del sistema TEX; se questo file fosse usato in una distribuzione di base, non è detto che l'hash collegato all'italiano sia stato generato, e quindi una associazione come quella che ho fatto io potrebbe dare origine ad errori; per evitarli sarebbe necessario eseguire un altro test; data la provvisorietà di questa descrizione della lingua romancia ho preferito omettere questo lievissimo perfezionamento.

Le successive righe dalla 5 alla 27 definiscono tutte le parole fisse che compaiono nei titolini o in certi ambienti; ho preso questi nomi dal vocabolario tedesco–romancio, messo a disposizione on line dalla Lia Rumantscha (la Lega Romancia, una istituzione grigionese per la promozione del rumantsch grischun); in parte ho preso queste traduzioni da libri in romancio che ho trovato in rete.

La data in romancio è definita nelle righe da 28 a 34; come si vede ci vuole l'articolo plurale prima dei numeri dei giorni superiori a 1, mentre per il primo del mese è necessario indicare l'ordinale secondo la tradizione tedesca, cioè con la cifra seguita dal punto ma senza articolo (non lo si scrive ma viene detto a voce quando si parla: *l'emprim da schaner 2012*).

Le righe da 35 a 44 impostano alcuni parametri per la sillabazione; la sillaba iniziale e quella finale non devono contenere meno di due caratteri; siccome moltissime parole romance terminano con gruppi di consonanti, tocca ai pattern provvedere a che l'ultima sillaba contenga almeno una vocale.

Inoltre sono impostate le penalità per le righe vedove e le righe orfane; il valore 3000 è abbastanza alto ma non infinito, per cui le righe orfane e vedove potranno essere presenti ma *pdflatex* cercherà di evitarle con sufficiente determinazione. Il valore enorme assegnato a `\finalhyphendemerits` serve per evitare che la penultima riga di un capoverso subisca la cesura in fin di riga; in questo modo l'ultima riga di un capoverso non comincia (quasi) mai con una frazione di parola.

Infine le righe da 45 a 50 impostano e disimpostano la condizione per la quale l'apostrofo possa essere considerato come parte di una parola in romancio e non lo sia più quando si cambia lingua. Le ultime due righe completano le impostazioni per il romancio e finiscono il file; dopo `\endinput` potrebbe esserci scritta qualunque cosa, ma questa parte del file non verrebbe nemmeno letta dal motore di composizione.

4.2 I file per la sillabazione

I file per generare gli hash di sillabazione sono tre: `loadhyph-rm.tex`, `hyph-quote-rm.tex` e `hyph-rm.tex`; `rm` è la sigla del romancio secondo le norme ISO; anche gli altri file destinati allo stesso scopo per le altre lingue hanno i tre file con i nomi strutturati nello stesso modo, salvo che `rm` è sostituito da `it` per i file destinati all'italiano, `de` per i file destinati al tedesco, eccetera[3].

3. Ci sono alcune eccezioni, ma in sostanza questa è la convenzione generale per dare un nome ai file dei pattern. Invece in greco, per esempio, esiste `loadhyph-grc.tex` per caricare i pattern per il greco classico validi se si usano i font codificati con la codifica di default `LGR`; mentre esiste un altro file che carica i corrispondenti pattern se si usano i font codificati in "BETA code"; le due codifiche si distinguono perché nella prima i segni diacritici sono premessi alle lettere a cui si riferiscono, mentre con la seconda i segni diacritici

Il primo file carica gli altri due quando si devono creare o ricreare i file di formato, al momento di generare gli hash per la sillabazione, non durante la composizione di un particolare testo. Il secondo contiene le impostazioni per la codifica dell'apostrofo (ma potrebbe contenere anche altre macro di utilità per scrivere i pattern); infine il terzo file contiene i pattern veri e propri.

Ho provato innanzi tutto a non generare il formato con i pattern per il romancio ed ho usato quelli italiani al loro posto come indicato nel paragrafo precedente. Mediante un mio programmino per scrivere tutte le parole di un testo spezzate in sillabe, ho verificato come funzionassero i pattern italiani per il romancio; evidentemente c'erano degli errori ma erano relativamente pochi; quasi tutti riguardavano le terminazioni consonantiche singolari e plurali del romancio, visto che in italiano normalmente tutte le parole terminano con una vocale e quindi l'italiano, in teoria, non dovrebbe prevedere pattern per parole terminanti in consonante[4]. Questi erano errori evidenti perché l'ultima "sillaba" trovata dal motore di composizione, non era una sillaba secondo la grammatica, in quanto non conteneva nemmeno una vocale; anche gli altri pochi errori erano tali da isolare una "sillaba" priva di vocali.

Sono inoltre riuscito a trovare in rete la *Grammatica per l'instrucziun dal rumantsch grischun*, molto completa e ben fatta e lì ho trovato le regole per la divisione grammaticale in sillabe; sono molto simili a quelle per l'italiano finché sono enunciate a parole, ma diventano specifiche per il romancio in molti dettagli che hanno richiesto molte modifiche e correzioni rispetto ai pattern italiani, come descrivo più sotto. Le regole sono le seguenti:

- Ogni sillaba contiene una vocale o un dittongo o un trittongo; in romancio le vocali possono essere accentate; i dittonghi e i trittonghi sono solamente *ai, au, ie, ia, iu, ui, uo, ieu, uai*. Con lo stesso concetto dei pattern per l'italiano ho scritto i pattern per il romancio evitando di esplicitare le vocali, cosicché, eventualmente, non vengono divisi alcuni iati, ma non vengono commessi errori dividendo gruppi vocalici dove non è consentito.

- Ogni consonante semplice o digrafo o trigrafo o tetragrafo seguito da una vocale o da un dittongo o un trittongo fa sillaba con la vocale che segue. Analogamente non si dividono altri gruppi che formano "suoni semplici" (di fatto sono quasi tutti digrafi, trigrafi e tetra grafi): *ch, tg, gl, gn, qu, sch, stg, tsch* e, nei toponimi, *s-ch, dsch*.

- Le consonanti doppie si dividono fra le due sillabe adiacenti.

- Due consonanti diverse si dividono fra le sillabe adiacenti se la coppia di consonanti non può trovarsi all'inizio di una parola romancia; più precisamente, in presenza di un gruppo di due o più consonanti la divisione va fatta lasciando a destra della cesura il massimo numero di consonanti che possano trovarsi all'inizio di una parola.

- La regola precedente implica quindi che a destra della cesura si può cominciare con *bl, cl, fl, pl, vl, br, cr, dr, fr, gr, pr, tr, vr*, ma non si può lasciare a destra della cesura *dl, sl, tl, sl, sr*; non dispongo di un *pledari* (dizionario) romancio, quindi non so se esistano delle parole che cominciano per *sl, sr*, che invece in italiano esistono; quindi i relativi pattern devono essere aggiustati a seconda della lingua; per esempio *pasler* si divide fra la 's' e la 'l'. In tutti gli altri casi, anche se una consonante è seguita da una liquida, *l, r*, la divisione avviene fra le due consonanti, al contrario dell'italiano; per esempio il nome della Svizzera, *Svizra* si divide fra la 'z' e la 'r'.

- La 's' seguita da una delle consonanti o digrafi *c, d, p, t, tg* non consente la cesura dopo la 's'.

- I prefissi si dividono etimologicamente al contrario dell'italiano dove la sillabazione fonetica è sempre lecita, anche se non è vietata la sillabazione etimologica; bisogna però tenere presente se dopo la preposizione-prefisso la parola che segue comincia con una consonante o con una vocale; nel primo caso la cesura cade fra la preposizione e la parola, mentre nel secondo caso si fa lo stesso solo se la parola esiste anche isolata. Per esempio il prefisso *ex* nella parola *examinar* non viene diviso etimologicamente perché la parola *aminar* non esiste isolata. Dunque per poter eseguire le divisioni etimologiche bisogna avere il dizionario a disposizione, cosa che LaTeX non può fare.

- Nello stesso modo occorre eseguire la divisione etimologica nelle parole composte evitando di dividere le parole componenti; questo LaTeX lo consente inserendo a mano il comando per la cesura facoltativa `\-`, comando che impedisce

sono aggiunti dopo la lettera a cui si riferiscono. Altri pattern greci invece di `gr` contengono la sigla `el`.

4. Questa leggenda metropolitana sopravvive da tempo immemorabile; certo la stragrande maggioranza delle parole termina in vocale, questo è vero, ma l'italiano accetta non solo l'elisione ma anche l'apocope che può lasciare le parole monche e quindi anche terminanti in consonante; inoltre esistono una quantità di termini scientifici e tecnici, sportivi, architettonici, medici, eccetera che terminano in consonante o perché prestiti assimilati da lingue straniere, o perché nati così come voci dotte greche o latine (straniere fino ad un certo punto, almeno quelle latine); esiste certamente almeno una parola italiana che termina con ciascuna delle 21 consonanti dell'alfabeto latino! E i pattern per l'italiano prevedono tutte queste terminazioni consonantiche.

la divisione negli altri punti della stringa che costituisce la parola composta.

- È consigliato di non dividere gli iati, ma la divisione è consentita nella divisione di parole composte, come *extra-ordinari, co-operativa*, eccetera. In questi casi i pattern per il romancio che ho predisposto prevedono la divisione al margine della separazione delle due parole componenti, ma non ne impediscono la divisione. Se necessario, si può intervenire a mano con `\-`.

- Le necessità tipografiche, non la grammatica, richiedono di non dividere dopo la prima sillaba o prima dell'ultima sillaba di una parola se questa prima o ultima sillaba è formata da una sola vocale.

- È vietato andare a capo dopo l'apostrofo, esattamente come in italiano.

Con queste regole ben esplicitate nella grammatica romancia, (Caduff *et al.*, 2009), ho creato i pattern per il romancio semplicemente prendendo i pattern per l'italiano che già facevano abbastanza bene i loro lavoro all'interno delle parole e ho aggiunto i pattern specifici del romancio, correggendo eventualmente quelli dell'italiano, quando erano in contrasto con le regole enunciate sopra. I pattern totali sono saliti a 378, una cinquantina in più di quelli per l'italiano. Ho rigenerato i file di formato e sulla mia macchina ora posso scrivere in romancio senza problemi.

Ecco ora un breve testo in romancio tratto dalla suddetta Grammatica[5]:

> En rumantsch n'èn las reglas da comma betg uschè strictas sco p.ex. en tudestg. Sco regla generala vala: nua ch'ins vul far ina pausa en la frasa, mett'ins ina comma. La comma è en rumantsch per ordinari il segn d'ina pitschna pausa e betg il segn d'ina disposiziun sintactica sco en tudestg. Las reglas da comma dependan damai pli ferm da quai ch'ins vul exprimer. Ellas han surtut la funcziun d'inditgar la melodia, il ritmus e las pausas ed èn tras quai in agid per chapir meglier il text. Tuttina n'è il diever da la comma betg arbitrar; i dat er contexts, nua ch'ella è fixa.

5. In romancio le regole della virgola non sono così stringenti come in tedesco. Come regola generale vale la seguente: dove si vuol fare una pausa nella frase mettiamo una virgola. La virgola in romancio serve per indicare il segno di una piccola pausa e non il segno di una costruzione sintattica come in tedesco. Le regole della virgola dipendono di più da ciò che si vuole esprimere; esse hanno soprattutto la funzione di indicare la melodia, il ritmo e le pause con le quali capire meglio il testo. Tuttavia l'uso della virgola non è arbitrario; ci sono dei contesti dove la virgola è fissa.

5 Conclusione

Conto di inviare i file di gestione del Romancio ai due team che si occupano di sillabazione e di descrizione delle lingue; un team si occupa di babel e l'altro di polyglossia; con il secondo team credo che non ci sano problemi di sorta; con il primo, invece, credo che ce ne saranno, perché a me appare inattivo: non so se sia solo una impressione mia o se si tratti di un fatto concreto.

Chiaramente sarebbe desiderabile che il sistema TEX sia in grado di gestire il romancio nel giro di poche settimane, invece che fra qualche anno; ma bisogna rimettersi alle tempistiche del lavoro volontario dei membri dei due team.

Per me è stato un esercizio molto interessante e mi ha dato la possibilità di arricchire le mie conoscenze linguistiche in campi che non avrei mai pensato di esplorare; la descrizione in questo articolo, benché il romancio sia solo una curiosità per la maggior parte dei lettori italiani, ha un valore didattico perché permette di rendersi meglio conto di chi fa che cosa fra i vari elementi che costituiscono il sistema TEX.

Riferimenti bibliografici

Braams, J. (2011). *Babel, a multilingual package for use with LATEX's standard document classes*. TUG. Leggibile con `texdoc babel` nella distribuzione TeX Live.

Caduff, R., Caprez, U. N. e Darms, G. (2009). *Grammatica per l'instrucziun dal rumantsch grischun*. Seminari da rumantsch da l'Universitad da Friburg. Dipartament da lingugatgs e litteraturas romanas, Friburg, CH. URL `http://www.unifr.ch/rheto/documents/Gramminstr.pdf`.

Charette, F. (2011). *Polyglossia: a Babel replacement for XELATEX*. TUG. Leggibile con `texdoc polyglossia` nella distribuzione TeX Live. L'attuale versione è affidata alla manutenzione di Arthur Reutenauer.

Fukui, R. (2004). *TIPA manual*. Graduate School of Humanities and Sociology, Tokyo University, Tokyo. Leggibile con `texdoc tipa` nella distribuzione TeX Live.

Liang, F. M. (1983). *Word Hy-phen-a-tion by Com-puter*. Tesi di Dottorato, Department of Computer Science, Stanford, CA. URL `www.tug.org/docs/liang/liang-thesis.pdf`.

▷ Claudio Beccari
Villarbasse
`claudio dot beccari at gmail dot com`

Scrivere un pacchetto per LaTeX3
Il pacchetto kantlipsum

Enrico Gregorio

Sommario

Il pacchetto kantlipsum viene usato come pretesto per illustrare alcune tecniche di programmazione con il linguaggio LaTeX3.

Abstract

The kantlipsum package is used as pretext to illustrate some programming techniques with the LaTeX3 language.

1 Introduzione

No, LaTeX3 non è ancora uscito. Ma è già disponibile e in uno stato piuttosto avanzato il livello più basso, quello della *programmazione*. Uno dei criteri che guidano gli sviluppatori è infatti la divisione più netta possibile fra i vari livelli: programmazione, produzione di interfacce con l'utente e livello utente. Alla fine i documenti LaTeX3 non saranno tanto diversi dagli attuali, ma tutti i comandi come `\section` saranno ridefiniti in termini di strutture del livello intermedio che a loro volta saranno basate sul livello più basso.

L'esperienza di LaTeX 2_ε permetterà di definire meglio tutti i comandi per gli utenti, evitando le acrobazie che hyperref o altri pacchetti devono eseguire per intervenire su comandi definiti quando, per esempio, degli ipertesti non si aveva la minima idea.

Con il livello intermedio completo non saranno più necessari pacchetti come titlesec, caption o geometry, perché le interfacce di accesso ai titoli, alle didascalie o alle impostazioni di pagina saranno molto più maneggevoli di ora. O forse ci sarà titlesec3 che avrà un compito più facile, potendosi appoggiare a un'interfaccia notevolmente più semplice da gestire.

Il livello più basso, quello della programmazione, è *molto* diverso dall'attuale, ma anche molto più ricco: non si deve inventare l'acqua calda ogni volta; forse a prezzo di scottarsi all'inizio.

Nel pacchetto che descriveremo viene solo sfiorata la superficie della ricca dotazione di funzioni messe a disposizione da LaTeX3. La distribuzione contiene la descrizione commentata di tutte le funzioni nel file `interface3.pdf`.

2 I pacchetti lipsum e kantlipsum

Qualche anno fa il pacchetto lipsum permise di evitare di inventarsi testi fasulli per scrivere esempi o esaminare pagine complete. Il comando `\lipsum` compone sette capoversi di finto latino, il famoso *Lorem ipsum dolor sit amet* che è in uso fra i tipografi da lungo tempo. Con `\lipsum[1-150]` si compongono tutti i capoversi che sono a disposizione, ma naturalmente è possibile anche sceglierne di meno o uno solo con `\lipsum[2]` o quello che si vuole (spesso uso il secondo che è più corto del primo).

Un difetto di lipsum è che il testo pseudolatino non si comporta bene con le regole di sillabazione dell'inglese, producendo spesso divisioni di riga imperfette e relative *overfull box*.

Il libro *Dive into Python* include un programmino scritto in Python che produce capoversi in stile kantiano, imitando un programma per Mac OS dei tempi andati. Per la precisione, il primo sviluppatore aveva cercato di imitare lo stile della *Kritik der reiner Vernunft*, cioè la "Critica della ragion pura", uno dei più famosi trattati di Immanuel Kant.

L'idea di usare questo programmino per produrre testo fittizio circolava già e ne ho colto l'occasione per esercitarmi nella nuova sintassi di LaTeX3. Il primo passo è stato di produrre un po' di capoversi, per ora sono 164; ne vediamo un esempio:

> As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.

La sintassi è del tutto simile a quella di lipsum.

- `\kant` produce i primi sette capoversi;
- `\kant[3]` produce il capoverso numero 3;
- `\kant[11-42]` produce i capoversi dal numero 11 al numero 42.

Se si usa la variante asterisco `\kant*` (con l'eventuale argomento facoltativo come prima) non ci saranno interruzioni di capoverso alla fine di ciascun pezzo.

In realtà si può scambiare il significato delle due varianti dando l'opzione `nopar` al caricamento di kantlipsum, esattamente come accade in lipsum. Il nuovo pacchetto ha un'altra opzione, `numbers`, che appiccica all'inizio di un capoverso il numero, in modo da poter controllare meglio l'effetto:

> 2 • Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

sarebbe il risultato di `\kant[2]` con quell'opzione.

3 Cominciamo con LaTeX3

Vediamo la struttura di un pacchetto che usi la sintassi di LaTeX3. L'inizio richiede il caricamento di expl3 e l'annuncio del pacchetto:

```
\RequirePackage{expl3}
\GetIdInfo$Id: kantlipsum.dtx
  0.5 2011-12-05 12:00:00Z Enrico $
  {Dummy text in Kantian style}
\ProvidesExplPackage
  {\ExplFileName}{\ExplFileDate}
  {\ExplFileVersion}{\ExplFileDescription}
```

È simile a `\ProvidesPackage`; il comando `\GetIdInfo` permette di evitare di dover scrivere le informazioni su versione, data e così via in più posti.

Le opzioni possono ancora essere dichiarate come nel modo tradizionale, è probabile che LaTeX3 ne definisca uno leggermente diverso.

```
\DeclareOption { par }
  {
   \cs_set:Nn \kgl_star:
     { \c_space_tl }
   \cs_set:Nn \kgl_nostar:
     { \par }
  }
\DeclareOption{ nopar }
  {
   \cs_set:Nn \kgl_star:
     { \par }
   \cs_set:Nn \kgl_nostar:
     { \c_space_tl }
  }
\DeclareOption{ numbers }
  {
   \cs_set:Nn \kgl_number:n
     {
      #1\nobreakspace
      \textbullet\nobreakspace
     }
  }
\cs_new_eq:NN \kgl_number:n \use_none:n
\ExecuteOptions { par }
\ProcessOptions \scan_stop:
```

Già si vede qualcosa di bizzarro. La prima cosa da sapere è che nell'ambiente di programmazione di LaTeX3 gli spazi sono *ignorati*, quindi possono essere usati con generosità senza rischi. Se si desidera un vero spazio, al suo posto si scrive `~`, come vedremo poi nella sezione sui messaggi.

L'altra cosa strana sono i nomi dei comandi. In LaTeX3 ogni comando dice qualcosa di sé già nel nome. Per esempio, `\c_space_tl` dice di sé che è una costante e che è di tipo 'token list'. Non è altro che la vecchia conoscenza `\space`. Non si può usare `~` là perché comunque TeX ignora gli spazi espliciti dopo i nomi di comandi.

I comandi propri di kantlipsum contengono tutti il prefisso `kgl` (per 'Kant generator lipsum') che è propriamente il nome del *modulo*. Per esempio, `\cs_set:Nn` fa parte del modulo `cs` ('control sequences'). La convenzione non è seguita in modo rigorosissimo: le costanti 'universali' come `\c_space_tl` non hanno nome di modulo. Il suffisso finale è diverso: `_tl` e `:Nn`. Un suffisso come il primo è riservato a variabili e costanti; un suffisso che comincia con i due punti è riservato alle *funzioni*.

LaTeX3 cerca di distinguere rigorosamente tra variabili e funzioni; una variabile (o una costante) contiene un dato, una funzione esegue un certo compito. Il compito di `\cs_set:Nn` è di definire un'altra funzione. Il suffisso dice che tipo di argomenti prende la funzione. Nel caso specifico `N` indica un argomento di tipo 'macro' mentre `n` indica una lista di token tra graffe (che può essere anche, secondo le regole sintattiche di TeX, un solo token). La riga

```
\cs_set:Nn \kgl_star: { \c_space_tl }
```

definisce `\kgl_star:` come una funzione senza argomenti, la cui azione è di inserire uno spazio. In LaTeX tradizionale avremmo scritto

```
\newcommand*{\kgl@star}{\space}
```

Notiamo che in LaTeX3 il carattere `@` non ha l'uso a cui molti sono abituati; come separatore per indicare comandi del livello più basso si usa infatti `_`.

Una sorpresa si ha con

```
\cs_set:Nn \kgl_number:n
  {
   #1\nobreakspace
   \textbullet\nobreakspace
  }
```

La funzione `\kgl_number:n` è quella che produce il numero, se viene scelta l'opzione `numbers`, oppure non fa nulla. La definizione tradizionale sarebbe stata

```
\newcommand*{\kgl@number}[1]{%
  #1~\textbullet~}
```

Un momento! Dov'è indicato il numero di argomenti nella nuova? Lo dice semplicemente il nome della funzione: `\kgl_number:n` è una funzione che prende un argomento e `\cs_set:Nn` prima di tutto esamina il *nome* della funzione da definire e ne imposta la definizione in modo opportuno.

I due punti appaiono solo nelle *funzioni* e ciò che segue è la *firma* (in inglese *signature*) della funzione stessa. Indica il numero e il tipo degli argomenti della funzione: `n` sta per un normale argomento tra graffe, `N` per un unico token simbolico.

Alla fine della serie di opzioni c'è la riga

```
\cs_new_eq:NN \kgl_number:n \use_none:n
```

che sarebbe stata

```
\let\kgl@number\@gobble
```

La differenza fra le famiglie di funzioni `\cs_set` e `\cs_new` è che quelle del secondo tipo controllano che il nome della funzione da definire non abbia già un significato; inoltre le funzioni `\cs_new` agiscono *globalmente*, mentre le `\cs_set` solo localmente. Come si vede dal suffisso, `\cs_new_eq:NN` prende come argomenti due funzioni; `eq` sta per 'equivalent': la prima è quella da definire, che viene resa equivalente alla seconda. Propriamente sostituisce `\global\let`, in questo caso non c'è differenza.

Dopo il classico `\ProcessOptions` c'è `\scan_stop:`, che è la vecchia conoscenza `\relax`, cioè una funzione non espandibile che non fa nulla. Segue `\RequirePackage{xparse}`, niente di strano.

Un'altra caratteristica di LaTeX3 è che rinomina *tutte* le primitive di TeX, minimizzando il rischio di conflitti. È ancora aperta la questione di quali saranno impiegabili ancora al livello utente; probabilmente parecchie, se non si vuole che molti documenti richiedano pesanti interventi per essere adattati alla nuova versione. Ma c'è molto tempo per pensarci.

4 Messaggi

La sezione successiva del pacchetto definisce i messaggi di errore o di avviso.

```
\msg_new:nnn {kantlipsum}
  {how-many}
  {The~package~provides~paragraphs~
   1~to~#1.~Values~outside~this~
   range~will~be~ignored.}
\msg_new:nnnn {kantlipsum}
  {already-defined}
  {Control~sequence~#1~already~defined.}
  {The~control~sequence~#1~is~
   already~defined,~I'll~ignore~it}
```

A ogni messaggio viene associato un nome; vedremo più avanti come vengono usati. Si noti che gli spazi vanno indicati con `~` ed eventuali fine riga con `\\` (non serve più `\MessageBreak`); ma i messaggi vengono mandati a capo in modo automatico e un `\\` serve raramente.

Il primo sarà un messaggio informativo (solo tre argomenti); il secondo di errore, perché ha i due tipici modi di avvisare l'utente: corto e lungo (che si legge premendo `h`). I messaggi possono leggere fino a quattro argomenti, in questo caso ciascuno ne ha uno.

5 Dichiarazioni di variabili e costanti

Una delle raccomandazioni degli sviluppatori è di dichiarare in anticipo le variabili e le costanti da usare in seguito. In questo pacchetto dichiariamo due variabili numeriche e una sequenza:

```
\int_new:N \l_kgl_start_int
\int_new:N \l_kgl_end_int
\seq_new:N \g_kgl_pars_seq
\seq_gput_right:Nx \g_kgl_pars_seq
  {This~is~the~Kant~lipsum~generator.}
```

Si tratta di due variabili intere (contatori, nel gergo usuale) e una sequenza, cioè una lista indicizzabile di elementi. Le sequenze cominciano da zero, perciò la inizializziamo con un elemento fasullo.

6 Comandi per l'utente

Qui faremo tesoro delle funzionalità di xparse che permette facilmente di definire comandi con varianti e con argomenti facoltativi.

```
\NewDocumentCommand{\kant}
  {s >{\SplitArgument{1}{-}}O{1-7} }
```

```
{
 \group_begin:
 \IfBooleanTF{#1}
   {
    \cs_set_eq:NN \kgl_par: \kgl_star:
   }
   {
    \cs_set_eq:NN \kgl_par: \kgl_nostar:
   }
 \kgl_process:nn #2
 \kgl_print:
 \group_end:
}
```

Con `s` dichiariamo che il comando `\kant` ha una variante asterisco e con `O` che ha un argomento facoltativo. La presenza dell'asterisco rende vero il condizionale `\IfBooleanTF{#1}`. Lo scopo è di definire `\kgl_par:` equivalente a `\kgl_star:` oppure a `\kgl_nostar:`. Questi ultimi riceveranno una definizione più avanti.

L'argomento opzionale ha, come valore di default, `1-7`. Nel caso non si voglia dare un valore di default, si può indicare l'argomento con `o` e, in tal caso, l'assenza dell'argomento opzionale rende vero il condizionale `\IfNoValueTF{#2}` (qui, perché l'argomento è il secondo nella lista). In genere un argomento opzionale viene indicato nel secondo argomento di `\NewDocumentCommand` con la sola `o` o con `O{...}`; in questo caso, però, diciamo anche a LaTeX3 di spezzare in corrispondenza del trattino (se c'è) quello che viene passato come argomento: questa è la funzione di `>{\SplitArgument{1}{-}}` che indica come va massaggiato l'argomento che segue. Nei tre casi possibili, ciò che verrà visto realmente come `#2` è:

- `{1}{7}` se l'argomento facoltativo non è espresso e quindi LaTeX usa quello di default;
- `{42}{\NoValue}` se si dà come argomento facoltativo `[42]` (o un altro numero qualsiasi;
- `{11}{42}` se si dà `\kant[11-42]`.

Lo stesso accade con `\kant*`, naturalmente. Il primo argomento di `\SplitArgument` dice quanti gruppi di parentesi graffe vogliamo (uno in più di quanto scritto) e il secondo qual è il token che usiamo come separatore; nel primo gruppo va la parte dell'argomento fino al primo trattino, nel secondo tutto il resto. Se non c'è trattino, il secondo gruppo conterrà `\NoValue`.

Nel primo caso le variabili intere `\l_kgl_start_int` e `\l_kgl_end_int` vengono impostate a 1 e 7; nel secondo caso viene chiamata la funzione `\kgl_process:nn` che ha *due* argomenti. Al termine viene eseguita la funzione `\kgl_print:`. Tutto ciò in un gruppo per mantenere locali i significati assegnati alle funzioni e i valori alle variabili. Naturalmente `\group_begin:` e `\group_end:` sono le vecchie conoscenze `\begingroup` e `\endgroup`.

Il pacchetto definisce anche `\kantdef`; lo lasciamo da parte, per ora.

7 Funzioni interne

Definiamo ora le funzioni che servono per la meccanica delle macro a livello utente. La prima è `\kgl_process:nn`.

```
\cs_new:Nn \kgl_process:nn
  {
   \int_set:Nn \l_kgl_start_int {#1}
   \IfNoValueTF{#2}
     { \int_set:Nn \l_kgl_end_int {#1} }
     { \int_set:Nn \l_kgl_end_int {#2} }
  }
```

Il suo scopo è di esaminare ciò che c'era nell'argomento facoltativo dato a `\kant` che sarà della forma `{42}{\NoValue}` o `{11}{42}`; LaTeX3 si trova davanti uno fra

```
\kpg_process:nn {42}{\NoValue}
\kpg_process:nn {11}{42}
```

In ogni caso la variabile (intera) `\l_kgl_start_int` viene impostata al valore dato dal primo argomento. La variabile `\l_kgl_start_int` viene impostata allo stesso valore nel primo caso, al secondo argomento altrimenti.

Sistemate le variabili, vediamo com'è definita la funzione `\kgl_print:`. Prima definiamo una funzione ausiliaria

```
\cs_new:Nn \kgl_use:n
  {
   \kgl_number:n {#1}
   \seq_item:Nn \g_kgl_pars_seq {#1}
  }
```

Essenzialmente è 'stampa il capoverso kantiano con il numero dato come argomento'. La funzione `\kgl_number:n` stampa il numero (se è stata data l'opzione `numbers`) oppure ingoia l'argomento e poi accede all'elemento con il dato numero della sequenza contenente i capoversi kantiani.

```
\cs_new:Nn \kgl_print:
  {
   \prg_stepwise_function:nnnN
     {\l_kgl_start_int}
     {1}
     {\l_kgl_end_int}
     \kgl_use:n
  }
```

La funzione fa avviare un ciclo. Il primo argomento di `\prg_stepwise_function:nnnN` è il valore iniziale, il secondo è il passo, il terzo è il valore finale, il quarto la funzione da eseguire, che deve

prendere un argomento. Per esempio, se i valori fossero 2 e 5, il risultato sarebbe equivalente alla successione di comandi

```
\kgl_use:n {2} \kgl_use:n {3}
\kgl_use:n {4} \kgl_use:n {5}
```

e quindi verrebbero stampati i capoversi kantiani dal 2 al 5.

Naturalmente `\kgl_print:n` non è strettamente necessaria: il codice avrebbe potuto essere inserito direttamente nella definizione di `\kant`. Ma gli sviluppatori di LaTeX3 insistono nel chiedere ai programmatori di spezzare il codice in moduli più piccoli e facilmente controllabili. I calcolatori moderni non sono molto rallentati da queste espansioni in più e la chiarezza è certamente un traguardo importante.

Ultima funzione: quella che definisce i capoversi.

```
\cs_new:Nn \kgl_newpara:n
  {
   \seq_gput_right:Nn \g_kgl_pars_seq
     { #1 \kgl_par: }
  }
```

Il capoverso, dato come argomento, viene inserito come nuovo elemento della sequenza globale che li conterrà tutti, con l'aggiunta alla fine di `\kgl_par:`. È facilissimo per la funzione `\kgl_use:n` accedere ai vari elementi della sequenza.

Questo non sarebbe stato possibile in LaTeX 2_ε, dove il concetto di sequenza non è presente. Non volendo definirsi tutta l'infrastruttura, si sarebbe potuto ovviare definendo, come fa lipsum, tante macro contenenti i vari capoversi. Per automatizzare la faccenda si sarebbe potuto usare un contatore e definire le macro con nomi come `\kgl@1`, `\kgl@2` e così via. Vediamo come:

```
\@tempcnta=\z@
\def\kgl@newpara#1{%
  \advance\@tempcnta\@ne
  \expandafter\def\csname kgl@%
    \number\@tempcnta\endcsname{%
    #1\kgl@par}%
}
```

Illeggibile.

8 I capoversi

Il codice prosegue con la definizione delle frasi kantiane.

```
\group_begin:
\char_set_catcode_space:n {`\ }

\kgl_newpara:n {As any dedicated reader
can clearly see, the Ideal of practical
reason is a representation of, as far as
I know, the things in themselves; as I
have shown elsewhere, the phenomena
...
Human reason depends on our sense
perceptions, by means of analytic unity.
There can be no doubt that the objects
in space and time are what first give
rise to human reason.}
```

⟨*altri 163 capoversi simili*⟩

```
\group_end:

\msg_info:nnx{kantlipsum}{how-many}
  {
   \int_eval:n
     {\seq_length:N \g_kgl_pars_seq - 1}
  }
```

Apriamo un gruppo e, per comodità, facciamo tornare lo spazio al suo significato usuale, per non dover scrivere `~` al suo posto nel testo. L'effetto di ciascuna delle funzioni `\kgl_newpara:n` è di aggiungere successivamente i capoversi alla sequenza `\g_kgl_pars_seq`. L'elemento numero 1 sarà

```
As any dedicated reader can clearly see,
...
give rise to human reason.\kgl_par:
```

L'azione di `\kgl_par:` è di emettere un comando `\par` se si è con l'opzione `par`, uno spazio altrimenti (ma la logica è rovescia se il tutto è prodotto con `\kant*`).

Alla fine emettiamo un messaggio informativo (solo nel file log): `\msg_info:nnx` prende come argomenti i due indicativi del messaggio definito all'inizio e un terzo argomento (espanso) che verrà inserito al posto di `#1` nel messaggio. Dobbiamo fare un piccolo calcolo perché la lunghezza della sequenza comprende anche l'elemento 0.

9 Il comando mancante

C'è un altro comando, definito più o meno solo per fare un esempio. Un utente del forum chiese come poter definire una macro che si espandesse a uno dei capoversi di lipsum. Perciò in kantlipsum è prevista la funzione.

```
\NewDocumentCommand{\kantdef}{mm}
  {
   \group_begin:
   \cs_set_eq:NN \kgl_number:n \use_none:n
   \cs_set_eq:NN \kgl_par: \prg_do_nothing:
   \cs_if_exist:NTF #1
     {
      \msg_error:nnx
        { kantlipsum }
        { already-defined }
        { \token_to_str:N #1 }
     }
     {
```

```
      \cs_new:Npx #1 { \kgl_use:n {#2} }
    }
  \group_end:
}
```

Usiamo ancora le funzionalità di xparse, definendo il comando con due argomenti obbligatori, indicati con `mm` ('mandatory'). Si apre un gruppo e lì si ridefiniscono `\kgl_number:n` come `\use_none:n` (cioè ingoiare l'argomento) e `\kgl_par:n` come `\prg_do_nothing:`, sinonimo di 'non fare nulla', ma espandibile (è il vecchio `\@empty`). A questo punto entra in azione un condizionale: `\cs_if_exist:NTF` controlla se il token dato come primo argomento è definito e se lo è esegue il secondo argomento, altrimenti il terzo. Perciò, se dicessimo `\kantdef{\kant}{1}`, ovviamente saremmo nel ramo 'vero' e si avrebbe il messaggio di errore che il comando è già definito e il secondo argomento sarebbe ignorato. Se invece diciamo `\kantdef{\pippo}{1}` (e supponiamo che `\pippo` non sia definito), verrà eseguito

```
\cs_new:Npx \pippo { \kgl_use:n {1} }
```

Qui usiamo un tipo ancora diverso di `\cs_new`. Infatti `\pippo` è un comando a livello utente e quindi non ha il suffisso che dice quanti argomenti prende. Dunque dobbiamo specificarlo come tutto ciò che va da `\pippo` alla graffa aperta: niente, quindi non ci sono argomenti. Ricordiamo che `\kgl_use:n {1}` contiene `\kgl_number:n` e `\kgl_par:`, ma siccome ne abbiamo neutralizzato le azioni, queste funzioni spariranno e resterà solo il testo.

Di fatto `\cs_new:Npx` è semplicemente `\long\xdef` con in più il controllo che il comando dato come primo argomento non sia definito.

Una piccola annotazione: abbiamo dovuto costruirci il messaggio di errore perché ancora non ci sono tutte le interfacce utente e se usassimo semplicemente `\cs_new:Npx` avremmo un errore ma il comando sarebbe *ridefinito* comunque.

▷ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
Enrico dot Gregorio at univr dot it

Font e TEX

*TUG**

Per principio TEX può usare qualunque font di cui abbia le metriche (larghezza dei caratteri, crenatura...) e le forme (al momento, in genere PostScript Type 1, TrueType oppure OpenType).

Un paio[1] di articoli notevoli sui concetti basilari dei font e dell'uso in LATEX:

- Essential NFSS, di Sebastian Rahtz;
- Font selection in LATEX, di Walter Schmidt;
- LATEX 2ε font selection reference, del LATEX team.

Per informazioni aggiuntive di ogni tipo:

- la nostra breve pagina di installazione dei font;
- la pagina di sezione dei font del TEX Web Resources;
- domande relative ai font nelle TEX FAQ;
- Free Font Resources for Free and Open Source Operating Systems di Stephen Hartke;
- pagina dei font di Nelson Beebe;
- articoli sui font in TUGboat.

La questione delle licenze dei font è tuttora oggetto di interminabili dibattiti e fonte di controversie. I font liberamente disponibili elencati oltre sono rilasciati sotto una certa varietà di licenze; alcune licenze comunemente usate per i font sono elencate separatamente.

Computer Modern: Entrando nello specifico, il primo e ancora prevalente font nel mondo di TEX è Computer Modern, sviluppato da Donald Knuth usando il suo originale programma METAFONT, distribuito insieme a TEX.

Altri Modern: Sono state create delle varianti di Computer Modern che comprendono virtualmente lettere e simboli di tutte le altre lingue basate sull'alfabeto latino. Tali caratteri sono stati anche convertiti in font PostScript Type 1, ecc. Latin Modern e cm-super sono due collezioni notevoli in quest'area.

PostScript di base: TEX supporta anche i 35 font standard PostScript: Times Roman, Helvetica, Courier, Palatino e altri (URW++ ha gentilmente donato questi font al pubblico nel 1996, e li ha anche resi disponibili sotto la LPPL nel 2009. Gli originali licenziati GPL sono disponibili sul sito Ghostscript).

Fate riferimento alla documentazione psnfss2e per l'uso in LATEX, e a `test1.tex` (e gli altri `test*.tex`) per ulteriori esempi di output e uso. In breve:

```
\usepackage{mathptmx}
  % times roman, including math
  % (where possible)
\usepackage{mathpazo}
  % palatino, including math
  % (where possible)
\usepackage{helvet} % helvetica
```

I font TEX Gyre realizzati da GUST Typefoundry estendono i 35 font PostScript di base a molti altri caratteri, analogamente alle estensioni del Latin Modern al Computer Modern.

Saggi di font

I font seguenti coprono la composizione di un'ampia varietà di lingue e alfabeti, così come della maggior parte della matematica. Sono disponibili documenti con saggi di font che mostrano come appaiono:

- A Survey of Free Math Fonts di Stephan Hartke mostra la maggior parte delle famiglie libere per testo e matematica disponibili all'uso con TEX;
- LATEX Font Catalogue di Palle Jørgensen del DK-TUG ha brevi esempi della maggior parte dei font tipicamente disponibili con le installazioni di LATEX;
- Mathematikschriften für LATEX di Walter Schmidt mostra molte combinazioni testo/matematica, sia libere che non;
- Free typeface sampler di Peter Flynn mostra molti dei font inclusi in TEX Live e in altre distribuzioni;
- Fonts in ConTEXt di Hans Hagen mostra esempi di testo dei font comuni (la maggior parte liberi, alcuni commerciali) e come specificarli in ConTEXt;
- Typographical comparison document di Hans Hagen mostra i font di TEX a grande dimensione in aggiunta alle informazioni contestuali e alle mini-biografie dei progettisti.

Font nelle distribuzioni TEX

Questi font sono inclusi nelle installazioni standard di TEX come TEX Live e MiKTEX e pronti da usare.

- Altri font dal GUST Typefoundry: Antykwa Toruńska, Antykwa Półtawskiego, Cyklop, Iwona, Kurier;

*Traduzione a cura di Gianluca Pignalberi, revisione di Massimiliano Dominici.

1. In realtà ne vengono elencati tre [NdT]

- Arkandis Digital Foundry ha creato diversi font liberi con le sembianze di Baskerville, Bodoni e altri. I file di supporto a (LA)TEX sono disponibili per la maggior parte dei casi;
- Bitstream ha donato la famiglia di font Vera (uno slab serif, un sans serif simil-Frutiger e un monospaziato) al pubblico. La versione sensibilmente migliorata per TEX è chiamata Bera. La variante arev serve specialmente per le presentazioni;
- Bitstream ha anche disegnato il carattere Charter, un serif liberamente disponibile;
- Gentium di Victor Gaultney. Gentium fa parte del SIL's font project;
- I font della Greek Font Society includono GFS Artemisia, Baskerville, Bodoni, Complutum, Didot, NeoHellenic, Porson e Solomos. Tutti supportano il greco, la maggior parte supporta anche il latino;
- Inconsolata è un font monospaziato che supporta la maggior parte degli alfabeti latini e inteso per essere massimamente compatibile con Computer Modern typewriter;
- Libertine è una famiglia libera di font con un'ampia varietà di forme e il supporto di diversi alfabeti (inclusi latino, cirillico, greco ed ebraico);
- I font matematici complementari a Garamond, Utopia e Charter sono disponibili presso il Math Design project;
- Il font Utopia serif è stato donato da Adobe. L'eccellente collezione Fourier-GUT è basata su Utopia.

Font liberi solo True/OpenType

Tali font non hanno la versione Type 1 e quindi sono più facilmente utilizzabili con XETEX e LuaTEX (alcuni dei font precedenti e successivi hanno anche le versioni OpenType e/o TrueType). Sono tutti inclusi nelle principali distribuzioni di TEX.

- GNU FreeFont fornisce le varianti serif, sans serif e monospaziate a supporto di un'ampia varietà di caratteri;
- The STIX Project ha rilasciato i font matematici OpenType per Unicode. Questo articolo su STIX di Barbara Beeton spiega alcuni *dietro le quinte* e problemi;
- XITS è un derivato esteso di STIX sviluppato da Khaled Hosny.

Per rimanere in argomento, l'articolo OpenType Math Illuminated di Ulrik Vieth confronta dettagliatamente i parametri dei font OpenType e matematici di TEX.

Alcuni font non liberi

Pochi ulteriori font sono disponibili per l'uso generale dei privati ma hanno alcune restrizioni che ne impediscono l'inclusione nelle distribuzioni libere di TEX:

- Garamond e Letter Gothic di URW, originariamente rilasciati come parte di GhostPCL. Altri font della medesima provenienza possono acquisire il supporto a TEX in futuro;
- LuxiMono di Bigelow & Holmes, una variante monospaziata di Lucida.

Lo script `getnonfreefonts` semplifica l'installazione di questi font e altri. Provate `getnonfreefonts --help`.

Font commerciali

Altri font standard per la normale composizione di testo (Baskerville, Bodoni e altri) sono sfortunatamente non liberi (al momento della stesura, per quanto ne sappiamo). Se siete interessati ad aiutare la produzione di più font liberi fate riferimento alla pagina web GNU Font Utilities e all'articolo di TUGboat *Making outline fonts from bitmap images* come punti di partenza, e fateci sapere.

Font commerciali eccellenti sono disponibili tramite molte aziende. Sfortunatamente quelle controllate (Adobe, Bitstream e altre) hanno prezzi alti per una libreria di font (migliaia di dollari). I prezzi per singolo font sono altrettanto alti. C'è una quantità di collezioni gratuite disponibili, di differente qualità e legalità, ma poiché nessuna di esse è legalmente utilizzabile ovunque nel mondo, il TUG non può, a malincuore, raccomandare specificamente alcuna di esse. I font commerciali non hanno di solito il supporto a TEX, cioè i file `.tfm`, ma molte metriche sono disponibili su CTAN; fate riferimento a Walter Schmidt's collection e a MinionPro TEX support.

Un font commerciale che possiamo raccomandare è MathTime Pro 2 di Michael Spivak, distribuito da Personal TEX, Inc. Questo ha la piena integrazione dei caratteri Times Roman per la composizione matematica con TEX, basata sui progetti originali. Può essere usato con ogni implementazione di TEX che supporti i font virtuali.

Infine, la famiglia di font Lucida, pur restando sotto licenza proprietaria, è disponibile tramite il TUG, sia nella versione Type 1 che OpenType.

Supportare lo sviluppo dei font liberi

Il TUG amministra il lato finanziario del Libre Font Fund, che supporta lo sviluppo dei font liberi (come in libertà, alias *libre*) e software correlati. Le donazioni al fondo sono fortemente apprezzate!

▷ TUG
`www.tug.org/fonts`

Questa rivista è stata stampata
presso Braille gamma S.r.l., Cittaducale (RI)
su carta Revive 100 uncoated natural 80 g
distribuita da Polyedra.

2012

meeting

Nono convegno nazionale su TEX, LATEX e tipografia digitale

Napoli, 27 ottobre 2012

Centro Congressi Federico II

Call for Paper

Sabato 27 ottobre 2012 a Napoli, presso il Centro Congressi dell'Università degli Studi di Napoli Federico II, si terrà il nono Convegno annuale su TEX, LATEX e tipografia digitale organizzato dal Gruppo Utilizzatori Italiani di TEX. Il Convegno sarà un momento di ritrovo e di confronto per la comunità LATEX italiana, tramite una serie di interventi atti sia a contribuire all'arricchimento sia a supportarne lo sviluppo.
Maggiori informazioni sul Convegno e sulle modalità di presentazione degli interventi sono disponibili all'indirizzo:

`http://www.guitex.org/home/meeting`

o possono essere richieste all'email:

`agostino.demarco@unina.it`

ArsTEXnica

Numero 13, Aprile 2012

20013
9 771828 236001